Star-Hspice User Guide

Release 2001.4
December 2001

Star-Hspice User Guide, Release 2001.4, December 2001
Previously printed June 2001, V2001.2

Contacting Avant! Corporation

Telephone:	(510) 413-8000
Facsimile:	(510) 413-8080
Toll-Free Telephone:	(800) 369-0080
URL:	http://www.avanticorp.com

Avant! Corporation
46871 Bayside Parkway
Fremont, CA 94538

Avant!

Using This Manual

This manual describes the Star-Hspice circuit and device simulation software and how to use it.

Audience

This manual is intended for design engineers who use Star-Hspice to develop, test, analyze, and modify circuit designs.

How this Manual is Organized

The manual set is divided into two volumes, as follows:

- Volume I (Chapters 1 through 13) describes how to run simulations with Star-Hspice and evaluate the results.
- Volume II contains detailed applications and examples of how to use Star-Hspice for a wide variety of circuit simulations (Chapters 14 through 22). Volume II also contains reference material (Appendices).

Related Documents

The following documents pertain to this guide:

- Star-Hspice, Star-Time, and AvanWaves Installation Guide
- Star-Sim and Star-Time User Guides
- Star-Hspice and AvanWaves Release Notes

If you have questions or suggestions about this documentation, send them to:

`techpubs@avanticorp.com`

Conventions

Avant! documents use the following conventions, unless otherwise specified:

Convention	Description
menuName > *commandName*	Indicates the name of the menu and the command name. For example: *Cell > Open* refers to the *Open* command in the Cell menu.
Tool: menuName > *commandName*	Indicates that a command is accessible only through an application tool. Tool is the tool through which you access the command, *menuName* is the name of the menu, and *commandName* is the name of the command. For example: *Data Prep: Pin Solution > Via* refers to the *Via* command on the Pin Solution menu, which you access by selecting *Data Prep* from the Tools menu in Apollo.
courier	In text, this font indicates a function or keyword that you must type exactly as shown. In examples, this font indicates system prompts, text from files, and messages printed by the system.

Convention	Description
courier italic	Arguments appear in this font when the value of an argument is a string. The string must be enclosed with quotation marks.
times italic	Indicates commands, functions, arguments, file names, and variables within a line of text. When a variable is included in italicized text, the variable is enclosed by angle brackets (<>). For example, “the name of the technology file is <libraryName>.tf, where <libraryName> is the name of the library.”
[ ]	Denotes optional arguments, such as: *pin1* [*pin2*, ...*pinN*] In this example, you must enter at least one pin name, the other arguments are optional.
({*instanceName orientation*} ...)	Indicates that you can repeat the construction enclosed in braces.
. . .	Indicates that text was omitted.

Convention	Description
’(item1 item2)	An apostrophe followed by parentheses indicate that the text within the parentheses enclose a list. When the list contains multiple items, the items are separated by spaces. Type this information exactly as it appears in the syntax.
|	Separates items in a list of choices. For example, on | off.
\	Indicates the continuation of a command line.

Obtaining Customer Support

If you have a maintenance contract with Avant!, you can obtain customer support by:

- Contacting your local Application Engineer (AE)
- Calling the Avant! Corporate office from 8:00 AM through 5:00 PM Pacific Standard Time (PST) at:

  `1-800-346-5953`

  Ask the receptionist for customer support.
- Emailing a description of the problem to the Hspice Support Center at:

  `hspice_nw@avanticorp.com`

Other Sources of Information

The Avant! external web site provides information for various products. You can access our web site at:

```
http://www.avanticorp.com
```

From our web site, you can register to become a member of the Avant! Users Research Organization for Real Applications (AURORA) user's group. By participating, you can share and exchange information pertaining to Integrated Circuit Design Automation (ICDA).

Avant!

Table of Contents

Avant!

Chapter 1
Introducing Star-Hspice

The Star-Hspice optimizing analog circuit simulator is Avant!'s industrial-grade circuit analysis product for the simulation of electrical circuits in steady-state, transient, and frequency domains. Circuits are accurately simulated, analyzed, and optimized from DC to microwave frequencies greater than 100 GHz.

Star-Hspice is ideal for cell design and process modeling and is the tool of choice for signal-integrity and transmission-line analysis.

This chapter covers the following topics:

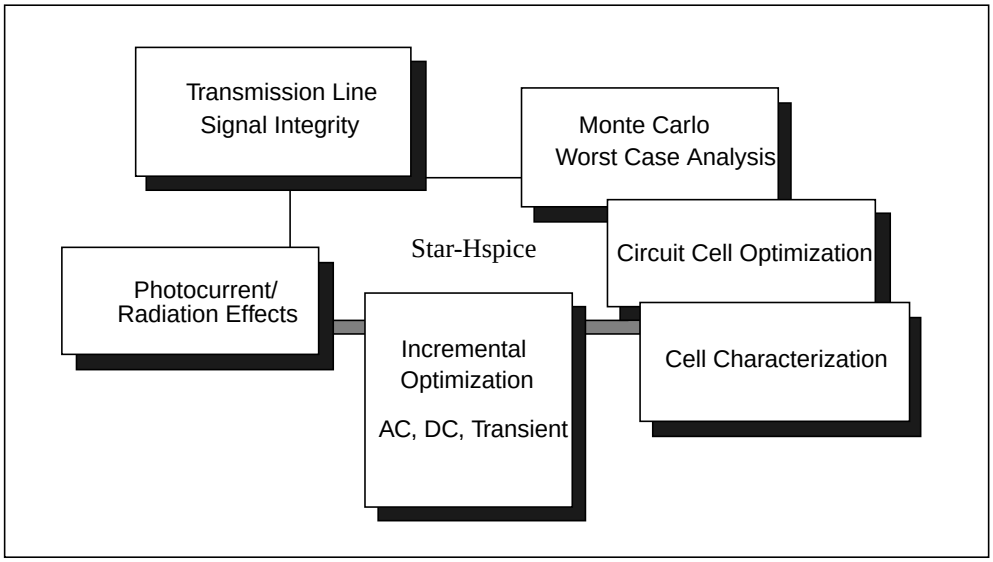
Star-Hspice Applications

Star-Hspice is unequalled for fast, accurate circuit and behavioral simulation. It facilitates circuit-level analysis of performance and yield utilizing Monte Carlo, worst case, parametric sweep, and data-table sweep analysis while employing the most reliable automatic convergence capability. Star-Hspice forms the cornerstone of a suite of Avant! tools and services that allow accurate calibration of logic and circuit model libraries to actual silicon performance.

The size of the circuits simulated by Star-Hspice is limited only by the virtual memory of the computer being used. Star-Hspice software is optimized for each computer platform with interfaces available to a variety of design frameworks.

Star-Hspice Features

Figure 1-1: Star-Hspice Design Features



Star-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 parameterizable cells
- Cell characterization tools for calibrating library models for higher-level logic simulators
- Geometric lossy coupled transmission lines for PCB, multi-chip, package, and IC technologies
- Discrete component, pin, package, and vendor IC libraries
- AvanWaves interactive waveform graphing and analysis from multiple simulations

Figure 1-2: Star-Hspice Circuit Analysis Types

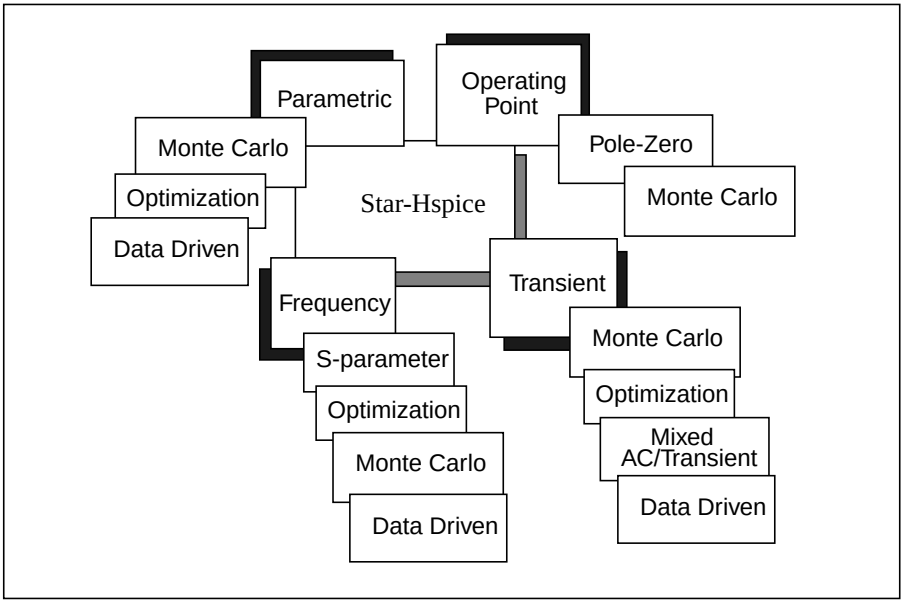
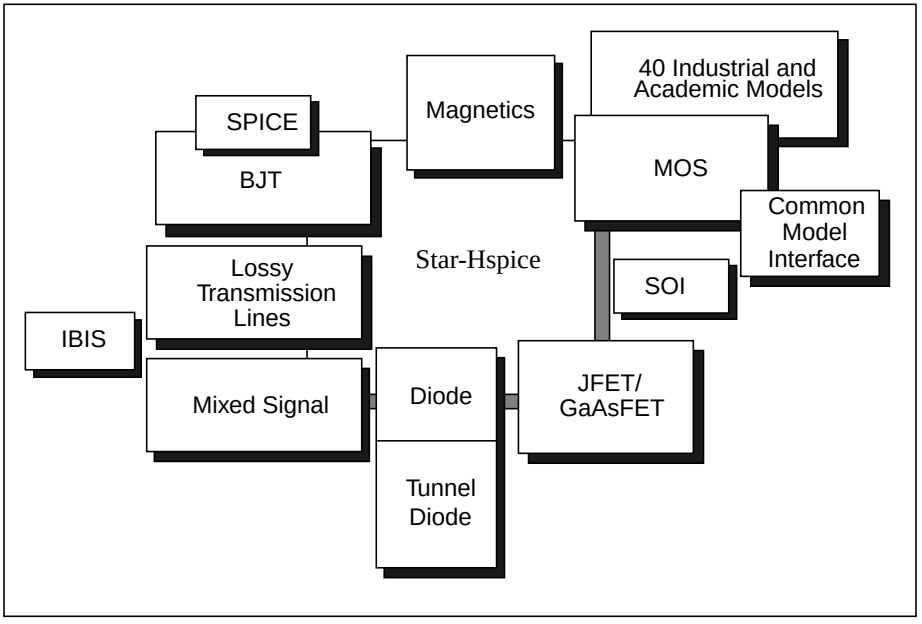


Figure 1-3: Star-Hspice Modeling Technologies



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

You can organize simulation circuits and models to run using the following Star-Hspice features:

- Explicit include files – .INC statement
- Implicit include files – .OPTION SEARCH = 'lib_directory'
- Algebraics and parameters for devices and models – .PARAM statement
- Parameter library files – .LIB statement
- Automatic model selector – LMIN, LMAX, WMIN, WMAX model parameters
- Parameter sweep – SWEEP analysis statement
- Statistical analysis – SWEEP MONTE analysis statement
- Multiple alternative – .ALTER statement
- Automatic measurements – .MEASURE statement

Star-Hspice Platforms

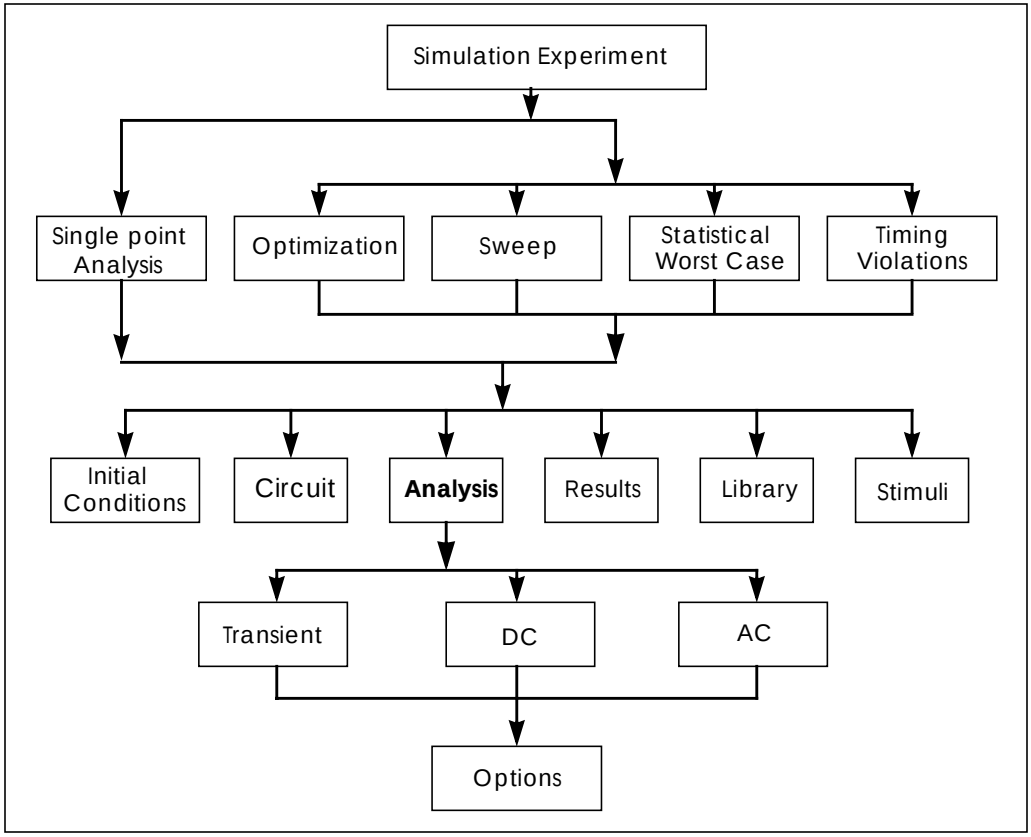
Star-Hspice is available for the following platforms and operating systems:

Platform	Operating System
Sun Ultra	Solaris 5.5, 5.7 and 5.8
Sun Sparc	Solaris 5.5
Sun Blade	Solaris 5.8
HP PA	UX 10.20, UX 11.00
IBM RS6000	AIX 4.3
DEC Alpha	OSF 4.0
SGI	IRIX 6.5
PC	Windows 95, 98, ME, 2000, NT 4.0, and XP.
Linux	RedHat 6.2, 7.0/7.1 (Does not support MOSFET level 29 and level 45).
Note: Star-Hspice supports a single AMD CPU for WinNT4.0, and RedHat 7.0/7.1	

Examining the Simulation Structure

Figure 1-4 shows the program structure for simulation experiments.

Figure 1-4: Simulation Program Structure



Analysis and verification of complex designs are typically organized around a series of experiments. These experiments are simple sweeps or more complex Monte Carlo, optimization, and setup and hold violation analyses that analyze DC, AC, and transient conditions.

For each simulation experiment, tolerances and limits must be specified to achieve the desired goals, such as optimizing or centering a design. Common factors for each experiment are process, voltage, temperature, and parasitics.

Two terms are used to describe experimental methods using Star-Hspice:

- Single point – a single point experiment is a simple procedure that produces a single result, or a single set of output data.
- Multipoint – an analysis (single point) sweep is performed 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, and 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 for analog circuits

Understanding the Data Flow

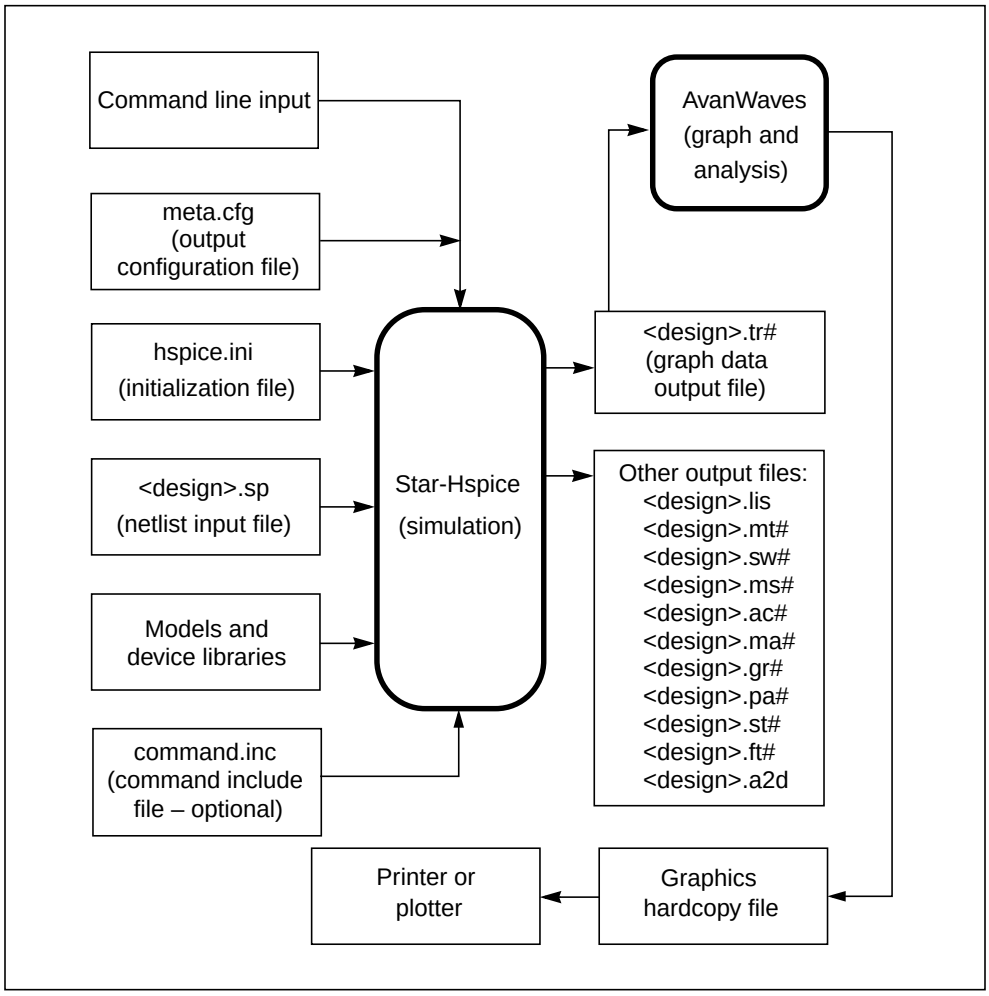
Star-Hspice accepts input and simulation control information from a number of different sources. It can output results in a number of convenient forms for review and analysis. The overall Star-Hspice data flow is shown in Figure 1-5.

To begin the design entry and simulation process, create an input netlist file. Most schematic editors and netlisters support the SPICE or Star-Hspice hierarchical format. The analyses specified in the input file are executed during the Star-Hspice run. Star-Hspice stores the simulation results requested in either an output listing file or, if .OPTIONS POST is specified, a graph data file. If POST is specified, the complete circuit solution (in either steady state, time, or frequency domain) is stored. The results for any nodal voltage or branch current can then be viewed or plotted using a high-resolution graphic output terminal or laser printer. Star-Hspice has a complete set of print and plot variables for viewing analysis results.

The Star-Hspice program has a textual command line interface. For example, the program is executed by entering the *hspice* command, the input file name, and the desired options at the prompt in a UNIX shell, on a DOS command line, or by clicking on an icon in a Windows environment. You can have the Star-Hspice program simulation output appear in either an output listing file or in a graph data file. Star-Hspice creates standard output files to describe initial conditions (*.ic* extension) and output status (*.st0* extension). In addition, Star-Hspice creates various output files in response to user-defined input options—for example, a *<design>.tr0* file in response to a .TRAN transient analysis statement.

The AvanWaves output display and analysis program has a graphical user interface. Execute AvanWaves operations using the mouse to select options and execute commands in various AvanWaves windows. Refer to the *AvanWaves User Guide* for instructions on using AvanWaves.

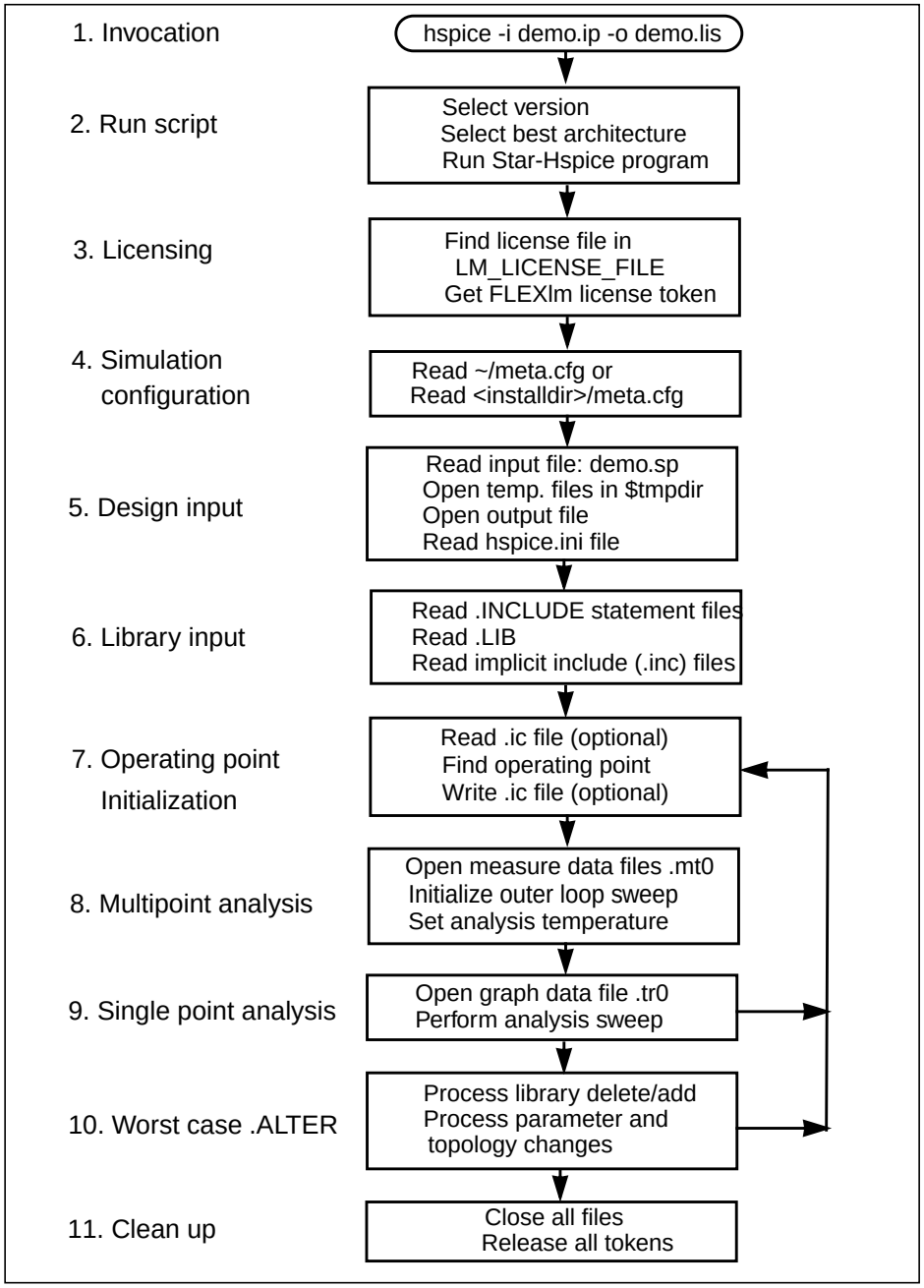
Figure 1-5: Overview of Star-Hspice Data Flow



Simulation Process Overview

Figure 1-6 is a diagram of the Star-Hspice simulation process. The following section summarizes the steps in a typical simulation.

Figure 1-6: Star-Hspice Simulation Process



Perform these steps to execute a Star-Hspice simulation.

1. Invocation

   Invoke Star-Hspice with a UNIX command such as:

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

   The Star-Hspice shell is invoked, with an input netlist file *demo.sp* and an output listing file *demo.out*. The "&" at the end of the command invokes Star-Hspice in the background so that the window and keyboard can still be used while Star-Hspice runs.

2. Script execution

   The Star-Hspice shell starts the *hspice* executable from the appropriate architecture (machine type) directory. The UNIX run script launches a Star-Hspice simulation. This procedure is used to establish the environment for the Star-Hspice executable. The script prompts for information, such as the platform you are running on and the version of Star-Hspice you want to run. (Available versions are determined when Star-Hspice is installed.)

3. Licensing

   Star-Hspice supports the FLEX*lm licensing management system*. With FLEX*lm* licensing, Star-Hspice reads the environment variable LM_LICENSE_FILE for the location of the *license.dat* file.

   If there is an authorization failure, the job terminates at this point, printing an error message in the output listing file.

4. Simulation configuration

   Star-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

   Star-Hspice opens the input netlist file. If the input netlist file does not exist, a "no input data" error appears in the output listing file.

   Three scratch files are opened in the */tmp* directory. You can change this directory by resetting the TMPDIR environment variable in the Star-Hspice command script.

Star-Hspice opens the output listing file. If you do not have ownership of the current directory, Star-Hspice terminates with a “file open” error.

An example of a simple Star-Hspice input netlist is:

```
Inverter Circuit

.OPTIONS 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

   Star-Hspice reads any files specified in .INCLUDE and .LIB statements.

7. Operating point initialization

   Star-Hspice reads any initial conditions specified in .IC and .NODESET statements, finds an operating point (that can be saved with a .SAVE statement), and writes any operating point information you requested.

8. Multipoint analysis

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

9. Single-point analysis

   Star-Hspice performs a single or double sweep of the designated quantity and produces one set of output files.

10. Worst case .ALTER

    Simulation conditions may be varied and the specified single or multipoint analysis repeated. In the above example, CLOAD is changed from 0.75 pF to 1.5 pF, and the multipoint transient analysis is repeated.

11. Normal termination

    After completing the simulation, Star-Hspice closes all files that it opened and releases all license tokens.

Avant!

Chapter 2
Getting Started

The examples in this chapter show you how to run Star-Hspice to perform some simple analyses.

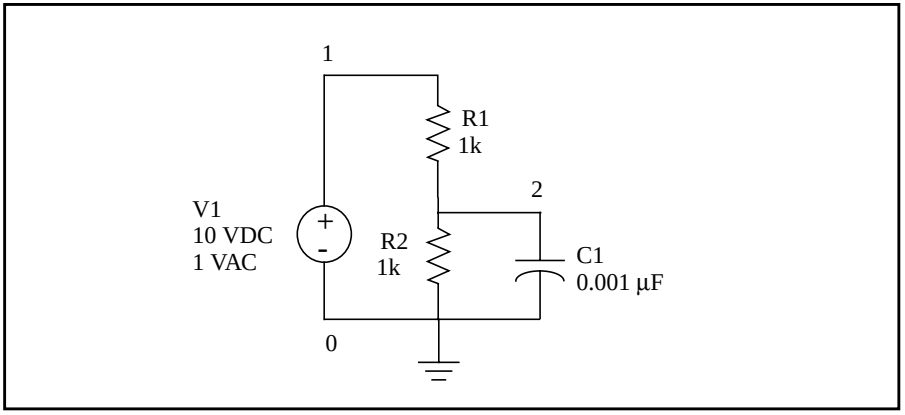
This chapter includes the following examples:

- AC Analysis of an RC Network
- Transient Analysis of an RC Network
- Transient Analysis of an Inverter

AC Analysis of an RC Network

Figure 2-1 shows a simple RC network with a DC and AC source applied. The circuit consists of two resistors, R1 and R2, capacitor C1, and the source V1. Node 1 is the connection between the source positive terminal and R1. Node 2 is where R1, R2, and C1 are connected. Star-Hspice ground is always node 0.

Figure 2-1: RC Network Circuit



The Star-Hspice netlist for the RC network circuit is:

```
A SIMPLE AC RUN
.OPTIONS LIST NODE POST
.OP
.AC DEC 10 1K 1MEG
.PRINT AC V(1) V(2) I(R2) I(C1)
V1 1 0 10 AC 1
R1 1 2 1K
R2 2 0 1K
C1 2 0 .001U
.END
```

Follow the procedure below to perform an AC analysis for the RC network circuit.

1. Type the above netlist into a file named *quickAC.sp*.
2. To run a Star-Hspice analysis, type:

```
hspice quickAC.sp > quickAC.lis
```

When the run finishes Star-Hspice displays

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

followed by a line that shows the amount of real time, user time, and system time needed for the analysis.

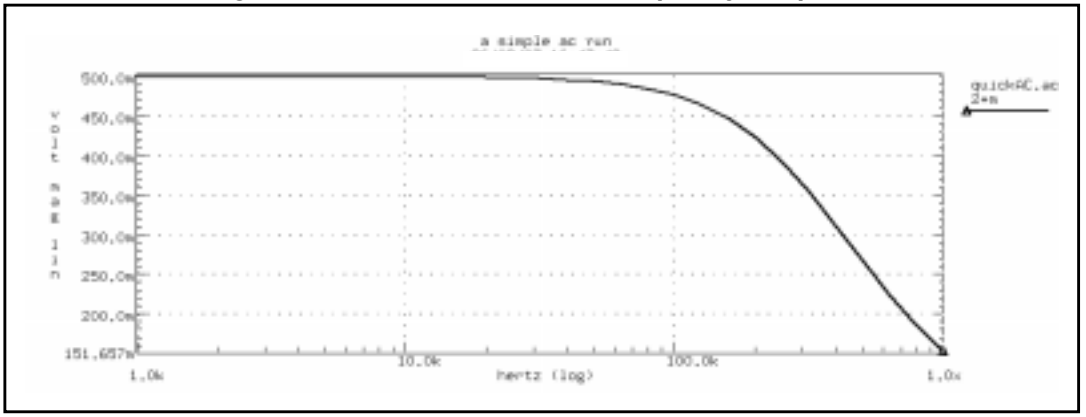
The following new files are present in your run directory:

- *quickAC.ac0*
- *quickAC.ic*
- *quickAC.lis*
- *quickAC.st0.*

3. Use an editor to view the *.lis* and *.st0* files to examine the simulation results and status.
4. Run AvanWaves and open the *.sp* file. Select the *quickAC.ac0* file from the ***Results Browser*** window to view the waveform. Display the voltage at node 2, using a log scale on the x-axis.

Figure 2-2 shows the waveform that was produced by sweeping the response of node 2 as the frequency of the input was varied from 1 kHz to 1 MHz.

Figure 2-2: RC Network Node 2 Frequency Response



The *quickAC.lis* file displays the input netlist, details about the elements and topology, operating point information, and the table of requested data as the input is swept from 1 kHz to 1 MHz. The files *quickAC.ic* and *quickAC.st0* contain information about the DC operating point conditions and the Star-Hspice run status, respectively. The operating point conditions can be used for subsequent simulation runs using the .LOAD statement.

Transient Analysis of an RC Network

As a second example, run a transient analysis using the same RC network as in Figure 2-1, but adding a pulse source to the DC and AC sources.

1. Type the following equivalent Star-Hspice netlist into a file named *quickTRAN.sp*.

```
A SIMPLE TRANSIENT RUN
.OPTIONS LIST NODE POST
.OP
.TRAN 10N 2U
.PRINT TRAN V(1) V(2) I(R2) I(C1)
V1 1 0 10 AC 1 PULSE 0 5 10N 20N 20N 500N 2U
R1 1 2 1K
R2 2 0 1K
C1 2 0 .001U
.END
```

Note: The V1 source specification has added a pulse source. The syntax for pulse sources and other types of sources is described in Using Sources and Stimuli on page 5-1.

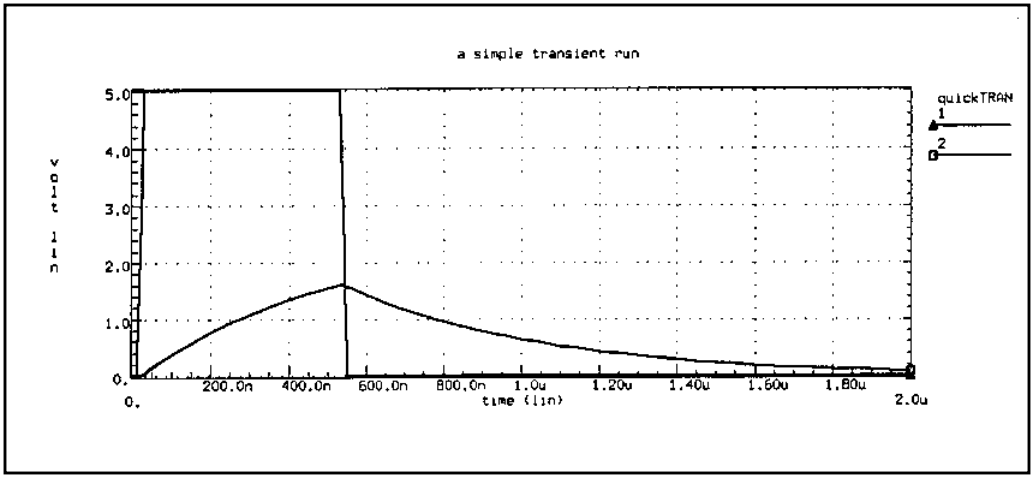
2. Type the following to run Star-Hspice.

```
hspice quickTRAN.sp > quickTRAN.lis
```

3. Use an editor to view the *.lis* and *.st0* files to examine the simulation results and status.
4. Run AvanWaves and open the *.sp* file. Select the *quickTRAN.tr0* file from the ***Results Browser*** window to view the waveform. Display the voltage at nodes 1 and 2 on the x-axis.

Figure 2-3 shows the waveforms.

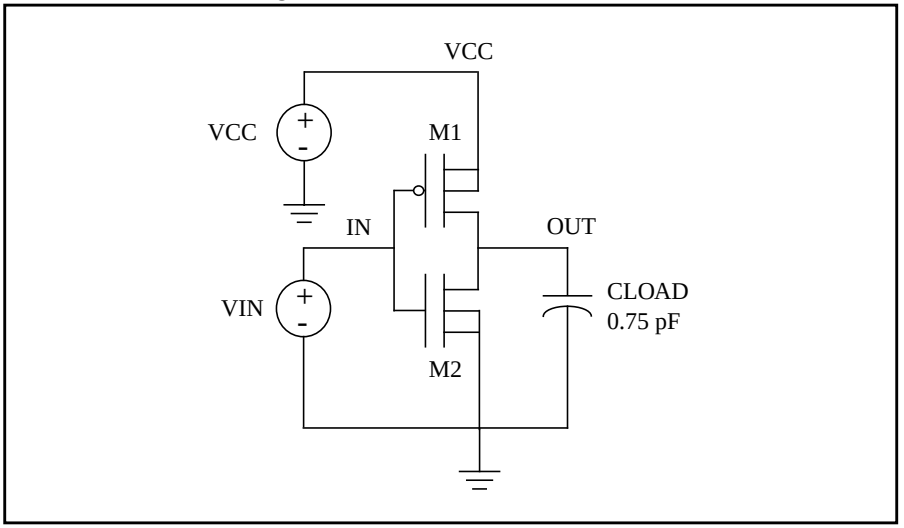
Figure 2-3: Voltages at RC Network Circuit Node 1 and Node 2



Transient Analysis of an Inverter

As a final example, analyze the behavior of the simple MOS inverter shown in Figure 2-4.

Figure 2-4: MOS Inverter Circuit



1. Type the following netlist data into a file named *quickINV.sp*.

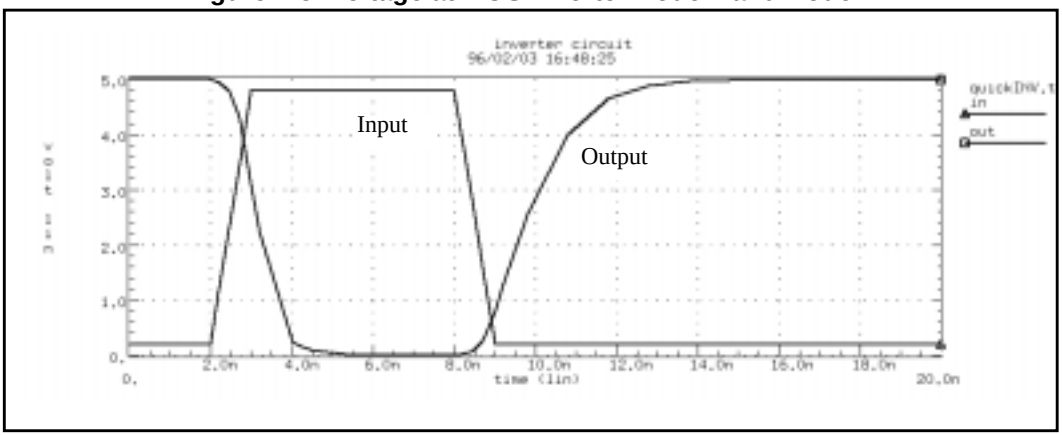
```
Inverter Circuit
.OPTIONS LIST NODE POST
.TRAN 200P 20N
.PRINT TRAN V(IN) V(OUT)
M1 OUT IN VCC VCC PCH L = 1U W = 20U
M2 OUT IN 0 0 NCH L = 1U W = 20U
VCC VCC 0 5
VIN IN 0 0 PULSE .2 4.8 2N 1N 1N 5N 20N
CLOAD OUT 0 .75P
.MODEL PCH PMOS LEVEL = 1
.MODEL NCH NMOS LEVEL = 1
.END
```

2. Type the following to run Star-Hspice.

```
hspice quickINV.sp > quickINV.lis
```

Use AvanWaves to examine the voltage waveforms at the inverter IN and OUT nodes. Figure 2-5 shows the waveforms.

Figure 2-5: Voltage at MOS Inverter Node 1 and Node 2



Avant!

Chapter 3
Simulation Input and Controls

This chapter describes the input requirements, methods of entering data, and Star-Hspice statements used to enter input. This chapter covers the following topics:

Using Netlist Input Files

This section describes how to use standard Star-Hspice netlist input files.

Input Netlist File (*<design>.sp*) Guidelines

Star-Hspice operates on an input netlist file and stores results in either an output listing file or a graph data file. The Star-Hspice input file, with the name *<design>.sp (although the filename can be anything, we recommend this form for clarity)*, contains the following:

- Design netlist (with subcircuits and macros, power supplies, and so on)
- Statement naming the library to be used (optional)
- Specification of the analysis to be run (optional)
- Specification of the output desired (optional)

Input netlist and library input files are generated by a schematic netlister or with a text editor.

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

Note: If there is no .END statement at the end of the input netlist file or no carriage return after the .END statement, an error message is issued.

Input Line Format

- The input netlist file cannot be in a packed or compressed format.
- The Star-Hspice input reader can accept an input token, such as a statement name, a node name, or a parameter name or value. A valid string of characters between two token delimiters is accepted as a token. See Delimiters on page 3-3.
- Input filename length, statement length, and equation length can be up to 1024 characters.
- Upper and lower case are ignored, except in quoted filenames.

- A statement may be continued on the next line by entering a plus (+) sign as the first nonnumeric, nonblank character in the next line.
- All statements, including quoted strings such as paths and algebraics, are continued with a backslash (\) or a double backslash (\\) at the end of the line to be continued. The single backslash preserves white space and the double backslash squeezes out any white space between the continued lines. The double backslash guarantees that path names are joined without interruption. Input lines can be 1024 characters long, so folding and continuing a line is generally only necessary to improve readability.
- Add comments at any place in the file. Lines beginning with an asterisk (*) are comments. To place a comment on the same line as input text, enter a dollar sign ($), preceded by one or more blanks, after the input text.
- An error is issued when a special control character is encountered in the input netlist file. Since most systems cannot print special control characters, the error message is ambiguous because the erroneous character cannot be shown in the error message. Use the .OPTIONS BADCHAR statement to locate such errors. The default for BADCHAR is “off”.

Names

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

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

- Names are input tokens that must be preceded and followed by token delimiters. See “Delimiters” below.
- Names can be 1024 characters long.
- Names are not case sensitive.

Delimiters

- An input token is any item in the input file that Star-Hspice recognizes. Input token delimiters are: tab, blank, comma, equal sign (=), and parentheses “( )”.
- Single or double quotes delimit expressions and filenames.
- Element attributes are delimited by colons (“M1:beta”, for example).
- Hierarchy is indicated by periods. For example, “X1.A1.V” is the V node on subcircuit A1 of circuit X1.

Nodes

- Node identifiers can be up to 1024 characters long, including periods and extensions.
- Numerical node names are valid in the range of 0 through 9999999999999999 (1-1E16).
- Leading zeros are ignored in node numbers.
- Trailing characters are ignored in node numbers. For example, node 1A is the same as node 1. Exception: Star-Hspice recognizes the following special alphabetic trailing characters (d, e, f, g, i, m, n, o, p, u, x, k, t).
- A node name can begin with any of the following characters: # _ ! %.
- Nodes are made global across all subcircuits by a .GLOBAL statement.
- Node 0, GND, GND!, and GROUND all refer to the global Star-Hspice ground. Star-Hspice treats nodes with any of these names as a *ground* node during simulation, and produces `v(0)` into the output files.

Instance Names

- The names of element instances begin with the element key letter (for example, M for a MOSFET element, D for a diode, R for a resistor, and so on), except in subcircuits.
- Subcircuit instance names begin with “X”. (Subcircuits are sometimes called macros or modules.)
- Instance names are limited to 1024 characters.
- .OPTIONS LENNAM controls the length of names in Star-Hspice printouts (default = 8).

Hierarchy Paths

- Path hierarchy is indicated by a period.
- Paths can be up to 1024 characters long.
- Path numbers compress the hierarchy for post-processing and listing files.
- Path number cross references are found in the listing and in the *<design>.pa0* file.
- .OPTIONS PATHNUM controls whether full path names or path numbers are shown in list files.

Numbers

- Numbers are entered as integer or real.
- Numbers can use exponential format or engineering key letter format, but not both (1e-12 or 1p, but not 1e-6u).
- Exponents are designated by D or E.
- Exponent size is limited by .OPTIONS EXPMAX.
- Trailing alphabetic characters are interpreted as units comments.
- Units comments are not checked.
- .OPTIONS INGOLD controls the format of numbers in printouts.
- .OPTIONS NUMDGT = x controls the listing printout accuracy.
- .OPTIONS MEASDGT = x controls the measure file printout accuracy.
- .OPTIONS VFLOOR = x specifies the smallest voltage for which the value will be printed. Smaller voltages are printed as 0.

Parameters and Expressions

- Parameter names follow Star-Hspice name syntax rules, except that names must begin with an alphabetic character. The other characters must be either a number. or one of the following characters:

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

- Parameter hierarchy overrides and defaults are defined by .OPTIONS PARHIER = global | local.
- The last parameter definition or .OPTIONS statement is used if multiple definitions exist. This is true even if the last definition or .OPTIONS statement is later in the input than a reference to the parameter or option. No warning is issued when a redefinition occurs.
- If a parameter is used in another parameter definition, the first parameter must be defined before the second parameter definition.
- In your design parameter name selection, be careful to avoid conflicts with parameterized libraries.
- Expressions are delimited by single or double quotes and are limited to 256 characters.
- A line can be continued to improve readability by using a double slash at end of the line (\\).
- Function nesting is limited to three levels.
- No user-defined function can have more than two arguments.

- Use the PAR(expression or parameter) function to evaluate expressions in output statements.

Input Netlist File Structure

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

You can use several high-level call statements to restructure the input netlist file modules. These are the .INCLUDE, .LIB and .DEL LIB statements. 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 that contains parameterized data for element sources and models.

Schematic Netlists

Star-Hspice circuits typically are generated from schematics by netlisters. Star-Hspice accepts either hierarchical or flat netlists. The normal SPICE netlisters flatten out all subcircuits and rename all nodes to numbers. Avoid flat netlisters if possible.

The process of creating a schematic involves:

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

Input Netlist File Sections and Chapter References

Sections	Examples	Ch	Definition
Title	.TITLE	3	The first line is the input netlist file title
Set-up	.OPTIONS	9	Sets conditions for simulation
	.IC or .NODESET	10	Initial values in circuit and subcircuit
	.PARAM	7	Set parameter values in the netlist
	.GLOBAL	7	Set node name globally in netlist
Sources	Sources (I or V) and digital inputs	5	Sets input stimuli
Netlist	Circuit elements	3-4	Circuit for simulation
	.SUBKCT, .ENDS	3	Subcircuit definitions
Analysis	.DC, .TRAN, .AC, etc.	10-12	Statements to perform analyses
	.SAVE and .LOAD	10	Save and load operating point info
	.DATA	3	Create table for data-driven analysis
	.TEMP	3	Set analysis temperature
Output	.PRINT, .PLOT, .GRAPH, .PROBE	8	Statements to output variables
	.MEASURE	8	Statement to evaluate and report user-defined functions of a circuit

Sections	Examples	Ch	Definition
Library, Model and File Inclusion	.INCLUDE	3	General include files
	.MODEL	3, 8	Element model descriptions
	.LIB	3	Library
	.<UN>PROTECT	3	Control printback to output listing
Alter blocks	.ALTER	3	Sequence for in-line case analysis
	.DELETE LIB	3	Removes previous library selection
End of netlist	.END	3	Required statement to end the netlist

Input Netlist File Composition

Title of Simulation and .TITLE Statement

The simulation title is set using the first line of the input file. This line is always read and used as the title of the simulation regardless of the contents of this line. The title is printed verbatim in each section heading of the output listing file of the simulation.

The .TITLE statement, as shown in the first syntax below, can be used on the first line of the netlist to set the title, although the .TITLE syntax is not necessary. In the second form shown below, the string is the first line of the input file. The first line of the input file is always the implicit title. If a Star-Hspice statement appears as the first line in a file, it is interpreted as a title and is not executed.

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

Syntax

```
.TITLE <string of up to 72 characters>
```

or

```
<string of up to 72 characters>
```

Comments

An asterisk (*) as the first nonblank character or an inline dollar sign ($) indicates a comment statement.

Syntax

```
* <comment on a line by itself>
```

or

```
<HSPICE statement> $ <comment following HSPICE input>
```

Example

```
*RF = 1K   GAIN SHOULD BE 100
$ MAY THE FORCE BE WITH MY CIRCUIT
VIN 1 0 PL 0 0 5V 5NS $ 10v 50ns
R12 1 0 1MEG $ FEED BACK
```

You can place comment statements anywhere in the circuit description.

The * must be in the first space on the line.

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

Element and Source Statements

Element statements describe the netlists of devices and sources. Elements are connected to one another by nodes, which can either be numbers or names. Element statements specify

- Type of device
- Nodes to which the device is connected
- Parameter values that describe the operating electrical characteristics of the device

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

Element statements for the various types of Star-Hspice elements are described in the chapters on those types of elements.

Syntax

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

or

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

or

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

where:

elname — Element name that cannot exceed 1023 characters, and must begin with a specific letter for each element type:

B	IBIS buffer
C	Capacitor
D	Diode
E,F,G,H	Dependent current and voltage sources
I	Current source
J	JFET or MESFET
K	Mutual inductor
L	Inductor
M	MOSFET
Q	BJT
R	Resistor
T,U,W	Transmission line
V	Voltage source
X	Subcircuit call

node1 ... — Node names are identifiers of the nodes to which the element is connected. Node names must begin with a letter that may be followed by up to 1023 additional alphanumeric characters. The following characters are not allowed in node names: = ( ),’ <space>

mname — Model reference name is required for all elements except passive devices.

pname1 ... — Element parameter name used to identify the parameter value that follows this name.

expression	Any mathematical expression containing values or parameters, i.e., ‘param1 * val2’
val1 ...	Value assigned to the parameter pname1 or to the corresponding model node. The value can be a number or an algebraic expression.
M = val	Element multiplier. Replicates the element “val” times in parallel.

Example

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

The previous 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 node SUBSTRATE. The transistor parameters are described in the model statement referenced by the name BJTMODEL.

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

The previous example specifies a MOSFET called M1, whose drain, gate, source, and substrate nodes are named ADDR, SIG1, GND, and SBS, respectively. The element statement calls an associated model statement, N1. MOSFET dimensions are specified as width = 100 microns and length = 10 microns.

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

The previous example specifies a MOSFET called M1, whose drain, gate, source, and substrate nodes are named ADDR, SIG1, GND, and SBS, respectively. The element statement calls an associated model statement, N1. MOSFET dimensions are also specified as algebraic expressions, width = w1+w and length = l1+l.

.SUBCKT or .MACRO Statement

The syntax is:

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

or

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

where:

subnam	Specifies reference name for the subcircuit model call
n1 …	Node numbers for external reference; cannot be ground node (zero). Any element nodes appearing in the subcircuit but not included in this list are strictly local, with three exceptions: 1. Ground node (zero) 2. Nodes assigned using BULK = node in the MOSFET or BJT models 3. Nodes assigned using the .GLOBAL statement
parnam	A parameter name set to a value. For use only in the subcircuit, overridden by an assignment in the subcircuit call or by a value set in a .PARAM statement.

Example

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

```
*
.MACRO SUB2 1 2 P6 = 11
     R1 1 2 P6
     R2 2 0 P2
.EOM
*
     X1 1 2 SUB1 P4 = 6
     X2 3 4 SUB1 P6 = 15
     X3 3 4 SUB2
*
.MODEL DA D CJA = CAJA CJP = CAJP VRB = -20 IS = 7.62E-18
+   PHI = .5 EXA = .5 EXP = .33
*
.PARAM CAJA = 2.535E-16 CAJP = 2.53E-16
.END
```

The above example defines two subcircuits: SUB1 and SUB2. These are resistor divider networks whose resistance values have been parameterized. They are called with the X1, X2, and X3 statements. Since the resistor value parameters are different in each call, these three calls produce different subcircuits.

.ENDS or .EOM Statement

The syntax is:

```
.ENDS <SUBNAM>
```

or

```
.EOM <SUBNAM>
```

Example

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

This statement must be the last for any subcircuit definition. The subcircuit name, if included, indicates which definition is being terminated. Subcircuit references (calls) may be nested within subcircuits.

Subcircuit Call Statement

The syntax is:

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

where:

Xyyy	Subcircuit element name. Must begin with an “X”, which may be followed by up to 15 alphanumeric characters.
n1 …	Node names for external reference
subnam	Subcircuit model reference name
parnam	A parameter name set to a value (val) for use only in the subcircuit. It overrides a parameter value assigned in the subcircuit definition, but is overridden by a value set in a .PARAM statement.
M	Multiplier. Makes the subcircuit appear as M subcircuits in parallel. This is useful in characterizing circuit loading. No additional calculation time is needed to evaluate multiple subcircuits.

Example

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

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

Example

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

The above example defines a subcircuit named YYY. The subcircuit consists of two 1 ohm resistors in series. The subcircuit node, NODEX, is initialized with the .IC statement through the passed parameter VCC.

Note: A warning message is generated if a nonexistent subcircuit node is initialized. This can occur if an existing *.ic* file (initial conditions) is used to initialize a circuit modified since the *.ic* file was created.

Element and Node Naming Conventions

Node Names

Nodes are the points of connection between elements in the input netlist file. In Star-Hspice, nodes are designated by either names or by numbers. Node numbers can be from 1 to 999999999999999; node number 0 is always ground. Letters that follow numbers in node names are ignored. Node names must begin with a letter and are followed by up to 1023 characters.

In addition to letters and digits, the following characters are allowed in node names:

+	plus sign
-	minus sign or hyphen
*	asterisk
/	slash
$	dollar sign
#	pound sign

[]	left and right square brackets
!	exclamation mark
<>	left and right angle brackets
_	underscore
%	percent sign

Braces, “{ }”, are allowed in node names, but Star-Hspice changes them to square brackets, “[ ]”.

The following characters are not allowed in node names:

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

The period is reserved for use as a separator between the subcircuit name and the node name:

<subcircuitName>.<nodeName>.

The sorting order for operating point nodes is

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

Instance and Element Names

Star-Hspice elements have names that begin with a letter designating the element type, followed by up to 1023 alphanumeric characters. Element type letters are R for resistor, C for capacitor, M for a MOSFET device, and so on (see Element and Source Statements on page 3-10).

Subcircuit Node Names

Subcircuit node names are assigned two different names in Star-Hspice. The first name is assigned by concatenating the circuit path name with the node name through the (.) extension – for example, X1.XBIAS.M5.

Note: Node designations starting with the same number followed by any letter are all the same. For example, *1c* and *1d* represent the same node.

The second subcircuit node name is a unique number that Star-Hspice assigns automatically to an input netlist file subcircuit. This number is concatenated using the (:) extension with the internal node name, giving the entire subcircuit's node name (for example, 10:M5). The node name is cross referenced in the output listing file Star-Hspice produces.

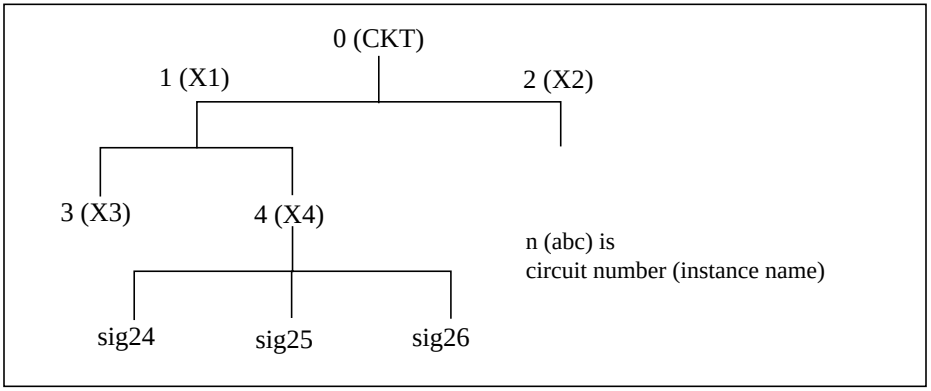
The ground node must be indicated by either the number 0, the name GND, or !GND. Every node should have at least two connections, except for transmission line nodes (unterminated transmission lines are permitted) and MOSFET substrate nodes (which have two internal connections). Floating power supply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

Path Names of Subcircuit Nodes

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

You can use path names in the .PRINT, .PLOT, .NODESET, and .IC statements as an alternative method to reference internal nodes (nodes not appearing on the parameter list). Any node, including any internal node, can be referenced by its path name. Subcircuit node and element names follow the rules illustrated in Figure 3-1.

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



The path name of the node named *sig25* in subcircuit X4 in Figure 3-1 is *X1.X4.sig25*. You can use this path in Star-Hspice statements such as:

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

Abbreviated Subcircuit Node Names

You can use circuit numbers as an alternative to path names to reference nodes or elements in .PRINT, .PLOT, .NODESET, or .IC statements. Star-Hspice assigns a circuit number to all subcircuits on compilation, creating an abbreviated path name:

```
<subckt-num>:<name>
```

Every occurrence of a node or element in the output listing file is prefixed with the subcircuit number and colon. For example, 4:INTNODE1 denotes a node named INTNODE1 in a subcircuit assigned the number 4.

Any node not in a subcircuit is prefixed by 0: (0 references the main circuit). All nodes and subcircuits are identified in the output listing file with a circuit number referencing the subcircuit where the node or element appears. Abbreviated path names allow use of DC operating point node voltage output as input in a .NODESET for a later run; the part of the output listing titled “Operating Point Information” can be copied or typed 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

Star-Hspice has an automatic system for assigning internal node names. You can check both nodal voltages and branch currents by printing or plotting with the assigned node name. Several special cases for node assignment occur in Star-Hspice. Node 0 is automatically assigned as a ground node.

For CSOS (CMOS Silicon on Sapphire), if the bulk node is assigned the value -1, the name of the bulk node is B#. Use this name for printing the voltage at the bulk node. When printing or plotting current, for example .PLOT I(R1), Star-Hspice inserts a zero-valued voltage source. This source inserts an extra node in the circuit named V*nn*, where *nn* is a number automatically generated by Star-Hspice and appears in the output listing file.

.GLOBAL Statement

The .GLOBAL statement globally assigns a node name. This means that all references to a global node name used at any level of the hierarchy in the circuit, will be connected to the same node.

The .GLOBAL statement is most often used when subcircuits are included in a netlist file. This statement assigns a common node name to subcircuit nodes. Power supply connections of all subcircuits are often assigned using a .GLOBAL statement. For example, .GLOBAL VCC connects all subcircuits with the internal node name VCC. Ordinarily, in a subcircuit the node name is given as the circuit number concatenated to the node name. When a .GLOBAL statement is used, the node name is not concatenated with the circuit number and is only assigned the global name. This allows exclusion of the power node name in the subcircuit or macro call.

Syntax

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

where:

node1 ...	Specifies global nodes, such as supply and clock names; overrides local subcircuit definitions.

Example

This example shows global definitions for the nodes VDD and input_sig.

```
.GLOBAL VDD input_sig
```

.TEMP Statement

The temperature of a circuit for a Star-Hspice run is specified with the .TEMP statement or with the TEMP parameter in the .DC, .AC, and .TRAN statements. The circuit simulation temperature set by either of these is compared against the reference temperature set by the TNOM control option. The difference between the circuit simulation temperature and the reference temperature, TNOM, is used in determining the derating factors for component values. Temperature analysis is discussed in “Temperature Analysis” on page Chapter 135.

Syntax

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

where:

t1 t2 …	Specifies temperatures, in °C, at which the circuit is to be simulated

Example

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

The .TEMP statement sets the circuit temperatures for the entire circuit simulation. Star-Hspice uses the temperature set in the .TEMP statement, along with the TNOM option setting (or the TREF model parameter) and the DTEMP element temperature, and simulates the circuit with individual elements or model temperatures.

Example

```
.TEMP 100
  D1 N1 N2 DMOD DTEMP = 30
  D2 NA NC DMOD
  R1 NP NN 100 TC1 = 1 DTEMP = -30
.MODEL DMOD D IS = 1E-15 VJ = 0.6 CJA = 1.2E-13
+ CJP = 1.3E-14 TREF = 60.0
```

The circuit simulation temperature is given from the .TEMP statement as 100°C. Since TNOM is not specified, it will default to 25°C. The temperature of the diode is given as 30°C above the circuit temperature by the DTEMP parameter; that is, D1temp = 100°C + 30°C = 130°C. The diode D2 is simulated at 100°C. R1 is simulated at 70°C. Since TREF is specified at 60°C in the diode model statement, the diode model parameters given are derated by 70°C (130°C - 60°C) for diode D1 and by 40°C (100°C - 60°C) for diode D2. The value of R1 is derated by 45°C (70°C - TNOM).

.DATA Statement

Data-driven analysis allows the user to modify any number of parameters, then perform an operating point, DC, AC, or transient analysis using the new parameter values. 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.

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

The .DATA statement provides a means for using concatenated or column laminated data sets for optimization on measured I-V, C-V, transient, or s-parameter data.

You can also use the .DATA statement for a first or second sweep variable in cell characterization and worst case corners testing. Data measured in a lab, such as transistor I-V data, is read one transistor at a time in an outer analysis loop. Within the outer loop, the data for each transistor (IDS curve, GDS curve, and so on) is read one curve at a time in an inner analysis loop.

The .DATA statement specifies the parameters for which values are to be changed and gives the sets of values that are to be assigned during each simulation. The required simulations are done as an internal loop. This bypasses reading in the netlist and setting up the simulation, and saves computing time. Internal loop simulation also allows simulation results to be plotted against each other and printed in a single output.

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

.DATA statements are referred to by their datanames, so each dataname must be unique. Star-Hspice supports three .DATA statement formats:

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

These formats are described below. The keywords MER and LAM tell Star-Hspice to expect external file data rather than inline data. The keyword FILE denotes the external filename. Simple file names like *out.dat* can be used without the single or double quotes (‘ ’ or “ ”), but using them prevents problems with file names that start with numbers like *1234.dat*. Remember that file names are case sensitive on UNIX systems.

For more details about using the .DATA statement in the different types of analysis, see Chapter 9, “Specifying Simulation Options”, Chapter 10, “Initializing DC/Operating Point Analysis”, and Chapter 11, “Performing Transient Analysis”. In short, the syntax is:

Operating point:

```
.DC DATA = dataname
```

DC sweep:

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

AC sweep:

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

TRAN sweep:

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

For any data driven analysis, make sure that the start time (time 0) is specified in the analysis statement, to ensure that the stop time is calculated correctly.

Inline .DATA Statement

Inline data is parameter data listed in a .DATA statement block. It is called by the *datanm* parameter in a .DC, .AC, or .TRAN analysis statement.

Syntax

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

where:

datanm	Specifies the data name referred to in the .TRAN, .DC or .AC statement
pnami	Specifies the parameter names used for source value, element value, device size, model parameter value, and so on. These names must be declared in a .PARAM statement.
pvali	Specifies the parameter value

The number of parameters read in determines the number of columns of data. The physical number of data numbers per line does not need to correspond to the number of parameters. In other words, if 20 parameters are needed, it is not necessary to put 20 numbers per line.

Example

```
.TRAN         1n    100n          SWEEP DATA = devinf
  .AC DEC     10    1hz   10khz   SWEEP DATA = devinf
  .DC TEMP    -55   125   10      SWEEP DATA = devinf

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

Star-Hspice performs the above analyses for each set of parameter values defined in the .DATA statement. For example, the program first takes the width = 50u, length = 30u, thresh = 1.2v, and cap = 1.2pf parameters and performs .TRAN, .AC and .DC analyses. The analyses are then repeated for width = 25u, length = 15u, thresh = 1.0v, and cap = 0.8pf, and again for the values on each subsequent line in the .DATA block.

This is an example of .DATA as the inner sweep:

```
M1 1 2 3 0 N W = 50u L = LN
   VGS 2 0 0.0v
   VBS 3 0 VBS
   VDS 1 0 VDS
   .PARAM VDS = 0 VBS = 0 L = 1.0u
   .DC DATA = vdot
   .DATA vdot
     VBS      VDS        L
      0       0.1        1.5u
      0       0.1        1.0u
      0       0.1        0.8u
     -1       0.1        1.0u
     -2       0.1        1.0u
     -3       0.1        1.0u
      0       1.0        1.0u
      0       5.0        1.0u
.ENDDATA
```

In the above example, a DC sweep analysis is performed for each set of VBS, VDS, and L parameters in the ".DATA vdot" block. That is, eight DC analyses are performed, one for each line of parameter values in the .DATA block.

This is an example of .DATA as an outer sweep:

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

In the previous example, the default start time for the .TRAN analysis is 0, the time increment is 1 ns, and the stop time is 100 ns. This results in transient analyses at every time value from 0 to 100 ns in steps of 1 ns, using the first set of parameter values in the ".DATA d1" block. Then the next set of parameter values is read, and another 100 transient analyses are performed, sweeping time from 0 to 100 ns in 1 ns steps. The outer sweep is time, and the inner sweep varies the parameter values. Two hundred analyses are performed: 100 time increments times 2 sets of parameter values.

External File .DATA Statement

Syntax

This is the syntax for concatenated data files.

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

where:

datanm	Specifies the data name referred to in the .TRAN, .DC or .AC statement
MER	Specifies concatenated (series merging) data files to be used
filenamei	Specifies the name of the data file to be read. Files are concatenated in the order they appear in the .DATA statement. A maximum of 10 files can be specified.
pnami	Specifies the parameter names used for source value, element value, device size, model parameter value, and so on. These names must be declared in a .PARAM statement.

colnum Specifies the column number in the data file for the parameter value. The column need not be the same between files.

fileouti Specifies the name of the data file to be written with all the data concatenated. This file will have the complete syntax for an inline .DATA statement, and can replace the .DATA statement that created it in the netlist. Outputting the file is optional, and can be used to generate one data file from many.

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

```
File A     File B     File C

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

the data appears as follows:

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

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

Example

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

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

For data files with fewer columns than others, the missing parameters will be given values of zero.

Column Laminated .DATA Statement

Syntax

This is the syntax for column laminated data files.

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

where:

datanm	Specifies the data name referred to in the .TRAN, .DC or .AC statement
LAM	Specifies column laminated (parallel merging) data files to be used
filenamei	Specifies the name of the data file to be read. Files are concatenated in the order they appear in the .DATA statement. A maximum of 10 files can be specified.

pnami	Specifies the parameter names used for source value, element value, device size, model parameter value, and so on. These names must be declared in a .PARAM statement.
colnum	Specifies the column number in the data file for the parameter value. The column need not be the same between files.
fileouti	Specifies the name of the data file to be written with all the data concatenated. This file will have the complete syntax for an inline .DATA statement, and can replace the .DATA statement that created it in the netlist. Outputting the file is optional, and can be used to generate one data file from many.

Column lamination means that the columns of files with the same number of rows are arranged side-by-side. For example, for three files *D*, *E*, and *F* containing the following columns of data,

```
File D        File E      File F

d1 d2 d3      e4 e5       f6
d1 d2 d3      e4 e5       f6
d1 d2 d3      e4 e5       f6
```

the laminated data appears as follows:

```
d1 d2 d3      e4 e5       f6
d1 d2 d3      e4 e5       f6
d1 d2 d3      e4 e5       f6
```

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

Note: The number of lines (rows) of data in each file need not be the same. Data points missing will be interpreted as zero.

Example

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

This listing takes columns from *file1*, and *file2*, and laminates them into the output file, *fileout*. Columns one, two, and three of *file1*, columns one and two of *file2* are designated as the columns to be placed in the output file. There is a limit of 10 files per .DATA statement.

Note: When Star-Hspice is run on a different machine than the one on which the input data files reside (such as when working over a network), use full path names instead of aliases whenever possible, since aliases may have different definitions on different machines.

.INCLUDE Statement

Syntax

```
.INCLUDE '<filepath> filename'
```

where:

filepath	Path name of a file for computer operating systems supporting tree structured directories.
filename	Name of a file to include in the data file. The file path plus file name can be up to 1024 characters in length and can be any valid file name for the computer's operating system. The file path and name *must* be enclosed in single or double quotation marks.

.MODEL Statement

Syntax

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

where:

mname Model name reference. Elements must refer to the model by this name.
Note: Model names that contain periods (.) can cause the Star-Hspice automatic model selector to fail under certain circumstances.

type Selects the model type, which must be one of the following:
AMPoperational amplifier model
Ccapacitor model
COREmagnetic core model
Ddiode model
Lmagnetic core mutual inductor model
NJFn-channel JFET model
NMOSn-channel MOSFET model
NPNnpn BJT model
OPToptimization model
PJFp-channel JFET model
PLOTplot model for the .GRAPH statement
PMOSp-channel MOSFET model
PNPpnp BJT model
Rresistor model
Ulossy transmission line model (lumped)
Wlossy transmission line model
SPS parameter

pname1 ... Parameter name. The model parameter name assignment list (pname1) must be from the list of parameter names for the appropriate model type. Default values are given in each model section. The parameter assignment list can be enclosed in parentheses and each assignment can be separated by either blanks or commas for legibility. Continuation lines begin with a plus sign (+).

VERSION Star-Hspice version number, used to allow portability of the BSIM (LEVEL=13), BSIM2 (LEVEL = 39) models between Star-Hspice releases. Star-Hspice release numbers and the corresponding version numbers are:

Star-Hspice release	*Version number*
9007B	9007.02
9007D	9007.04
92A	92.01
92B	92.02
93A	93.01
93A.02	93.02
95.3	95.3
96.1	96.1

The VERSION parameter is only valid for LEVEL 13 and LEVEL 39 models, and in Star-Hspice releases starting with Release H93A.02. Using the parameter with any other model or with a release prior to H93A.02 results in a warning message, but the simulation continues. **Note:** VERSION is also used to denote the BSIM3v3 version number only in model LEVELs 49 and 53. For LEVELs 49 and 53, the parameter HSPVER is used to denote the Star-Hspice release number.

Example

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

.LIB Call and Definition Statements

You can place commonly used commands, device models, subcircuit analysis and statements in library files by using the .LIB call statement. As each .LIB call name is encountered in the main data file, the corresponding entry is read in from the designated library file. The entry is read in until an .ENDL statement is encountered.

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

.LIB Library Call Statement

Syntax

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

where:

filepath Path to a file. Used where a computer supports tree-structured directories. When the LIB file (or alias) resides in the same directory in which Star-Hspice is run, no directory path need be specified; the netlist runs on any machine. You can use the “../” syntax in the filepath to designate the parent directory of the current directory.

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

entryname Entry name for the section of the library file to include. The first character of an entryname cannot be an integer.

Example

```
.LIB 'MODELS' cmos1
```

.LIB Library File Definition Statement

You can build libraries by using the .LIB statement in a library file. For each macro in a library, a library definition statement (.LIB entryname) and an .ENDL statement is used. The .LIB statement begins the library macro, and the .ENDL statement ends the library macro.

Syntax

```
.LIB entryname1
.
. $ ANY VALID SET OF Star-Hspice STATEMENTS
.
.ENDL entryname1
.LIB entryname2
.
. $ ANY VALID SET OF Star-Hspice STATEMENTS
.
.ENDL entryname2
.LIB entryname3
.
. $ ANY VALID SET OF Star-Hspice STATEMENTS
.
.ENDL entryname3
```

The text following a library file entry name must consist of valid Star-Hspice statements.

.LIB Nested Library Calls

Library calls may call other libraries, provided they are different files.

Example

Shown below are an illegal example and a legal example for a library assigned to library “file3.”

Illegal:

```
.LIB MOS7
...
```

```
.LIB 'file3' MOS7    $ This call is illegal within
library MOS7
...
...
.ENDL
```

Legal:

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

Library calls are nested to any depth. This capability, along with the .ALTER statement, allows the construction of a sequence of model runs composed of similar components with different model parameters, without duplicating the entire Star-Hspice input file.

Library Building Rules

1. A library cannot contain .ALTER statements.
2. A library may contain nested .LIB calls to itself or other libraries. The depth of nested calls is only limited by the constraints of your system configuration.
3. A library *cannot* contain a call to a library of its own entry name within the same library file.
4. A library cannot contain the .END statement.
5. .LIB statements within a file called with an .INCLUDE statement cannot be changed by .ALTER processing.

The simulator accesses the models and skew parameters through the .LIB statement and the .INCLUDE statement. The library contains parameters that

modify .MODEL statements. The following example of a .LIB of model skew parameters features both worst case and statistical distribution data. The statistical distribution median value is the default for all non-Monte Carlo analysis.

Example

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


    .LIB FF
    $HIGH GAIN P-CH AND N-CH CMOS LIBRARY 3SIGMA VALUES
.PARAM TOX = 220 XL = -0.03 DELVTON = -.2V DELVTOP = -0.15V
    .INC '/usr/meta/lib/cmos1_mod.dat'$ model include file
    .ENDL FF
```

The model would be contained in the include file */usr/meta/lib/cmos1_mod.dat*.

```
    .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 keyword (left-hand side) is being equated to the skew parameter (right-hand side). A model keyword can be the same as a skew parameter.

.OPTIONS SEARCH Statement

This statement allows a library to be accessed automatically.

Syntax

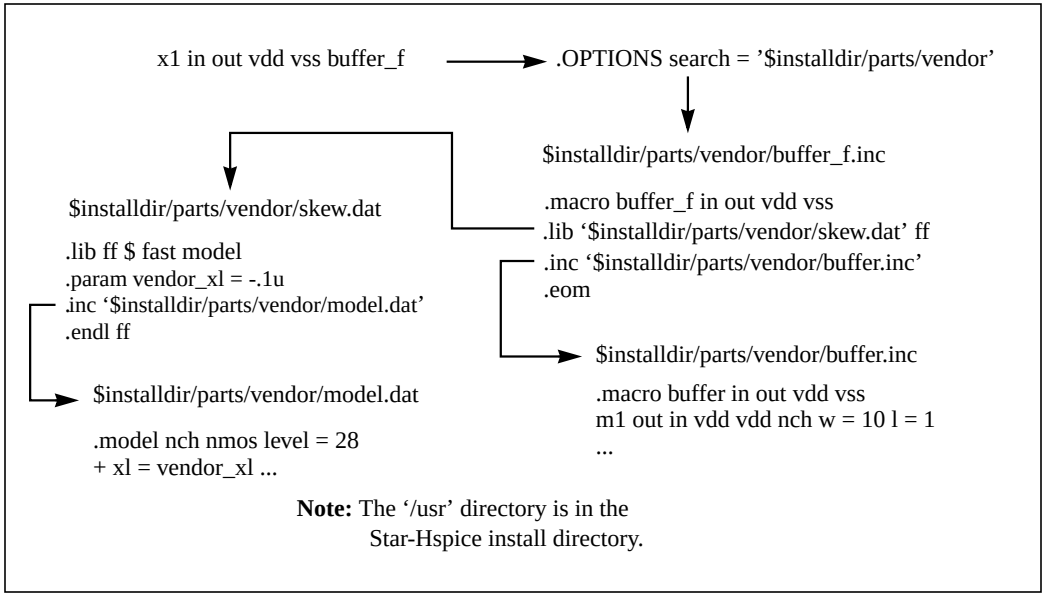
```
.OPTIONS SEARCH = 'directory_path'
```

Example

```
.OPTIONS SEARCH = '$installdir/parts/vendor'
```

The above example sets the search path to find models by way of a *vendor* subdirectory under the installation directory, *$installdir/parts* (see Figure 3-2). The DDL subdirectories are contained in the *parts*/ directory.

Figure 3-2: Vendor Library Usage



Automatic Library Selection

Automatic library selection allows a search order for up to 40 directories. The *hspice.ini* file sets the default search paths. Use this file for any directories that should always be searched. Star-Hspice searches for libraries in the order in

which libraries are specified in .OPTIONS SEARCH statements. When Star Hspice encounters a subcircuit call, the search order is as follows:

1. Read the input file for a .SUBCKT or .MACRO with call name.
2. Read any .INC files or .LIB files for a .SUBCKT or .MACRO with the call name.
3. Search the directory that the input file was in for a file named *call_name.inc*.
4. Search the directories in the .OPTIONS SEARCH list.

By using the Star-Hspice library search and selection features you can, for example, simulate process corner cases, using .OPTIONS SEARCH = '<*libdir*>' to target different process directories. If you have an input/output buffer subcircuit stored in a file named *iobuf.inc*, create three copies of the file to simulate *fast*, *slow* and *typical* corner cases. Each file contains different Star-Hspice transistor models representing the different process corners. Store these files (all named *iobuf.inc*) in separate directories.

.PARAM Statement

Use the .PARAM statement to define parameters. Parameters in Star-Hspice are names that have associated numeric values. You can use any of the following methods to define parameters:

- Simple Parameter
- Algebraic Parameter
- .User-Defined Function
- Subcircuit Default Definition
- Predefined Analysis
- Measurement Parameters

Simple Parameter

A simple parameter assignment is a constant real number. The parameter keeps this value unless there is a later definition that changes its value, or it is assigned a new value by an algebraic expression during simulation. There is no warning if a parameter is reassigned.

Syntax

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

Algebraic Parameter

Algebraic parameter assignments can be made by an algebraic expression of real values, predefined function, user-defined function, or circuit or model values. A complex expression must be enclosed in single quotes to invoke the Star-Hspice algebraic processor ***unless*** the expression begins with an alphabetic character and contains no blank spaces. A simple expression consists of a single parameter name.

Syntax

```
.PARAM <ParamName>='<Expression>'
```

or

```
.PARAM <ParamName1>=<ParamName2>
```

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

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

Example

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

.User-Defined Function

A user-defined function assignment is similar to the definition of an algebraic parameter definition. Star-Hspice extends the algebraic parameter definition to include function parameters that are used in the algebraic that defines the function.

User-defined functions cannot be nested more than three deep.

Syntax

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

Subcircuit Default Definition

The specification of hierarchical subcircuits allows you to pick default values for circuit elements. This is typically used in cell definitions so that the circuit can be simulated with typical values.

Syntax

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

where *SubDefaultsList* is:

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

Predefined Analysis

Star-hspice has specialized analysis types, primarily Optimization and Monte Carlo, that require a method of controlling the analysis. The syntax for these uses of .PARAM are described in “.PARAM Distribution Function Syntax” on page Chapter 1316.

Measurement Parameters

.MEASURE statements in Star-Hspice produce a type of parameter called a measurement parameter. In general, the rules for measurement parameters are the same as those for standard parameters with the exception of the definition: measurement parameters are not defined in a .PARAM statement but directly in the .MEASURE statement. For more information, see “Measure Parameter Types” on page Chapter 839.

.PROTECT Statement

Use the .PROTECT statement to keep models and cell libraries private. The .PROTECT statement suppresses the printout of the text from the list file, like the option BRIEF. The .UNPROTECT command restores normal output functions. In addition, any elements and models located between a .PROTECT and an .UNPROTECT statement inhibit the element and model listing from the option LIST. Any nodes that are contained within the .PROTECT and .UNPROTECT statements are not listed in the .OPTIONS NODE nodal cross reference, and are not listed in the .OP operating point printout.

Syntax

```
.PROTECT
```

.UNPROTECT Statement

The .UNPROTECT statement restores normal output functions from a .PROTECT statement. Any elements and models located between .PROTECT and .UNPROTECT statements inhibit the element and model listing from the option LIST. Any nodes contained within the .PROTECT and .UNPROTECT statements are not listed in either the .OPTIONS NODE nodal cross reference, or in the .OP operating point printout.

Syntax

```
.UNPROTECT
```

.ALTER Statement

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

Print and plot statements must be parameterized to be altered. The .ALTER block *cannot* include .PRINT, .PLOT, .GRAPH or any other input/output statements. You can include all analysis statements (.DC, .AC, .TRAN, .FOUR, .DISTO, .PZ, and so on) in only one .ALTER block in an input netlist file, but *only if* the analysis statement type has not been used previously in the main program.

The .ALTER sequence or block can contain:

- Element statements (except source elements)
- .DATA statements
- .DEL LIB statements
- .INCLUDE statements
- .IC (initial condition) and .NODESET statements
- .LIB statements
- .MODEL statements
- .OP statements
- .OPTIONS statements

- .PARAM statements
- .TEMP statements
- .TF statements
- .TRAN, .DC, and .AC statements
- .ALIAS statements

Altering Design Variables and Subcircuits

The following rules are used when altering design variables and subcircuits.

1. If the name of a new element, .MODEL statement, or subcircuit definition is identical to the name of an original statement of the same type, the new statement replaces the old. New statements are added to the input netlist file.
2. Element and .MODEL statements within a subcircuit definition can be changed and new element or .MODEL statements can be added to a subcircuit definition. Topology modifications to subcircuit definitions should be put into libraries and added with .LIB and deleted with .DEL LIB.
3. If a parameter name of a new .PARAM statement in the .ALTER module is identical to a previous parameter name, the new assigned value replaces the old.
4. If elements or model parameter values were parameterized when using .ALTER, these parameter values must be changed through the .PARAM statement. Do not redescribe the elements or model parameters with numerical values.
5. Options turned on by an .OPTION statement in an original input file or a .ALTER block can be turned off.
6. Only the actual altered input is printed for each .ALTER run. A special .ALTER title identifies the run.
7. .LIB statements within a file called with an .INCLUDE statement cannot be revised by .ALTER processing, but .INCLUDE statements within a file called with a .LIB statement can be accepted by .ALTER processing.

Using Multiple .ALTER Statements

For the first run, Star-Hspice reads the input file only up to the first .ALTER statement and performs the analyses up to that .ALTER statement. After the first simulation is completed, Star-Hspice reads the input between the first .ALTER and the next .ALTER or .END statement. These statements are then used to modify the input netlist file. Star-Hspice then resimulates the circuit.

For each additional .ALTER statement, Star-Hspice performs the simulation preceding the first .ALTER statement, then performs another simulation using the input between the current .ALTER statement and the next .ALTER statement or the .END statement. If you do not want to rerun the simulation preceding the first .ALTER statement every time, put the statements preceding the first .ALTER statement in a library and use the .LIB statement in the main input file, and put a .DEL LIB statement in the .ALTER section to delete that library.

Syntax

```
.ALTER <title_string>
```

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

.ALIAS Statement

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

For example, if you use `.del lib` in the `.alter` block to delete a library, the `.alter` command deletes all models in this library. If your netlist references one or more models in the deleted library, then Star-Hspice no longer finds the models.

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

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

```
.alias pa1 par1
```

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

You must specify both an old model name, and a new model name to use in its place. You cannot use the .alias command without *any* model names:

```
.alias
```

or with only *one* model name:

```
.alias pa1
```

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

```
.alias pa1 par1 par2
```

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

```
.alias pa1 par1
.alias par1 par2
```

If your netlist does not contain a .alter command, and if the .alias does not report a usage error, then the .alias does not affect the simulation results. For example, your netlist might contain the statement

```
.alias myfet nfet
```

Without a .alter statement, Star-Hspice does not use *nfet* to replace *myfet* during simulation.

If your netlist contains one or more .alter commands, the first simulation uses the original *myfet* model. After the first simulation, when the netlist references *myfet*, .alias substitutes *nfet*.

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

.DEL LIB Statement

The .DEL LIB statement is used with the .ALTER statement to remove library data from memory. The .DEL LIB statement causes the .LIB call statement with the same library number and entry name to be removed from memory the next time the simulation is run. A .LIB statement can then be used to replace the deleted library.

Syntax

```
.DEL LIB '<filepath>filename' entryname
.DEL LIB libnumber entryname
```

where:

entryname	Entry name used in the library call statement to be deleted.
filename	Name of a file for deletion from the data file. The file path plus file name can be up to 64 characters in length and can be any file name that is valid for the operating system being used. The file path and name must be enclosed in single or double quote marks.
filepath	Path name of a file, if the operating system supports tree-structured directories.
libnumber	Library number used in the library call statement to be deleted.

Example

```
FILE1: ALTER1 TEST  CMOS INVERTER
```

```
    .OPTIONS ACCT LIST
    .TEMP 125
    .PARAM WVAL = 15U VDD = 5
    *
    .OP
    .DC VIN 0  5  0.1
    .PLOT DC V(3) V(2)
    *
    VDD 1 0 VDD
    VIN 2 0
    *
    M1  3  2  1  1  P 6U 15U
    M2  3  2  0  0  N 6U W = WVAL
    *
.LIB 'MOS.LIB' NORMAL
.ALTER
    .DEL LIB 'MOS.LIB' NORMAL$removes LIB from memory
    $PROTECTION
    .PROT $protect statements below .PROT
    .LIB 'MOS.LIB' FAST $get fast model library
.UNPROT
    .ALTER
    .OPTIONS NOMOD OPTS $suppress model parameters printing
    *  and print the option summary
    .TEMP -50  0  50 $run with different temperatures
    .PARAM WVAL = 100U VDD = 5.5 $change the parameters
    VDD 1 0 5.5 $using VDD 1 0 5.5 to change the
     $power supply VDD value doesn't
    $work
```

```
VIN 2 0 PWL 0NS 0 2NS 5 4NS 0 5NS 5
 $change the input source
.OP VOL $node voltage table of operating
 $points
.TRAN 1NS 5NS $run with transient also
M2 3 2 0 0 N 6U WVAL $change channel width
.MEAS SW2 TRIG V(3) VAL = 2.5 RISE = 1 TARG V(3)
+ VAL = VDD CROSS = 2  $measure output
*
.END
```

Example 1 calculates a DC transfer function for a CMOS inverter. The device is first simulated using the inverter model NORMAL from the MOS.LIB library. By using the .ALTER block and the .LIB command, a faster CMOS inverter, FAST, is substituted for NORMAL and the circuit is resimulated. With the second .ALTER block, DC transfer analysis simulations are executed at three different temperatures and with an n-channel width of 100 μm instead of 15 μm. A transient analysis also is conducted in the second .ALTER block, so that the rise time of the inverter can be measured (using the .MEASURE statement).

This is a second example:

```
FILE2: ALTER2.SP CMOS INVERTER USING SUBCIRCUIT
.OPTIONS LIST ACCT
.MACRO INV 1  2  3
M1  3  2  1  1  P  6U 15U
M2  3  2  0  0  N  6U  8U
.LIB 'MOS.LIB' NORMAL
.EOM INV
XINV  1  2 3  INV
VDD  1  0  5
VIN  2  0
.DC VIN  0  5 0. 1
.PLOT V(3)  V(2)
.ALTER
.DEL LIB 'MOS.LIB' NORMAL
.TF V(3) VIN              $DC small-signal transfer function
*
.MACRO INV 1  2  3        $change data within subcircuit def
M1  4 2 1 1 P 100U 100U   $change channel length,width,also
                          $topology
```

```
M2  4 2 0 0 N 6U   8U      $change topology
R4  4  3  100              $add the new element
C3  3  0  10P              $add the new element
.LIB 'MOS.LIB' SLOW        $set slow model library
$.INC 'MOS2.DAT'           $not allowed to be used inside
                           $subcircuit allowed outside
                           $subcircuit
.EOM INV
*
.END
```

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

.END Statement

The Star-Hspice input netlist file must have an .END statement as the last statement. The period preceding END is a required part of the statement.

Any text that follows the .END statement is treated as a comment and has no effect on that simulation. A Star-Hspice input file that contains more than one Star-Hspice run must have an .END statement for each Star-Hspice run. Any number of simulations may be concatenated into a single file.

Syntax

```
.END <comment>
```

Example

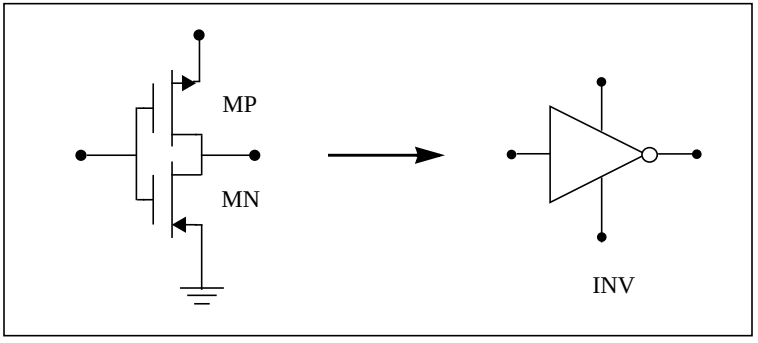
```
MOS OUTPUT
    .OPTIONS NODE NOPAGE
    VDS 3 0
    VGS 2 0
    M1 1 2 0 0 MOD1 L = 4U W = 6U AD = 10P AS = 10P
    .MODEL MOD1 NMOS VTO = -2 NSUB = 1.0E15 TOX = 1000
    UO = 550
    VIDS 3 1
    .DC   VDS 0 10 0.5    VGS 0 5 1
```

```
    .PRINT DC I(M1) V(2)
    .END MOS OUTPUT
    MOS CAPS
    .OPTIONS SCALE = 1U  SCALM = 1U WL ACCT
    .OP
    .TRAN .1 6
    V1 1 0 PWL 0 -1.5V 6 4.5V
    V2 2 0 1.5VOLTS
    MODN1 2 1 0 0 M  10 3
    .MODEL M NMOS VTO = 1 NSUB = 1E15 TOX = 1000 UO = 800
    LEVEL = 1
    +CAPOP = 2
    .PLOT TRAN V(1) (0,5) LX18(M1)  LX19(M1) LX20(M1) (0,6E-
    13)
.END MOS CAPS
```

Using Subcircuits

Reusable cells are the key to saving labor in any CAD system, and this also applies to circuit simulation. To create a reusable circuit, it must be constructed as a subcircuit. Use parameters to expand the utility of a subcircuit. SPICE includes the basic subcircuit but does not provide for the consistent naming of nodes. Star-Hspice provides a simple method for the naming of the subcircuit nodes and elements: simply prefix the node or element with the subcircuit call name.

Figure 3-3: Subcircuit Representation



The following Star-Hspice input creates an instance named X1 of the INV cell macro, which consists of two MOSFETs, 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 subcircuit INV called by X1, the names are X1.MP and X1.MN. So to print the current through the MOSFETs:

```
.PRINT I (X1.MP)
```

Hierarchical Parameters

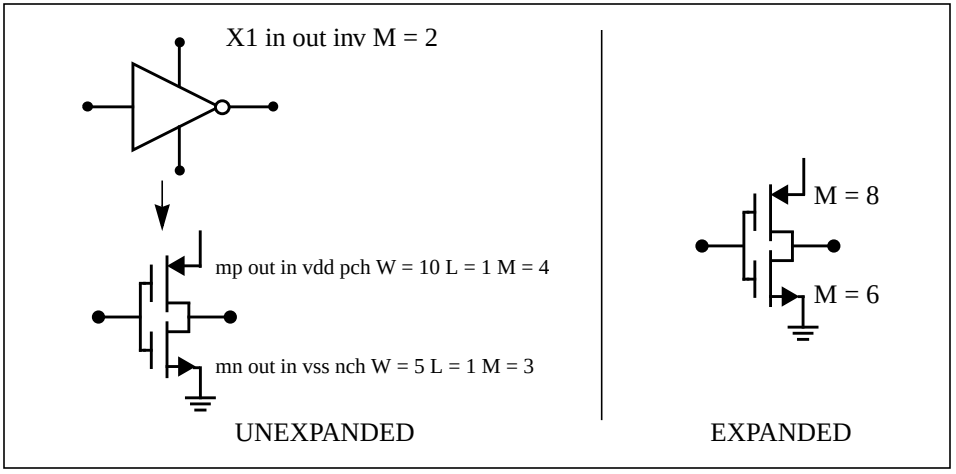
M (Multiply) Parameter

The most basic subcircuit parameter is the M or multiply parameter. This is actually a keyword common to all elements including subcircuits, except for voltage sources. The multiply parameter multiplies the internal component values to give the effect of making parallel copies of the element or subcircuit. To simulate the effect of 32 output buffers switching simultaneously, you only need to place one subcircuit:

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

Multiply works hierarchically. A subcircuit within a subcircuit is multiplied by the product of both levels.

Figure 3-4: Hierarchical Parameters Simplify Flip-flop Initialization



Example

```
X1 D Q Qbar CL CLBAR dlatch flip = 0
macro dlatch
+ D Q Qbar CL CLBAR flip = vcc
.nodeset v(din) = flip
xinv1 din qbar inv
```

```
xinv2 Qbar Q inv
m1 q CLBAR din nch w = 5 l = 1
m2 D CL din nch w = 5 l = 1
.eom
```

S (Scale) Parameter

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

Syntax

```
.option hier_scale=value
.option scale=value
X1 node1 node2 subname S = value
```

The `option hier_scale` statement defines how Star-Hspice interprets the *S* parameter, where *value* is either:

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

The `option scale` statement defines the original (default) scale of the sub-circuit. The specified `S` scale applies relative to this default scale of the sub-circuit.

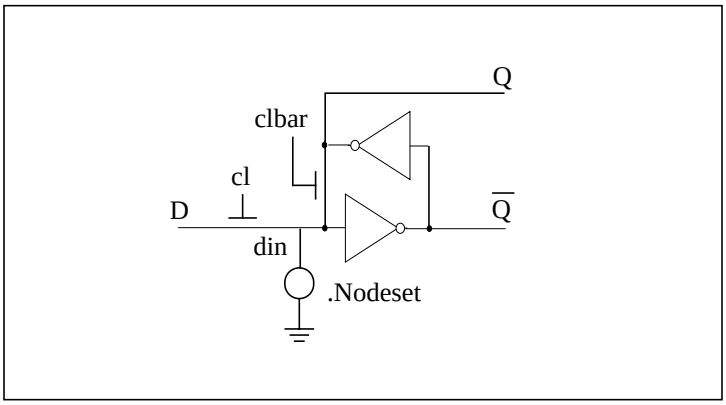
The scale in the *subname* sub-circuit is `value*scale`. Because sub-circuits can originate from multiple sources, scaling is multiplicative, or cumulative, throughout your design hierarchy. For example:

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

In this example, first the *X1* sub-circuit is scaled by the first *S* scaling value, `1u*(SCALE)`. Then, because scaling is cumulative, *X2* (a sub-circuit of *X1*) is scaled, in effect, by the *S* scaling values of **both** X1 and X2:

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

Figure 3-5: D Latch with Nodeset



There is no limit to the size or complexity of subcircuits; they may contain subcircuit references and any model or element statement. To specify subcircuit nodes in .PRINT or .PLOT statements, specify the full subcircuit path and node name.

Undefined Subcircuit Search

When a subcircuit call is in a data file that does not contain the subcircuit description, Star-Hspice automatically searches the:

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

Star-Hspice searches for the model reference name file with an *.inc* suffix. For example, if an undefined subcircuit such as “X 1 1 2 INV” is included in the data file, Star-Hspice searches the system directories for the file called *inv.inc* and when found, places it in the calling data file.

Discrete Device Libraries

Avant!'s Discrete Device Library (DDL) is a collection of Star-Hspice models of discrete components. The *$installdir/parts directory* contains the various subdirectories that make up the DDL. BJT, MESFET, JFET, MOSFET, and diode models are derived from laboratory measurements using Avant!'s ATEM discrete device characterization system. Behavior of op-amp, comparator, timer, SCR and converter models closely resembles that described in manufacturers' data sheets. Op-amp models are created using the built-in Star-Hspice op-amp model generator.

Note: *$installdir* is an environment variable whose value is the path name to the directory in which Star-Hspice is installed. That directory is called the installation directory. The installation directory contains subdirectories such as */parts* and */bin*, as well as certain files, such as a prototype *meta.cfg* file and Star-Hspice license files.

DDL Library Access

To include a DDL library component in a data file, use the X subcircuit call statement with the DDL element call. The DDL element statement includes the model name that is used in the actual DDL library file. For example, the following Star-Hspice 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 Chapter 310 and the chapters on specific types of devices 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, the designer can override offset voltage and gain and offset current. Since the DDL library devices are based on Star-Hspice circuit level models, the effects of supply voltage, loading, and temperature are automatically compensated for in a simulation.

Star-Hspice accesses DDL models in several ways on most computers:

1. An *hspice.ini* initialization file is created when the installation script is run. The search path for the DDL and vendor libraries is written into a .OPTIONS SEARCH = '<lib_path>' statement to give all users immediate access to all libraries. The models are automatically included on usage in the input netlist. When a model or subcircuit is referenced in the input netlist, the directory to which the = DDLPATH environment variable points is searched for a file with the same name as the reference name. This file is an include file, so its filename has the suffix *.inc*. The DDLPATH variable is set in the *meta.cfg* configuration file when Star-Hspice is installed.
2. Set .OPTIONS SEARCH = '<library_path>' in the input netlist. This method allows you to list personal libraries to be searched. The default libraries referenced in the *hspice.ini* file are searched first. Libraries are searched in the order in which they are encountered in the input file.
3. 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:

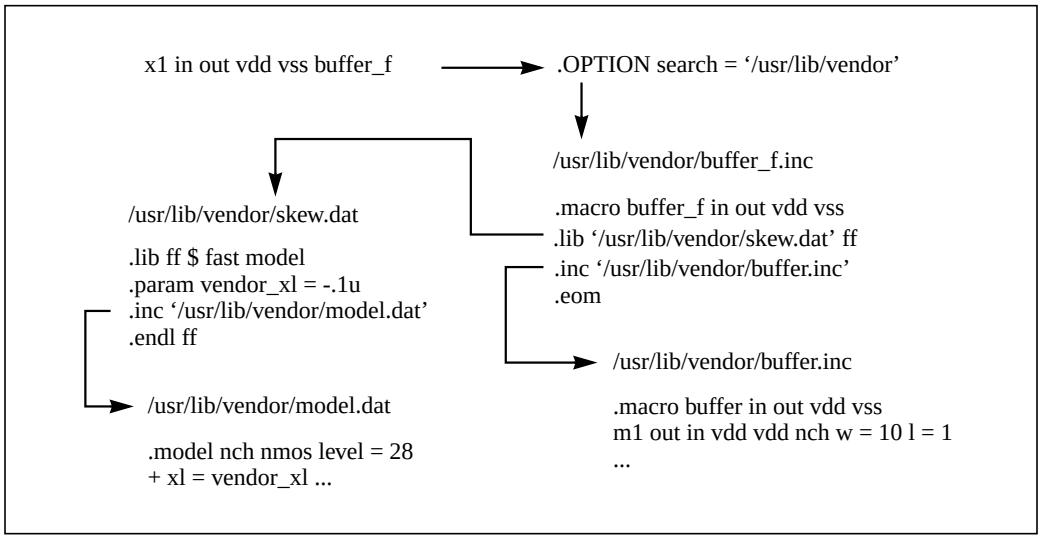
   ```
   .INCLUDE <path>/T2N2211.inc
   ```

   Since this method requires that each model be stored in its own *.inc* file, it is not generally useful, but it can be used for debugging new models when the number of models to be tested is small.

Vendor Libraries

The interface between commercial parts and circuit or system simulation is the vendor library. ASIC vendors provide comprehensive cells corresponding to inverters, gates, latches, and output buffers. Memory and microprocessor vendors generally supply input and output buffers. Interface vendors supply complete cells for simple functions and output buffers for generic family output. Analog vendors supply behavioral models. To avoid name and parameter conflicts, vendor cell libraries should keep their models within the subcircuit definitions.

Figure 3-6: Vendor Library Usage



Subcircuit Library Structure

Your library structure must adhere to the Star-Hspice implicit subcircuit .INCLUDE statement specification feature. This Star-Hspice function allows you to specify the directory that the subcircuit file (<*subname*>.*inc*) resides in, and then reference the name <*subname*> in each subcircuit call.

Component naming conventions require that each subcircuit be of the form <*subname*>.*inc* and stored in a directory that is accessible through the .OPTIONS SEARCH = '<*libdir*>' statement.

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

Using Standard Input Files

This section describes how to use standard input files.

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

Both Star-Hspice and AvanWaves read and write files related to the current circuit design. All files related to a design generally reside in one directory, although the output file is standard output on UNIX platforms and can be redirected.

Star-Hspice input file types and their standard names are listed in Table 3-1. These files are described in the following sections.

Table 3-1: Star-Hspice Input Files

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

Configuration File (*meta.cfg*)

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

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

Initialization File (*hspice.ini*)

User defaults are specified in an *hspice.ini* initialization file. If an *hspice.ini* file exists in the run directory, Star-Hspice includes its contents at the top of the Star-Hspice input file.

Other ways to include initialization files are to define "DEFAULT_INCLUDE = <filename>" in the system or in a *meta.cfg* file.

Typical uses of an initialization file are to set options (with an .OPTIONS statement) and for library access, as is done in the Avant! installation procedure.

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

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

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

Output Files

Star-Hspice produces various types of output files, as listed in the following table.

Table 3-2: Star-Hspice Output Files and Suffixes

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

Table 3-2: Star-Hspice Output Files and Suffixes

Output File Type	Extension

is either a sweep number or a hardcopy file number.

†Only created if a .POST statement is used to generate graphical data.

††Requires a .GRAPH statement or a pointer to a file exists in the meta.cfg file. This file is not generated by the PC version of Star-Hspice.

†††Only created if a .FFT statement is used.

The files are listed in Table 3-2 and described below.

Output listing can appear as *output_file* (no file extension), *output_file.lis*, or have a user-specified file extension, depending upon which format is used to start the simulation. *Output_file* is the output file specification, less extension. This file includes the following information:

- Name and version of simulator used
- Avant! message block
- Input file name
- User name
- License details
- Copy of the input netlist file
- Node count
- Operating point parameters
- Details of volt drop, current, and power for each source and subcircuit
- Low resolution plots originating from the .PLOT statement
- Results of .PRINT statement
- Results of .OPTIONS statements

Transient analysis results are placed in *output_file*. *tr#*, where # is specified as 0-9 or a-z following the -n argument. This file contains a list of transient analysis numerical results. It is the result of an input file .TRAN statement together with an .OPTION POST statement to create a post-analysis file. The output file is in proprietary binary format if POST = 0 or 1, or ASCII format if POST = 2. The explicit expressions POST = BINARY, POST=ASCII may also be used.

Transient analysis measurement results are written to *output_file.mt#*. This output file is the result of an input file .MEASURE TRAN statement.

DC analysis results appear in *output_file.sw#,* which is produced as a result of a .DC statement. This file contains the results of the applied stepped or swept DC parameters defined in that statement. The results may include noise, distortion, or network analysis.

DC analysis measurement results are given in the file *output_file. ms#* when a .MEASURE DC statement exists in the input file.

AC analysis results are placed in *output_file.ac#.* These results contain a listing of output variables as a function of frequency, according to user specification following the .AC statement.

AC analysis measurement results appear in *output_file.ma#* when a .MEASURE AC statement exists in the input file.

Hardcopy graph data are placed in *output_file.gr#,* which is produced as a result of a *.GRAPH* statement. It is in the form of a printer file, typically in Adobe PostScript or HP PCL format. This facility is not available in the PC version of Star-Hspice.

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

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

Subcircuit cross-listing is automatically generated and written into *output_file.pa#* when the input netlist includes subcircuits. It relates the subcircuit node names in subcircuit call statements to the node names used in the corresponding subcircuit definitions.

Output status is named with the output file specification, with a *.st#* extension, and 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 used by Star-Hspice on difficult circuits

The information in this file is useful in diagnosing problems, particularly when communicating with Avant! Customer Support.

Operating point node voltages are DC operating point initial conditions stored by the .SAVE statement.

Using the Star-Hspice Command

You can start Star-Hspice in either a prompting mode or a nonprompting command line mode.

Prompting Script Mode

Use the following procedure to start Star-Hspice in the prompting mode.

1. cd to your Star-Hspice run directory and type:

   ```
   hspice
   ```

2. The following prompt appears:

   ```
   Enter input file name:
   ```

3. Enter the name of your Star-Hspice input netlist file. If you do not include a file name extension, Star-Hspice looks for the file name with an *.sp* extension.

   If no file name exists with the name you enter, the following message appears and the Star-Hspice startup script terminates:

   ```
   **error** Cannot open input file <filename>
   ```

4. The following prompt appears:

   ```
   Enter output file name or directory:
   [<filename>.lis]
   ```

5. Enter the path and name you want to give the Star-Hspice output listing file. The default is the input file name with a *.lis* extension.

6. A numbered list of the Star-Hspice versions that are available appears, followed by a prompt to specify the version you want to run. Enter the number in the list of the Star-Hspice version you want to run.

7. For releases of Star-Hspice prior to Release H93A.02, the following prompt appears:

   ```
   How much memory is needed for this run?
   ```

Enter the number of 8-byte words of memory you want to allocate for the Star-Hspice run.

8. The following prompt appears:

```
The default is to use the standard system priority.
Run Star-Hspice at a lower priority? (y,n) [n]
```

9. To use the default priority, enter n, or just press Return.

   To specify the priority, enter y. The following prompt appears:

```
HINT: The larger the number the lower the priority.
Enter the priority scheduling factor: (5 10 15 20) [15]
```

   The default is 15. Enter your choice from the list of factors.

The Star-Hspice run begins.

Nonprompting Command Line Mode

Star-Hspice accepts the following arguments when run in the nonprompting command line mode:

```
hspice <-i> <path/>input_file <-v HSPICE_version>
+<-n number> <-a arch> <-o path>/output_file>
```

where:

input_file	Specifies the input netlist file name, for which an extension <.*ext*> is optional. If no input filename extension is provided in the command, Star-Hspice searches for a file named <*input_file*>.*sp*. The input file can be preceded by -i. The input filename is used as the root filename for the output files. Star-Hspice also checks to see if there is an initial conditions file (.*ic*) with the input file root name. The following is an example of an input file name: */usr/sim/work/rb_design.sp* *where:* */usr/sim/work/ is the directory path to the design* *rb_design is the design root name* *.sp is the filename suffix*
-v	Specifies the version of Star-Hspice to use.
-n	Specifies the number at which to start numbering output data file revisions (*output_file.tr#*, *output_file.ac#*, *output_file.sw#*, where # is the revision number).
-a	Is an argument that overrides the default architecture.

Available Star-Hspice command arguments are listed in Table 3-3.

Table 3-3: Star-Hspice Command Options

Option	Description
-a <arch>	Platform architecture. Choices are: • sun4, sol4 (SparcStation, Ultra) • pa (HP 700/800/9000) • alpha (DEC ALPHA) • rs (IBM RS6000) • sgi (SGI) • pc (Windows 95/NT)
-i <input_file>	Name of the input netlist file. If no extension is given, *.sp* is assumed.
-m <mem_needed>	Amount of memory requested for the simulation, in 8-byte words (only required for Star-Hspice releases prior to Release H93A.01)
-n <number>	Revision number at which to start numbering *.gr#*, *.tr#*, and other output files. By default, the file numbers start at zero: *.gr0*, *.tr0*, and so on. This option allows you to specify the number (-n 7 for *.gr7*, *.tr7*, for example).
-o <output_file>	Name of the output file. If no extension is given, *.lis* is assigned.
-r <remote_host>	Name of the machine on which to run the simulation
-v <version>	Star-Hspice version. Choices are determined at the time of installation by the Star-Hspice installation script.
-x	Displays the Star-Hspice script on the window as it runs

You do not need to include a filename extension in the output file specification. Star-Hspice names it *output_file.lis*. In output file names, Star-Hspice considers everything up to the final period to be the root filename, and everything following the last period to be the filename extension.

If you do not enter an output filename with the -o option, the input root filename is used as the output file root filename. If you include the extension *.lis* in the filename you enter with -o, Star-Hspice does not append another *.lis* extension to the output file root filename.

If no output file is specified, output is directed to the terminal. Use the following syntax to redirect the output to a file instead of the terminal:

```
hspice input_file <-v HSPICE_version> <-n number> <-a arch>
> output_file
```

For example, for the invocation command

```
hspice demo.sp -v /usr/meta/96 -n 7 -a sun4 > demo.out
```

where:

demo.sp	Is the input netlist file; the *.sp* extension to the input filename is optional
-v /usr/meta/96	Specifies the version of Star-Hspice to use
-n 7	Starts the output data file revision numbers at 7: *demo.tr7*, *demo.ac7*, and *demo.sw7*
-a sun4	Overrides the default platform
>	Redirects the program output listing to *demo.out*

Sample Star-Hspice Commands

Some additional examples of Star-Hspice commands are explained below.

- **hspice -i demo.sp**

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

- **hspice -i demo.sp -o demo**

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

- **hspice -i rbdir/demo.sp**

"demo" is the root name. Output files *demo.lis*, *demo.tr0*, and *demo.st0* are written in the directory where the Star-Hspice command is executed. Output file *demo.ic* is written in the same directory as the input source, that is, *rbdir*.

- **`hspice -i a.b.sp`**

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

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

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

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

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

  "a.b" is the root for the *.ic* file. The *.ic* file is written in a file named *a.b.ic*.

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

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

  "a.b" is the root for the *.ic* file. The *.ic* file is written in a file named *indir/a.b.ic*.

  "d.e" is the root for the output files.

Improving Simulation Performance Using Multithreading

Star-Hspice simulations involve both model evaluations and matrix solutions. Running model evaluations concurrently on multiple CPUs using multithreading can significantly improve simulation performance. In most cases, the model evaluation will dominate. To determine how much time is spent in model evaluation and solving, specify `.option acct = 2` in the netlist. Using multithreading results in faster simulations with no loss of accuracy.

Multithreaded (MT) Star-Hspice is supported on Sun Solaris 2.5.1 (SunOS 5.5.1) and on Windows NT as a prerelease version using win32 threads.

Running Star-Hspice-MT

You can run Star-Hspice-MT using the syntax described below.

Sun Solaris Platform

Enter on the command line:

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

Windows NT Platform

Under the Windows NT DOS prompt type:

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

Note: If the *#num* is omitted, the number of threads will be set to the number of online CPUs.
If you omit the `-o output_file` option, the result will be printed to the standard output.

Under Windows NT explorer:

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

In Windows, the program will automatically detect and use the number of online CPUs.

Under the Avant! HSPUI interface:

1. Select the correct version of *hsp_mt.exe* in the Version Combo Box.
2. Select the correct number of processors in the MT Option Box.
3. Click the **Open** button to select the input netlist file.
4. Click the **Simulate** button to start the simulation.

Performance Improvement Estimations

For multithreaded Star-Hspice, the CPU time is:

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

where:

Tserial	Represents the Star-Hspice calculations that are not threaded
Tparallel	Represents the threaded Star-Hspice calculations
Ncpu	The number of CPUs used. *Toverhead* is the overhead from multithreading. Typically, this represents a small fraction of the total run time.

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
```

For six CPUs the speedup is:

```
T1cpu/(0.2T1cpu + 0.8T1cpu/6cpus) = 3.0
```

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

Using PKG and EBD Simulation

PKG & EBD simulation support the package data from [Package], [Pins] and [Define Package Model] sections in *.ibs, *.pkg, and *.ebd files. You can simulate the packaging and the board-level trace effects in the whole system, using Star-Hspice. You can also simulate the packaging effect and the pin-connected trace effect stand-alone, with the additional stimulus and loads on the corresponding pins.

Note: The subcircuit interface port names must be same as the pin names listed in IBIS file.

Options Statements

To support the PKG and EBD feature in system simulation, the netlist must include the following lines:

```
.option PKGMAP="pkg.map"
.option EBDMAP="ebd.map"
.option PKGTYP=RLC / T_LINE
.option EBDTYP=RLC / T_LINE
```

Parameter	Description
PKGMAP	Specifies the name of a map file, which lists the relationship between the Hspice sub-circuit name and the IBIS component name. You can assign this option (with the map file) up to 40 times.

Parameter	Description
EBDMAP	Specifies the name of a map file, which lists the relationship between the Hspice sub-circuit name and: n The IBIS board-level module. n The X element name in the sub-circuit. n The on-board component. You can assign this option (with the map file) up to 40 times.
PKGTYP	Specifies the types of elements to use, to represent the package effect. n If the value is RLC (the default), Hspice uses RLC elements as the parasitic packaging elements. n If the value is T_LINE, Hspice uses the transmission line. n If you specify the package data in matrix form, Hspice uses the W element. n If the package data is in the [Package] or [Pin] section, Hspice uses the RLC element. Use this option only if you use the section form to specify the package data in [Define Package Model], and the section length is not 0.
EBDTYP	Specifies the type of elements to use, to represent the board-level pin connected traces. ■ If the value is RLC, Hspice selects RLC element netlists as the traces. ■ If the value is T_LINE, Hspice uses the transmission line.

The PKG map file format is:

```
HSP_SUBCIRCUIT_NAME_1 IBIS_FILE_NAME1 COMPONENT_NAME1
HSP_SUBCIRCUIT_NAME_2 IBIS_FILE_NAME2 COMPONENT_NAME2
...                   ...             ...
```

Parameter	Description
HSP_SUBCIRCUIT_NAME_1	Specifies the name of the sub-circuit to consider with the package effect.
IBIS_FILE_NAME1	Specifies the name of the IBIS file that includes the package information for the HSP_SUBCIRCUIT_NAME_1 sub-circuit.
COMPONENT_NAME1	Specifies the name of the component that corresponds to the sub-circuit.

The EBD map file format is:

```
[FILE NAME] filename
[EBD MAP DATA]
[BOARD LEVEL SUBCIRCUIT] subcircuit_name
[EBD FILE Name]          EBD_file_name
x_element_name1 component_on_board_name1
x_element_name2 component_on_board_name2
...
[END EBD MAP DATA]
```

Parameter	Description
[FILE NAME]	Keyword. The *filename* argument specifies the file name, to verify file consistency.
[EBD MAP DATA] [END EBD MAP DATA]	Between these two keywords is information about the relationship between the Hspice sub-circuit and a board-level module. You can specify multiple [EBD MAP DATA] & [END EBD MAP DATA] blocks, but you can include only one board-level module in each block.
[BOARD LEVEL SUBCIRCUIT]	A keyword. The *subcircuit_name* argument specifies the Hspice sub-circuit to consider with the trace effect.

Parameter	Description
[EBD FILE Name]	A keyword. The *EBD_file_name* argument specifies the name of the file that includes the pin-related trace information for the Hspice sub-circuit.
X_ELEMENT_NAME1	Specifies the X element in the Hspice subcircuit, which corresponds to the on-board component specified by COMPONENT_ON_BOARD_NAME1.
COMPONENT_ON_BOARD_NAME1	Specifies the on-board component referenced in the EBD file, which corresponds to the X_ELEMENT_NAME1.

The PKG & EBD effect are shown from the first simulation.

System-Level PKG and EBD Simulation

To simulate an entire system, including PKG & EBD information, associate the Hspice netlist with the PKG & EBD information. To do this, use the related options, the PKG map files, and the END map files. You can obtain detailed information from your local Technical Support teams.

Stand-alone PKG Simulation

Use the Stand-alone PKG Simulation feature to focus only on studying the packaging effect. To do this, follow these steps:

1. Use the `hspice` command to produce a subcircuit that corresponds to the package component.
2. Prepare an Hspice netlist that calls the generated subcircuit,.

   The subcircuit is added with the suitable stimulus and loads.
3. Simulate the Hspice netlist

The command syntax for pkg2ckt is:

```
hspice -t RLC/T_LINE -p ibis_file -c component_name
```

Parameter	Description
hspice	Program name.
-t	Specifies the elements that handle the package effect, either: n RLC (the default) for RLC elements, or n T_LINE (transmission line) for the W element. This argument is not available for the package data in matrix form.
-p	Specifies that the next argument is *ibis file*.
ibis_file	Specifies the name of the IBIS file that includes the package data. The file extension must be either `.ibs` or `.pkg`.
-c	Specifies that the next argument is *component name*.
component_name	Specifies the name of the component for which simulation focuses only on the package (PKG).

The generated sub-circuit file name is the *component_name* with a `.inc` extension.

The number of interface nodes is twice the number of pins listed in the ibis file, for internal nodes and external pins.

- Half of these nodes use the pin names from the ibis file. These node names, without new prefixes, are pins that connect outside of the circuit.
- The remaining interface nodes use the same name as the pin listed in the ibis file, but with the `PO_` prefix. These internal nodes connect to the original sub-circuit interface node, based on their names.

Stand-alone EBD Simulation

To study *only* the pin-connected trace effect, use the Stand-alone, Pin-connected Trace Simulation feature, as described in the following steps:

1. Use the `hspice` command to create a sub-circuit that corresponds to the pin-connected trace component.
2. Prepare an Hspice netlist that calls the generated sub-circuit.

   The subcircuit is added, with suitable stimulus and loads.
3. Simulate the Hspice netlist.

The command syntax for pkg2ckt is:

```
hspice -t RLC/T_LINE -e ibis_file -o output_subckt_name
```

Parameter	Description
hspice	Program name.
-t	Specifies the elements that handle the package effect, either: n RLC (the default) for RLC elements, or n T_LINE (transmission line) for the W element. This argument is not available for the package data in matrix form.
-e	Specifies that the next argument is *ibis file*.
ibis_file	Specifies the name of the IBIS file that includes the pin-connected trace data. The file extension must be `.ebd`.
-o	Specifies that the next argument is *subckt_name*.
subckt_name	Specifies the name of the sub-circuit for which simulation focuses only on the pin-connected trace.

The generated file name is the *subckt_name* with a `.inc` extension.

The number of interface nodes consists of the *pins* listed in the `.ebd` file, and the *nodes* listed in the `.ebd` file.

- The interface node from the pin uses the same name as the pin.
- The interface node from node in the `.ebd` file uses a name in the following format:

```
{ ref_name + "_" + pin_name [ + digit ] }
```

Limitation

In any specified package, the number of pins must not exceed 512.

If you use an EBD file, the EBD simulation feature does not support series components (such as resistors), because the current IBIS specification does not let you specify a resistance value. Currently, IBIS describes *only* the board pin - connect traces; IBIS ignores the other on-board traces.

You can use the PKG & EBD features for:

- Simulation for both PKG and EBD.
- System simulation of a PKG.
- System simulation using EBD.
- Simulation of a PKG circuit to a PKG subcircuit.
- Simulation of an EBD circuit to an EBD subcircuit.

You can select the parasitic element type as either RLC, or W transmission lines.

- The command:

  ```
  hspice test_ebd1.sp
  ```

  inserts the parasitic elements, due to the package and board-pin connected traces into the original netlist. It then performs the simulation as usual.

- The command:

  ```
  hspice -t RLC -p test_9_1_1.ibs -c test_9_1_1
  ```

  outputs a sub-circuit into the `test_9_1_1.inc` file, which describes the package parasitic effects for the R/L/C elements.

- The command:

  ```
  hspice -t T_LINE -e test_ebd1.ebd -o test_ebd
  ```

  outputs the sub-circuit information into the `test_ebd.inc` file, which describes the board parasitic effect for the W transmission line.

Avant!

Chapter 4
Using Elements

This chapter describes the syntax for the basic elements of a circuit netlist. Refer to the *True-Hspice Device Models Reference Manual* for detailed syntax descriptions and model descriptions.

This chapter covers the following topics:

Passive Elements

Resistors

The general syntax for including a resistor element in a Star-Hspice netlist is:

General Form

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

where the resistance can be either a value (in units of ohms) or an equation. The only required fields are the two nodes and the resistance or the model name. If the parameter labels are used, the optional arguments may come in any order, although the nodes and model name must come first. If a resistor model is specified (see Chapter 2, “Using Passive Device Models”, in the *True-Hspice Device Models Reference Manual*), the resistance value is optional.

The arguments are defined as:

Rxxx	Resistor element name. Must begin with “R”, which can be followed by up to 1023 alphanumeric characters.
n1	Positive terminal node name
n2	Negative terminal node name
mname	Resistor model name. This name is used in elements to reference a resistor model.
R = resistance	Resistance value at room temperature. This may be a numeric value or parameter in ohms, or a function of any node voltages, branch currents, or any independent variables such as time, frequency (HERTZ), or temperature.

TC1	First-order temperature coefficient for the resistor. Refer to Chapter 2, “Using Passive Device Models”, in the *True-Hspice Device Models Reference Manual* for temperature-dependent relations.
TC2	Second-order temperature coefficient for the resistor
SCALE	Element scale parameter; scales resistance by its value. Default = 1.0.
M	Multiplier used to simulate parallel resistors. For example, to represent two parallel instances of a resistor, set M = 2 to multiply the number of resistors by 2. Default = 1.0.
AC	AC resistance used in the AC analysis. Default = Reff.
DTEMP	Temperature difference between the element and the circuit in Celsius. Default = 0.0.
L	Resistor length in meters. Default = 0.0, if L is not specified in a resistor model.
W	Resistor width. Default = 0.0, if W is not specified in the model.
C	Capacitance connected from node n2 to bulk. Default = 0.0, if C is not specified in a resistor model.

Example

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

```
R1 Rnode1 Rnode2 100
```

Resistor RC1 connected 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
```

Resistor Rterm connected from node input to ground with a resistance determined by the square root of the analysis frequency (nonzero for AC analysis only).

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

Resistor Rxxx 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
```

Capacitors

The general syntax for including a capacitor element in a Star-Hspice netlist is:

General Form

```
Cxxx n1 n2 <mname> <C = >capacitance <<TC1 = >val>
+ <<TC2 = >val> <SCALE = val> <IC = val> <M = val>
+ <W = val> <L = val> <DTEMP = val>
```

or

```
Cxxx n1 n2 <C = >'equation' <CTYPE = val>
+ <above options...>
```

Polynomial Form

```
Cxxx n1 n2 POLY c0 c1... <above options...>
```

where the capacitance can be specified 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 the parameter labels are used, the optional arguments may come in any order, although the nodes and model name must come first. If a capacitor model is specified (see Chapter 2, "Using Passive Device Models", in the *True-Hspice Device Models Reference Manual*), the capacitance value is optional.

If the equation form of the capacitance specification is used, the CTYPE parameter is used to determine the method of capacitance charge calculation. The calculation is different depending on whether a self-referential voltage is used in the equation (that is. the voltage across the capacitor whose capacitance is determined by the equation).

To avoid syntactic conflicts, if a capacitor model exists using the same name as a parameter used to specify the capacitance, the model name is taken. In the following example, C1 assumes the value of capacitance determined using the model and not the parameter.

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

The arguments are defined as:

Cxxx	Capacitor element name. Must begin with a “C”, which can be followed by up to 1023 alphanumeric characters.
n1	Positive terminal node name
n2	Negative terminal node name
mname	Capacitance model name. This name is used in elements to reference a capacitor model.
C = capacitance	Capacitance at room temperature as a numeric value or parameter in farads.
TC1	First-order temperature coefficient for the capacitor. Refer to Chapter 2, “Using Passive Device Models”, in the *True-Hspice Device Models Reference Manual* for temperature-dependant relations.
TC2	Second-order temperature coefficient for the capacitor
SCALE	Element scale parameter, scales capacitance by its value. Default = 1.0.
IC	Initial voltage across the capacitor in volts. This value is used as the DC operating point voltage when UIC is specified in the .TRAN statement and is overridden by the .IC statement.
M	Multiplier used to simulate multiple parallel capacitors. Default = 1.0
W	Capacitor width in meters. Default = 0.0, if W is not specified in a capacitor model.
L	Capacitor length in meters. Default = 0.0, if L is not specified in a capacitor model.
DTEMP	Element temperature difference with respect to the circuit temperature in Celsius. Default = 0.0.

C = 'equation'	Capacitance at room temperature specified as a function of any node voltages, branch currents, or any independent variables such as time, frequency (HERTZ), or temperature.
CTYPE	Determines capacitance charge calculation for elements with capacitance equations. If capacitance equation is a function of v(n1,n2), set CTYPE = 1. This setting must be used correctly to ensure proper capacitance calculations and hence simulation results. Default = 0.
POLY	Keyword to specify capacitance given by a polynomial.
c0 c1...	Coefficients of a polynomial in voltage describing the capacitor value. c0 represents the magnitude of the 0th order term, c1 represents the magnitude of the 1st order term, and so on. Note that the coefficients can not be parameterized.

Example

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

```
C1 1 2 20p
```

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

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

Capacitor Cload connected from node driver to node output with a capacitance determined by the voltage on node capcontrol times 1E-6, and an initial voltage across the capacitor of 0 volts.

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

Capacitor C99 connected from node in to node out with a capacitance 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
```

Inductors

The general syntax for including an inductor element in a Star-Hspice netlist is:

General Form

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

or

```
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...>
```

where the inductance can be either a value (in units of henries), an equation, a polynomial of the current or a magnetic winding. The only required fields are the two nodes and the inductance or model name. If the parameter labels are used, the optional arguments may come in any order, although the nodes and model name must come first. If an inductor model is specified (see Chapter 2, "Using Passive Device Models", in the *True-Hspice Device Models Reference Manual*), the inductance value is optional.

The arguments are defined as:

Lxxx	Inductor element name. Must begin with L, which can be followed by up to 1023 alphanumeric characters.
n1	Positive terminal node name.
n2	Negative terminal node name.
TC1	First-order temperature coefficient for the inductor. Refer to Chapter 2, "Using Passive Device Models", in the *True-Hspice Device Models Reference 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. This value is used as the DC operating point voltage when UIC is specified in the .TRAN statement and is overridden by the .IC statement.
L = inductance	Inductance value. This may be a numeric value or parameter in henries, or a function of any node voltages, branch currents, or any independent variables such as time, frequency (HERTZ), or temperature.
M	Multiplier used to simulate parallel inductors. Default = 1.0.
DTEMP	Temperature difference between the element and the circuit in Celsius. Default = 0.0.
R	Resistance of inductor in ohms. Default = 0.0.
L = 'equation'	Inductance at room temperature specified as a function of any node voltages, branch currents, or any independent variables such as time, frequency (HERTZ), or temperature.
LTYPE	Determines inductance flux calculation for elements with inductance equations. If inductance equation is a function of i(Lxxx), set LTYPE = 1. This setting must be used correctly to ensure proper inductance calculations and hence simulation results. Default = 0.
POLY	Keyword to specify inductance given by a polynomial.
c0 c1...	Coefficients of a polynomial in current describing the inductor value. c0 represents the magnitude of the 0th order term, c1 represents the magnitude of the 1st order term, and so on.

NT = turns Number representing the number of turns of an inductive magnetic winding.

Example

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

```
L1 coilin coilout 100n
```

Inductor Lloop connected from node 12 to node 17 with an inductance of 1 microhenry, and temperature coefficients of 0.001 and 0.

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

Inductor Lcoil connected from node input to ground with an inductance determined by the product of the current through the inductor and 1E-6.

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

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

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

Inductor L connected from node 1 to node 2 as a magnetic winding element with 10 turns of wire.

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

Mutual Inductors

The general syntax for a mutual inductor element in a Star-Hspice netlist is:

General Form

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

Mutual Core Form

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

where “coupling” is a unitless value from zero to one representing the coupling strength. If you use parameter labels, the nodes and model name must be first. but the optional arguments can be in any order. If you specify an inductor model (see Chapter 2, “Using Passive Device Models”, in the *True-Hspice Device Models Reference Manual*), the inductance value is optional.

The arguments are defined as:

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 greater than 0 and less than or equal to 1. If K is negative, the direction of coupling is reversed. This reversal is equivalent to reversing the polarity of either of the coupled inductors. The K = coupling syntax should be used when using a parameterized value or an equation.
Kaaa	Saturable core element name. Must begin with “K”, which can be followed by up to 1023 alphanumeric characters.
Lbbb, Lccc, Lddd	The names of the windings about the Kaaa core. One winding element is required, and each winding element must have the magnetic winding syntax.

mname	Saturable core model name. See Chapter 2, “Using Passive Device Models”, in the *True-Hspice Device Models Reference Manual* for model information.
MAG = magnetization	Initial magnetization of the saturable core. Can be set to +1, 0 and -1, where +/- 1 refer to positive and negative values of the model parameter BS (see Chapter 2, “Using Passive Device Models”, in the *True-Hspice Device Models Reference Manual*).

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

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

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

Example

Inductors Lin and Lout are coupled with a coefficient of 0.9.

```
K1 Lin Lout 0.9
```

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

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

The two mutual inductors K1 and K2 couple L1 and L2, and L2 and L3, respectively. The coupling coefficients are 0.98 and 0.87. Star-Hspice automatically calculates the mutual inductance between L1 and L3, with a coefficient of 0.98*0.87 = 0.853.

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

Active Elements

Diode Element

The general syntax for a diode element in a Star-Hspice netlist is:

Geometric (LEVEL = 1) and 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>
```

or

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

Fowler-Nordheim (LEVEL = 2) Form

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

The only required fields are the two nodes and the model name. If the parameter labels are used, the optional arguments may come in any order, although the nodes and model name must come first.

The arguments are as follows:

Dxxx	Diode element name. Must begin with “D”, which can be followed by up to 1023 alphanumeric characters.
nplus	Positive terminal (anode) node name. The series resistor of 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 diode model LEVEL = 1 and square meters for diode model LEVEL = 3). This affects saturation currents, capacitances and resistances (diode model parameters IK, IKR, JS, CJO and RS). Area factor for diode model LEVEL = 1 is not affected by the SCALE option. Default = 1.0. Overrides AREA from the diode model. If unspecified, is calculated from width and length specifications.
PJ	Periphery of junction (unitless for diode model LEVEL = 1 and meters for diode model LEVEL = 3). Overrides PJ from the diode model. If unspecified, calculated from the width and length specifications.
WP	Width of polysilicon capacitor in meters (for diode model LEVEL = 3 only). Overrides WP in diode model. Default = 0.0.
LP	Length of polysilicon capacitor in meters (for diode model LEVEL = 3 only). Overrides LP in diode model. Default = 0.0.
WM	Width of metal capacitor in meters (for diode model LEVEL = 3 only). Overrides WM in diode model. Default = 0.0.
LM	Width of metal capacitor in meters (for diode model LEVEL = 3 only). Overrides LM in diode model. Default = 0.0.
OFF	Sets initial condition to OFF for this element in DC analysis. Default = ON.
IC = vd	Initial voltage across the diode element. This value is used when the UIC option is present in the .TRAN statement and is overridden by the .IC statement.
M	Multiplier to simulate multiple diodes in parallel. All currents, capacitances and resistances are affected by the setting of M. Default = 1.

DTEMP	The difference between the element temperature and the circuit temperature in Celsius. Default = 0.0.
W	Width of the diode in meters (diode model LEVEL=3 only)
L	Length of the diode in meters (diode model LEVEL = 3 only)

Example

Diode D1 with anode and cathode connected to nodes 1 and 2 where the diode model is given by diode1.

```
D1 1 2 diode1
```

Diode Dprot with anode and cathode connected to node output and ground references diode model firstd and specifies an area of 10 (unitless for LEVEL = 1 model) with the diode OFF as an initial condition.

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

Diode Ddrive with anode and cathode connected to nodes driver and output with a width and length of 500 microns and references diode model model_d.

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

Bipolar Junction Transistors (BJTs) Element

The general syntax for including a BJT element in a Star-Hspice netlist is:

General Form

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

or

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

The only required fields are the collector, base and emitter nodes, and the model name. The nodes and model name must come first.

The arguments are as follows:

Qxxx	BJT element name. Must begin with “Q”, which can be followed by 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. Can also be set in the BJT model with the parameter BULK.
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 to OFF for this element in DC analysis. Default = ON.
IC = vbeval, vceval, VBE, VCE	Initial internal base-emitter voltage (vbeval) and collector-emitter voltage (vceval). These are used when UIC is present in the .TRAN statement and is overridden by the .IC statement.
M	Multiplier to simulate multiple BJTs in parallel. All currents, capacitances and resistances are affected by the setting of M. Default = 1.
DTEMP	The difference between the element temperature and the circuit temperature in Celsius. Default = 0.0.
AREAB	Base area multiplying factor that affects currents, resistances and capacitances. Default = AREA.
AREAC	Collector area multiplying factor that affects currents, resistances and capacitances. Default = AREA.

Example

BJT Q1 with collector, base, and emitter connected to nodes 1, 2 and 3 where the BJT model is given by model_1.

```
Q1 1 2 3 model_1
```

BJT Qopamp1 with collector, base, and emitter connected to nodes c1, b3 and e2 and the substrate connected to node s. The BJT model is given by 1stagepnp and the area factors AREA, AREAB and AREAC are 1.5, 2.5 and 3.0, respectively.

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

BJT Qdrive with collector, base, and emitter connected to nodes driver, in and output with an area factor of 0.1 and references BJT model model_npn.

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

JFETs and MESFETs

The general syntax for including a JFET or MESFET element in a Star-Hspice netlist is:

General Form

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

or

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

The only required fields are the drain, gate and source nodes, and the model name. The nodes and model name must come first.

The arguments are as follows:

Jxxx	JFET or MESFET element name. Must begin with “J”, which can be 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). These are used when UIC is present in the .TRAN statement and is overridden by the .IC statement.
M	Multiplier to simulate multiple JFETs or MESFETs in parallel. All currents, capacitances, and resistances are affected by the setting of M. Default = 1.
DTEMP	The difference between the element temperature and the circuit temperature in Celsius. Default = 0.0.

Example

JFET J1 with drain, source, and gate connected to nodes 1, 2 and 3 where the JFET model is given by model_1.

```
J1 1 2 3 model_1
```

JFET Jopamp1 with drain, gate, and source connected to nodes d1, g3 and s2 and the bulk connected to node b. The JFET model is given by 1stage and the area is given as 100 microns.

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

JFET Jdrive with drain, gate, and source connected to nodes driver, in and output with a width and length of 10 microns and references JFET model model_jfet.

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

MOSFETs

The general syntax for including a MOSFET element in a Star-Hspice netlist is:

General Form

```
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>
```

or

```
.OPTION WL
Mxxx nd ng ns <nb> mname <width> <length>
+ <other options...>
```

The only required fields are the drain, gate and source nodes, and the model name. The nodes and model name must come first. The second syntax is used in conjunction with the .OPTION WL statement that allows exchanging the width and length options when no label is given.

The arguments are as follows:

Mxxx	MOSFET element name. Must begin with “M”, which can be 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. Can be set in MOSFET model using parameter BULK.
mname	MOSFET model name reference
L	MOSFET channel length in meters. This parameter overrides DEFL in an OPTIONS statement. Default = DEFL with a maximum of 0.1m.
W	MOSFET channel width in meters. This parameter overrides DEFW in an OPTIONS statement. Default = DEFW.
AD	Drain diffusion area. Overrides DEFAD in the OPTIONS statement. Default = DEFAD only when the MOSFET model parameter ACM = 0.
AS	Source diffusion area. Overrides DEFAS in the OPTIONS statement. Default = DEFAS only when the MOSFET model parameter ACM = 0.
PD	Perimeter of the drain junction, including the channel edge. Overrides DEFPD in the OPTIONS statement. Default = DEFAD when the MOSFET model parameter ACM = 0 or 1, and default = 0.0 when ACM = 2 or 3.
PS	Perimeter of the source junction, including the channel edge. Overrides DEFPS in the OPTIONS statement. Default = DEFAS when the MOSFET model parameter ACM = 0 or 1, and default = 0.0 when ACM = 2 or 3.
NRD	Number of squares of drain diffusion for resistance calculations. Overrides DEFNRD in the OPTIONS statement. Default = DEFNRD when the MOSFET model parameter ACM = 0 or 1, and default = 0.0 when ACM = 2 or 3.

NRS	Number of squares of source diffusion for resistance calculations. Overrides DEFNRS in the OPTIONS statement. Default = DEFNRS when the MOSFET model parameter ACM = 0 or 1, and default = 0.0 when ACM = 2 or 3.
RDC	Additional drain resistance due to contact resistance with 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 with units of ohms. This value overrides the RSC setting in the MOSFET model specification. Default = 0.0.
OFF	Sets initial condition to OFF for this element in DC analysis. Default = ON. **Note:** this command does not work for depletion devices.
IC = vds, vgs, vbs	Initial voltage across the external drain and source (vds), gate and source (vgs), and bulk and source terminals (vbs). These are used when UIC is present in the .TRAN statement and are overridden by the .IC statement.
M	Multiplier to simulate multiple MOSFETs in parallel. All channel widths, diode leakages, capacitances and resistances are affected by the setting of M. Default = 1.
DTEMP	The difference between the element temperature and the circuit temperature in Celsius. Default = 0.0.
GEO	Source/drain sharing selector for MOSFET model parameter value ACM = 3. Default = 0.0.
DELVTO	Zero-bias threshold voltage shift. Default = 0.0.

Example

MOSFET M1 with drain, gate, and source connected to nodes 1, 2 and 3 where the MOSFET model is given by model_1.

```
M1 1 2 3 model_1
```

MOSFET Mopamp1 with drain, gate, and source connected to and nodes d1, g3 and s2 and the bulk connected to node b. The MOSFET model is given by 1stage and the length and width of the gate are given as 2 and 10 microns, respectively.

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

MOSFET Mdrive with drain, gate, and source connected to nodes driver, in and output with a width and length of 3 and 0.25 microns, respectively. This device references MOSFET model bsim3v3 and specifies a temperature for the device that is 4 degrees Celsius above the circuit temperature.

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

Transmission Lines

W Element Statement

The general syntax for including a lossy (W Element) transmission line element in a Star-Hspice netlist is:

RLGC File Form

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

U-model Form

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

Field Solver Form

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

where the number of ports on a single transmission line are not limited. One input and output port, the ground references, a model or file reference, a number of conductors and a length are all required.

The arguments are defined as:

Wxxx	Lossy (W Element) transmission line element name. Must begin with a “W”, which can be followed by up to 1023 alphanumeric characters.
inx	Signal input node for the x^{th} transmission line (in1 is required).
refin	Ground reference for input signal
outx	Signal output node for the x^{th} transmission line (each input port must have a corresponding output port).

refout	Ground reference for output signal.
RLGCfile = fname	File name reference for file containing the RLGC information for the transmission lines (see Chapter 6, “Using Transmission Lines”, in the *True-Hspice Device Models Reference Manual* for syntax).
N	Number of conductors (excluding the reference conductor).
L	Physical length of the transmission line in units of meters.
Umodel = mname	U-model lossy transmission-line model reference name. A lossy transmission line model, used here to represent the characteristics of the W-element transmission line.
FSmodel = mname	Internal field solver model name. References the PETL internal field solver as the source of the transmission-line characteristics (see Chapter 6, “Using Transmission Lines”, in the *True-Hspice Device Models Reference Manual* for syntax).

Example

Lossy transmission line W1 connected from node in to node out with both signal references grounded, using the RLGC file named cable.rlgc and length of 5 meters.

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

Two-conductor lossy transmission line Wcable is connected from nodes in1 and in2 to out1 and out2 with grounds on both signal references. References the U-model named umod_1 and is 10 meters in length.

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

Five-conductor lossy transmission line Wnet1 connected from nodes i1, i2, i3, i4 and i5 to nodes o1, o3, o5 and the second and fourth outputs grounded with

both signal references grounded as well. References the Field Solver model named board1 and is 1 millimeter long.

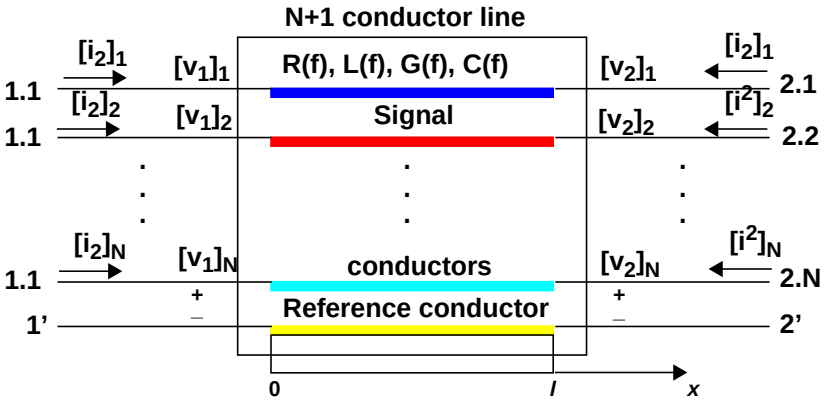
```
Wnet1 i1 i2 i3 i4 i5 gnd o1 gnd o3 gnd o5 gnd
+ FSmodel = board1 N = 5 L = 1m
```

The order of parameters in the W-element card does not matter and the number of signal conductors, *N*, can be specified after the list of nodes. Moreover, the nodes and parameters in the W-element card can be mixed freely.

Only one of the `RLGCfile,FSmodel` or `Umodel` models can be specified in a single W-element card.

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

Figure 4-1: Terminal Node Numbering for W Element



T Element Statement

The general syntax for including a lossless (T Element) transmission line element in a Star-Hspice netlist is:

General Form

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

or

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

U-model Form

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

where only one input and output port is allowed.

The arguments are defined as:

T*xxx*	Lossless transmission line element name. Must begin with a “T”, which can be followed by up to 1023 alphanumeric characters.
in	Signal input node
refin	Ground reference for input signal
out	Signal output node
refout	Ground reference for output signal
Z0	Characteristic impedance of the transmission line
TD	Signal delay from the transmission line in units of seconds per meter
L	Physical length of the transmission line in units of meters. Default = 1.

IC = v1,i1,v2,i2	Initial conditions of the transmission line. Specify the voltage on the input port (v1), current into the input port (i1), voltage on the output port (v2) and the current into the output port (i2).
F	Frequency at which the transmission line has the electrical length given by NL.
NL	Normalized electrical length of the transmission line at the frequency, specified in the F parameter, in units of wavelengths per line length. Default = 0.25, which corresponds to a quarter-wavelength.
mname	U-model reference name. A lossy transmission line model, used here to represent the characteristics of the lossless transmission line.

Example

Transmission line T1 connected from node in to node out with both signal references grounded, with a 50 ohm impedance and a 5 nanosecond per meter transmission delay and a length of 5 meters.

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

Transmission line Tcable is connected from node in1 to out1 with grounds on both signal references, a 100 ohm impedance, and a normalized electrical length of 1 wavelength at 100 kHz.

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

Transmission line Tnet1 connected from node driver to node output with both signal references grounded. References the U-model named Umodel1 and is 1 millimeter long.

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

U Element Statement

The general syntax for including a lossy (U Element) transmission line element in a Star-Hspice netlist is:

General Form

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

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

The arguments are defined as:

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

Example

Lossy transmission line U1 connected from node in to node out with both signal references grounded, using the U-model named umodel_RG58 and length of 5 meters.

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

Two-conductor lossy transmission line Ucable is connected from nodes in1 and in2 to out1 and out2 with grounds on both signal references. References the U-model named twistpr and is 10 meters in length.

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

Five-conductor lossy transmission line Unet1 connected from nodes i1, i2, i3, i4 and i5 to nodes o1, o3, o5 and the second and fourth outputs grounded with both signal references grounded as well. References the U-model named Umodel1 and is 1 millimeter long.

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

Buffers

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

General Form

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

where:

bname	Is the buffer name and starts with the letter *B.*
node_1 node_2 ... node_N	Is a list of input/output buffer external nodes. The number of nodes and their meaning are specific to different buffer types.
keyword_i = value_i	Assigns value *value_i* to the keyword *keyword_i.* Optional keywords are given in square brackets.

See Chapter 7, “Using IBIS Models”, in the *True-Hspice Device Models Reference Manual* for information about the keywords.

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, B1, with the 4 terminals *nd_pc*, *nd_gc*, *nd_in* and *nd_out_of_in.* The IBIS model IBIS_IN is located in the IBIS file named test.ibs. Note that nodes *nd_pc* and *nd_gc* are connected by Star-Hspice to the voltage sources. Therefore, users should not connect these nodes to voltage sources. See Chapter 7, “Using IBIS Models”, in the *True-Hspice Device Models Reference Manual* for more examples.

Avant!

Chapter 5
Using Sources and Stimuli

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

The chapter covers the following topics:

Independent Source Elements

Use independent source element statements to specify DC, AC, transient, and mixed independent voltage and current sources. Some types of analysis use the associated analysis sources. For example, in a DC analysis, if both DC and AC sources are specified in one independent source element statement, the AC source is taken out of the circuit for the DC analysis. If an independent source is specified for an AC, transient, and DC analysis, transient sources are removed for the AC analysis and DC sources are removed after the performance of the operating point. Initial transient value always overrides the DC value.

Source Element Conventions

Voltage sources need not be grounded. Positive current is assumed to flow 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 identify these parameters using any of the following reserved keywords:

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

Independent Source Element

The general syntax for an independent source in a Star-Hspice netlist is:

General Form

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

or

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

The arguments are defined as follows:

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

Example

Voltage source VX has a 5 volt DC bias, and the positive terminal is connected to node 1while the negative terminal is grounded.

```
VX 1 0 5V
```

Voltage source VB has a DC bias specified by the parameter ‘VCC’, and the positive terminal is connected to node 2 while the negative terminal is grounded.

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

Voltage source VH has a 2 volt DC bias, a 1 volt RMS AC bias with 90 degree phase offset, and the positive terminal is connected to node 3 while the negative terminal is connected to node 6.

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

Current source IG has a time-varying response given by the piecewise-linear relationship with 1 milliamp at time=0 and 5 milliamps at 25 milliseconds, and the positive terminal is connected to node 8 while the negative terminal is connected to node 7.

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

Voltage source VCC has a DC bias specified by the parameter ‘VCC’, and a time-varying response given by the piecewise-linear relationship with 0 volts at time=0, ‘VCC’ from 10 to 15 nanoseconds and back to 0 volts at 20 nanoseconds. The positive terminal is connected to node in while the negative terminal is connected to node out. The operating point for this source will be determined without the DC value (that is, it will be 0 volts).

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

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

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

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

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

Voltage source VMEAS has a 0 volt DC bias, and the positive terminal is connected to node 12 while the negative terminal is connected to node 9.

```
VMEAS 12 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. Specify the magnitude and phase of the impulse with the AC keyword.

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

The above two examples specify an AC voltage source with a magnitude of 10 V and a phase of 90 degrees. Specify the frequency sweep range of the AC analysis in the .AC analysis statement. The AC or frequency domain analysis provides the impulse response of the circuit.

Transient Sources

For transient analysis, you can specify the source as a function of time. The functions available are pulse, exponential, damped sinusoidal, single frequency FM, and piecewise linear function.

Mixed Sources

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

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

The above example specifies a DC source of 0.5 V, an AC source of 1 V, and a transient damped sinusoidal source, each of which are connected between nodes 13 and 2. For DC analysis, the program uses zero source value since the sinusoidal source is zero at time zero.

Star-Hspice Independent Source Functions

Star-Hspice provides the following types of independent source functions:

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

PWL also comes in a data driven version. The data driven PWL allows the results of an experiment or a previous simulation to provide one or more input sources for a transient simulation.

The independent sources supplied with Star-Hspice permit the designer to specify a variety of useful analog and digital test vectors for either steady state, time domain, or frequency domain analysis. For example, in the time domain, both current and voltage transient waveforms can be specified as exponential, sinusoidal, piecewise linear, single-sided FM functions, or AM functions.

Pulse Source Function

Star-Hspice has 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 *per* from onset to onset.

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

General Form

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

or

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

The arguments are defined as:

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

Below is a table showing the time-value relationship for a PULSE source:

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

Intermediate points are determined by linear interpolation.

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

Example

The following example shows the pulse source connected between node 3 and node 0. The pulse has an output high voltage of 1 V, an output low voltage of -1 V, a delay of 2 ns, a rise and fall time of 2 ns, a high pulse width of 50 ns, and a period of 100 ns.

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

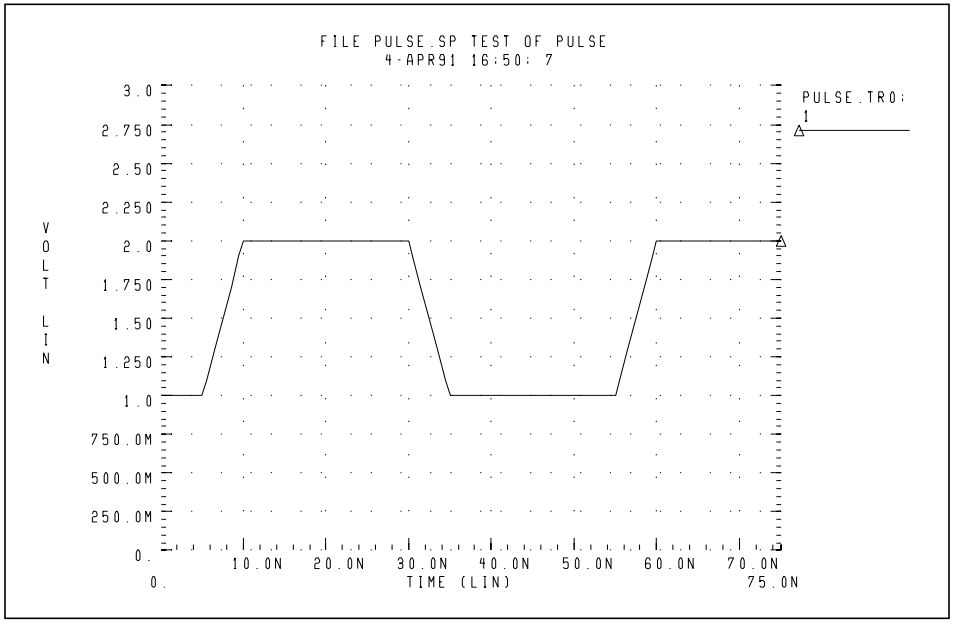
Pulse source connected between node 99 and node 0. The syntax shows parameterized values for all the specifications.

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

This example shows an entire Star-Hspice netlist, which contains a PULSE voltage source. The source has an initial voltage of 1 volt, a pulse voltage of 2 volts, a delay time, rise time and fall time each of 5 nanoseconds, a pulse width of 20 nanoseconds, and a pulse period of 50 nanoseconds. Figure 5-1 shows the result of simulating this netlist.

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

Figure 5-1: Pulse Source Function



Sinusoidal Source Function

Star-Hspice has a damped sinusoidal source that is the product of a dying exponential with a sine wave. Application of this waveform requires the specification of the sine wave frequency, the exponential decay constant, the beginning phase, and the beginning time of the waveform, as explained below.

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

General Form

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

or

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

The arguments are defined as:

Vxxx, Ixxx	Independent voltage source that will exhibit the sinusoidal response.
SIN	Keyword for a sinusoidal time-varying source
vo	Voltage or current offset in volts or amps
va	Voltage or current RMS amplitude in volts or amps
freq	Source frequency in Hz. Default=1/TSTOP.
td	Time delay before beginning the sinusoidal variation in seconds. Default=0.0, response will be 0 volts or amps until the delay value is reached, even with a non-zero DC voltage.
q	Damping factor in units of 1/seconds. Default=0.0.
j	Phase delay in units of degrees. Default=0.0.

The waveform shape is given by the following table of expressions:

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

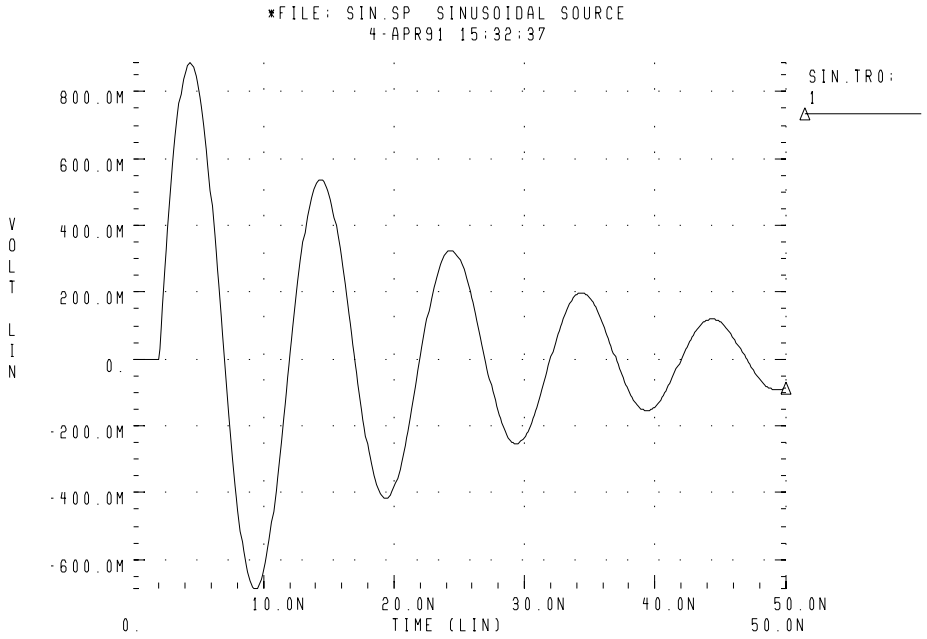
where TSTOP is the final time; see the .TRAN statement for a detailed explanation.

Example

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

Damped sinusoidal source connected between nodes 3 and 0. The waveform has a peak value of 1 V, an offset of 0 V, a 100-MHz frequency, a time delay of 1 ns, a damping factor of 1e10, and a phase delay of zero degree. See Figure 5-2 for a plot of the source output.

Figure 5-2: Sinusoidal Source Function




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

This example shows an entire Star-Hspice netlist that contains a SIN voltage source. The source has an initial voltage of 0 volts, a pulse voltage of 1 volt, a delay time of 2 nanoseconds, a frequency of 100 MHz, and a damping factor of 50 MHz.

Exponential Source Function

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

General Form

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

or

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

The arguments are defined as:

Vxxx, *Ixxx*	Independent voltage source that will exhibit the exponential response.
EXP	Keyword for a exponential time-varying source
v1	Initial value of voltage or current in volts or amps
v2	Pulsed value of voltage or current in volts or amps
td1	Rise delay time in seconds. Default=0.0.
td2	Fall delay time in seconds. Default=td1+TSTEP.
t 1	Rise time constant in seconds. Default=TSTEP.
t 2	Fall time constant in seconds. Default=TSTEP.

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

The waveform shape is given by the following table of expressions:

$$\text{Time} \quad \text{Value}$$

$$0 \text{ to } td1 \quad v1$$

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

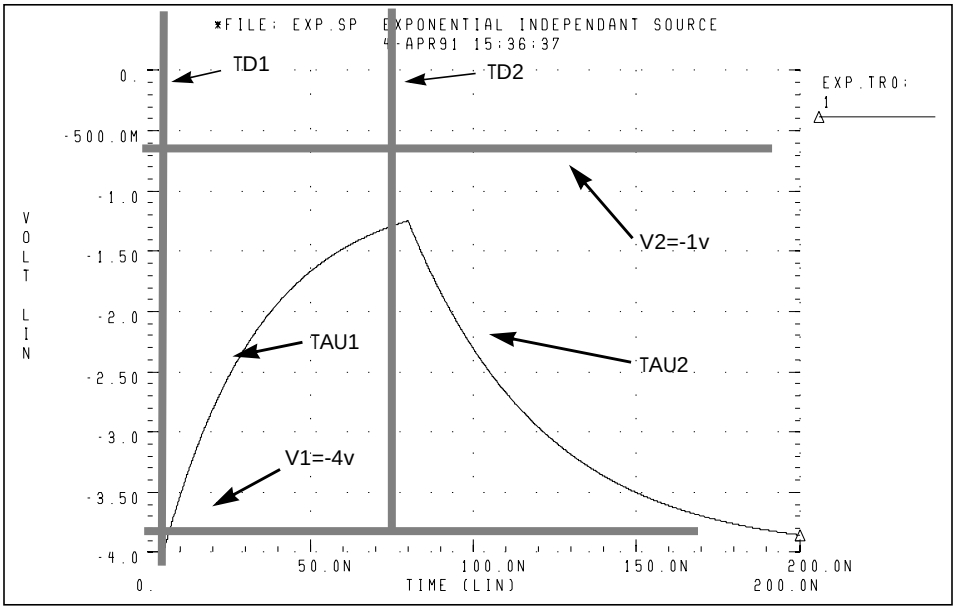
$$td2 \text{ to } tstop \quad v1 + (v2 - v1) \cdot \left[1 - \text{Exp}\left(-\frac{td2 - td1}{\tau_1}\right)\right] \cdot \text{Exp}\left[\frac{-(\text{Time} - td2)}{\tau_2}\right]$$

Example

```
VIN 3 0 EXP (-4 -1 2NS 30NS 60NS 40NS)
```

The above example describes an exponential transient source that is connected between nodes 3 and 0. It has an initial t=0 voltage of -4 V and a final voltage of -1 V. The waveform rises exponentially from -4 V to -1 V with a time constant of 30 ns. At 60 ns it starts dropping to -4 V again, with a time constant of 40 ns.

Figure 5-3: Exponential Source Function




```
*FILE: EXP.SP THE EXPONENTIAL WAVEFORM
.OPTIONS POST
.PARAM V1=-4 V2=-1 TD1=5N TAU1=30N TAU2=40N TD2=80N
V 1 0 EXP (V1 V2 TD1 TAU1 TD2 TAU2)
R 1 0 1
.TRAN .05N 200N
.END
```

This example shows an entire Star-Hspice netlist that contains an EXP voltage source. It has an initial t=0 voltage of -4 V and a final voltage of -1 V. The waveform rises exponentially from -4 V to -1 V with a time constant of 30 ns. At 80 ns it starts dropping to -4 V again, with a time constant of 40 ns.

Piecewise Linear (PWL) Source Function

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

General Form

```
Vxxx n+ n- PWL <(> t1 v1 <t2 v2 t3 v3…> <R <=repeat>>
+ <TD=delay> <)>
```

Or

```
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> <)>
```

Or

```
Ixxx n+ n- PL <(> v1 t1 <v2 t2 v3 t3…> <R <=repeat>>
+ <TD=delay> <)>
```

The arguments are defined as:

Vxxx, Ixxx	Independent voltage source that will exhibit the piecewise linear response.
PWL	Keyword for a piecewise linear time-varying source
v1 v2 … vn	Current or voltage values at 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 time in units of seconds which specifies the start point of the waveform that is to be repeated. This time needs to be less than the greatest time point tn.
TD=delay	Time in units of seconds that specifies the length of time to 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. The value of the source at intermediate values of time is determined by linear interpolation between the time points. ASPEC style formats are accommodated by the “PL” form of the function, which reverses the order of the time-voltage pairs to voltage-time pairs. Star-Hspice uses the DC value of the source as the time-zero source value if no time-zero point is given. Also, Star-Hspice does not force the source to terminate at the TSTOP value specified in the .TRAN statement.

If the slope of the piecewise linear function changes below a certain tolerance, the timestep algorithm may not choose the specified time points as simulation time points, thereby obtaining a value for the source voltage or current by extrapolation of neighboring values. In this situation, you may notice a small deviation of the simulated voltage from that specified in the PWL list. To force Star-Hspice to use the specified values, use the SLOPETOL option to reduce the slope change tolerance (see “Specifying Simulation Output” on page Chapter 81for more information about this option).

Specify “R” to cause the function to repeat. You can specify a value after this “R” to indicate the beginning of the function to be repeated: the repeat time must equal a breakpoint in the function. For example, if t1 = 1, t2 = 2, t3 = 3, and t4 = 4, “repeat” can be equal to 1, 2, or 3.

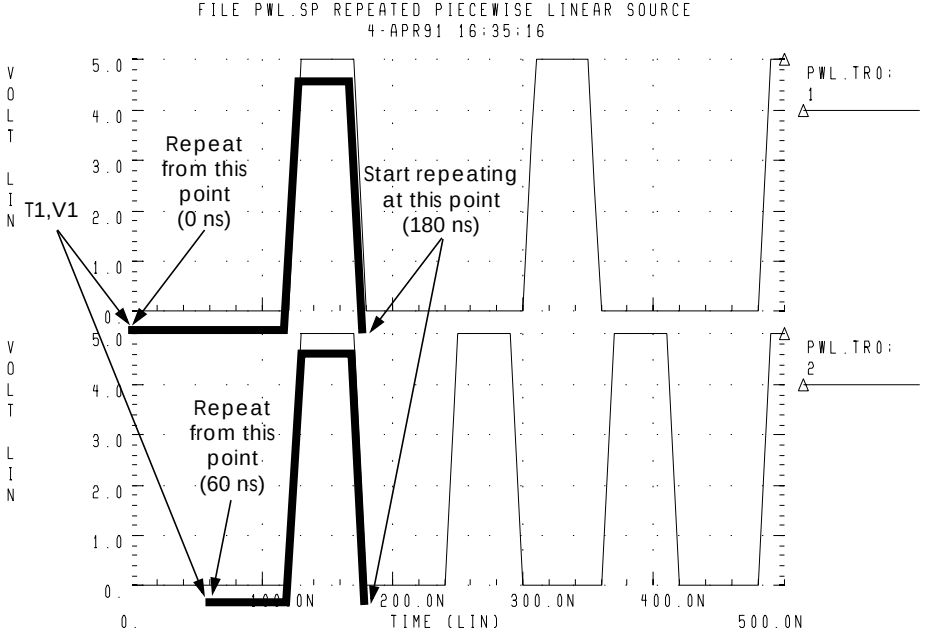
Specify TD=val to cause a delay at the beginning of the function. You can use TD with or without the repeat function.

Example

```
*FILE: PWL.SP THE REPEATED PIECEWISE LINEAR SOURCE
*ILLUSTRATION OF THE USE OF THE REPEAT FUNCTION “R”
*file pwl.sp REPEATED PIECEWISE LINEAR SOURCE
.OPTION POST
.TRAN 5N 500N
V1 1 0 PWL 60N 0V, 120N 0V, 130N 5V, 170N 5V, 180N 0V,
+ R 0N
R1 1 0 1
V2 2 0 PL 0V 60N, 0V 120N, 5V 130N, 5V 170N, 0V 180N,
+ R 60N
R2 2 0 1
.END
```

This example shows an entire Star-Hspice netlist that contains two piecewise linear voltage sources. The two sources have the same function (the first one is in normal format, and the second in ASPEC format). The first source has a repeat specified to start at the beginning of the function, whereas the second repeat starts at the first timepoint. See Figure 5-4 for the difference in responses.

Figure 5-4: Results of Using the Repeat Function



Data Driven Piecewise Linear Source Function

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

General Form

```
Vxxx n+ n- PWL (TIME, PV)
```

or

```
Ixxx n+ n- PWL (TIME, PV)
```

along with:

```
.DATA dataname
TIME PV
t1 v1
t2 v2
t3 v3
t4 v4
.. ..
.ENDDATA
.TRAN DATA=datanam
```

The arguments are defined as:

TIME	Parameter name for time value provided in a .DATA statement.
PV	Parameter name for amplitude value provided in a .DATA statement.

You must use this source with a .DATA statement that contains time-value pairs. For each t*n*-v*n* (time-value) pair given in the .DATA block, the data driven PWL function outputs a current or voltage of the given t*n* duration and with the given v*n* amplitude.

This source allows you to use the results of one simulation as an input source in another simulation. The transient analysis must be data driven.

Example

```
*DATA DRIVEN PIECEWISE LINEAR SOURCE
V1 1 0 PWL(TIME, pv1)
R1 1 0 1
V2 2 0 PWL(TIME, pv2)
R2 2 0 1
.DATA dsrc
TIME pv1 pv2
0n 5v 0v
5n 0v 5v
10n 0v 5v
.ENDDATA
.TRAN DATA=dsrc
.END
```

This example shows an entire Star-Hspice netlist that contains two data-driven piecewise linear voltage sources. The .DATA statement contains the two sets of value data referenced in the sources, *pv1* and *pv2*. The .TRAN statement references the data name.

Single-Frequency FM Source Function

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

General Form

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

or

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

The arguments are as follows:

Vxxx, Ixxx	Independent voltage source that will exhibit the frequency-modulated response.
SFFM	Keyword for a single-frequency, frequency-modulated time-varying source
vo	Output voltage or current offset, in volts or amps

va	Output voltage or current amplitude, in volts or amps.
fc	Carrier frequency in Hz. Default=1/TSTOP.
*md*i	Modulation index that 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 waveform shape is given by the following expression:

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

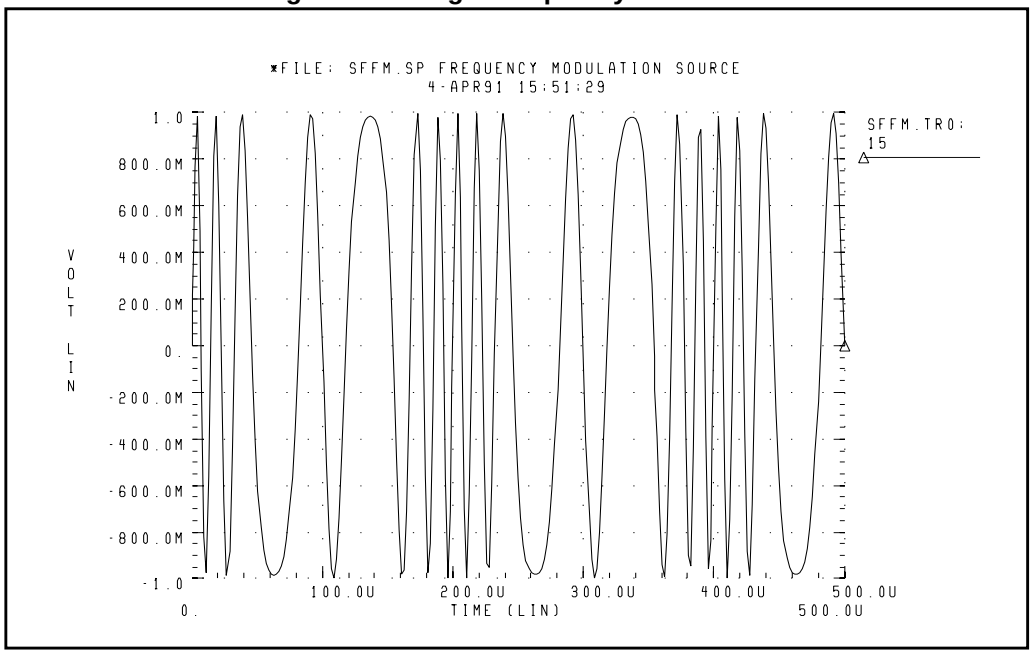
Note: TSTOP is discussed in the .TRAN statement description.

Example

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

This example shows an entire Star-Hspice netlist that contains a single-frequency, frequency-modulated voltage source. The source has an offset voltage of 0 volts, and a maximum voltage of 1 millivolt. The carrier frequency is 20 kHz, and the signal is 5 kHz, with a modulation index of 10 (the maximum wavelength is roughly 10 times longer than the minimum).

Figure 5-5: Single Frequency FM Source



Amplitude Modulation Source Function

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

General Form

```
Vxxx n+ n- AM < (> sa oc fm fc <td> <)>
```

or

```
Ixxx n+ n- AM < (> sa oc fm fc <td> <)>
```

The arguments are as follows:

Vxxx, Ixxx	Independent voltage source that will exhibit 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 before start of signal in seconds. Default=0.0.

The waveform shape is given by the following expression:

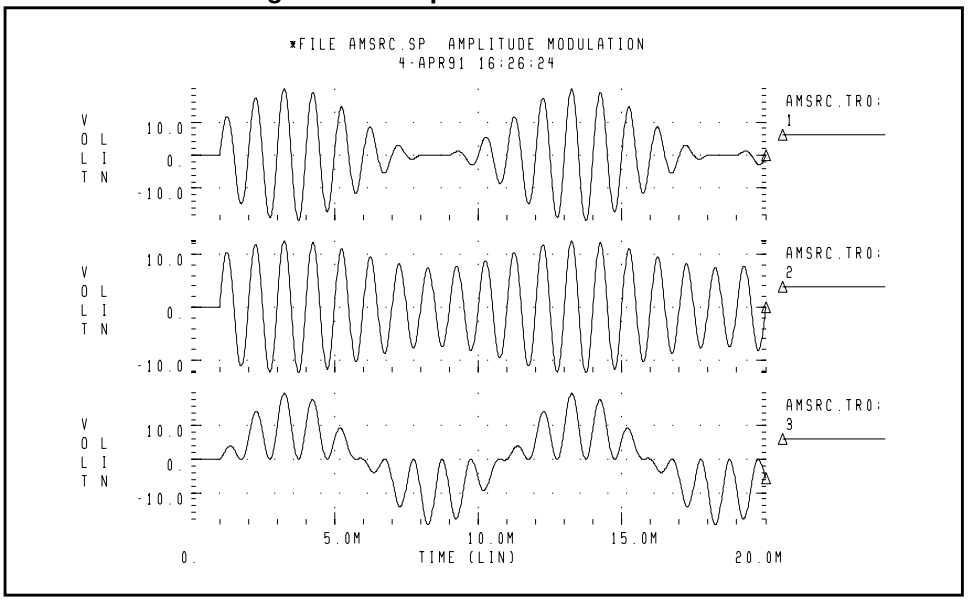
$$\text{sourcevalue} = \text{sa} \cdot \{\text{oc} + \text{SIN}[2 \cdot \pi \cdot \text{fm} \cdot (\text{Time} - \text{td})]\} \cdot \text{SIN}[2 \cdot \pi \cdot \text{fc} \cdot (\text{Time} - \text{td})]$$

Example

```
.OPTION POST
.TRAN .01M 20M
V1 1 0 AM(10 1 100 1K 1M)
R1 1 0 1
V2 2 0 AM(2.5 4 100 1K 1M)
R2 2 0 1
V3 3 0 AM(10 1 1K 100 1M)
R3 3 0 1
.END
```

This example shows an entire Star-Hspice netlist that contains three amplitude-modulated voltage sources. The first has an amplitude of 10, an offset constant of 1, a carrier frequency of 1 kHz, a modulation frequency of 100 Hz, and a delay of 1 millisecond. The second source has the same frequencies and delay, but with an amplitude of 2.5 and an offset constant of 4. The third source is the same as the first but with the carrier and modulation frequencies exchanged.

Figure 5-6: Amplitude Modulation Plot



Using Voltage and Current Controlled Elements

Star-Hspice has four voltage and current controlled elements, known as E, G, H, and F Elements. You can use these controlled elements in Star-Hspice to model both MOS and bipolar transistors, tunnel diodes, SCRs, as well as analog functions such as operational amplifiers, summers, comparators, voltage controlled oscillators, modulators, and switched capacitor circuits. The controlled elements are either linear or nonlinear functions of controlling node voltages or branch currents, depending on whether you use the polynomial or piecewise linear functions. Each controlled element has different functions:

- The E Element is a voltage and/or current controlled voltage source, an ideal op-amp, an ideal transformer, an ideal delay element, or a piecewise linear voltage controlled multi-input AND, NAND, OR, and NOR gate.
- The G Element is a voltage and/or current controlled current source, a voltage controlled resistor, a piecewise linear voltage controlled capacitor, an ideal delay element, or a piecewise linear multi-input AND, NAND, OR, and NOR gate.
- The H Element is a current controlled voltage source, an ideal delay element, or a piecewise linear current controlled multi-input AND, NAND, OR, and NOR gate.
- The F Element is a current controlled current source, an ideal delay element, or a piecewise linear current controlled multi-input AND, NAND, OR, and NOR gate.

The following sections discuss the polynomial and piecewise linear functions and describe element statements for linear or nonlinear functions.

Polynomial Functions

The controlled element statement allows the definition of the controlled output variable (current, resistance, or voltage) as a polynomial function of one or more voltages or branch currents. You can select three polynomial equations through the POLY(NDIM) parameter.

POLY(1)	One-dimensional equation
POLY(2)	Two-dimensional equation
POLY(3)	Three-dimensional equation

The POLY(1) polynomial equation specifies a polynomial equation as a function of one controlling variable, POLY(2) as a function of two controlling variables, and POLY(3) as a function of three controlling variables.

Along with each polynomial equation are polynomial coefficient parameters (P0, P1 … Pn) that can be 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 function value FV is determined by the following expression:

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

FV	Controlled voltage or current from the controlled source
P0. . .PN	Coefficients of polynomial equation
FA	Controlling branch current or nodal voltage

Note: If the polynomial is one-dimensional and exactly one coefficient is specified, Star-Hspice assumes it to be P1 (P0 = 0.0) to facilitate the input of linear controlled sources.

Example

The following controlled source statement is an example of a one-dimensional function:

E1 5 0 POLY(1) 3 2 1 2.5

The above voltage-controlled voltage source is connected to nodes 5 and 0. The single-dimension polynomial function parameter, POLY(1), informs Star-Hspice that E1 is a function of the difference of one nodal voltage pair, in this case, the voltage difference between nodes 3 and 2, hence 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

$$V(5, 0) = 1 + 2.5 \cdot V(3,2)$$

Two-Dimensional Function

Where the function is two-dimensional (a function of two node voltages or two branch currents), FV is determined by the following expression:

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

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.

Example

For example, generate a voltage controlled source that gives the controlled voltage, V(1,0), as:

$$V(1, 0) = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2$$

To implement this function, use the following controlled source element statement:

E1 1 0 POLY(2) 3 2 7 6 0 3 0 0 0 4

This specifies a controlled voltage source connected between nodes 1 and 0 that is controlled by two differential voltages: the voltage difference between nodes 3 and 2 and the 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, and P5=4.

Three-Dimensional Function

For a three-dimensional polynomial function with arguments FA, FB, and FC, the function value FV is determined by the following expression:

$$\begin{aligned} FV = {} & P0 + (P1 \cdot FA) + (P2 \cdot FB) + (P3 \cdot FC) + (P4 \cdot FA^2) \\ & + (P5 \cdot FA \cdot FB) + (P6 \cdot FA \cdot FC) + (P7 \cdot FB^2) + (P8 \cdot FB \cdot FC) \\ & + (P9 \cdot FC^2) + (P10 \cdot FA^3) + (P11 \cdot FA^2 \cdot FB) + (P12 \cdot FA^2 \cdot FC) \\ & + (P13 \cdot FA \cdot FB^2) + (P14 \cdot FA \cdot FB \cdot FC) + (P15 \cdot FA \cdot FC^2) \\ & + (P16 \cdot FB^3) + (P17 \cdot FB^2 \cdot FC) + (P18 \cdot FB \cdot FC^2) \\ & + (P19 \cdot FC^3) + (P20 \cdot FA^4) + \ldots \end{aligned}$$

Example

For example, generate a voltage controlled source that gives the voltage as:

$$V(1, 0) = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2 + 5 \cdot V(9,8)^3$$

from the above defining equation and the three-dimensional polynomial equation:

$$FA = V(3,2)$$

$$FB = V(7,6)$$

$$FC = V(9,8)$$

$$P1 = 3$$

$$P7 = 4$$

$$P19 = 5$$

Substituting these values into the voltage controlled voltage source statement yields the following:

```
V(1, 0) POLY(3) 3 2 7 6 9 8 0 3 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 5
```

The above specifies a controlled voltage source connected between nodes 1 and 0 that is controlled by three differential voltages: the voltage difference between nodes 3 and 2, the voltage difference between nodes 7 and 6, and the voltage difference between nodes 9 and 8, that is, FA=V(3,2), FB=V(7,6), and FC=V(9,8). The statement gives the polynomial coefficients as P1=3, P7=4, P19=5, and the rest are zero.

Piecewise Linear Function

The one-dimensional piecewise linear function allows you to model some special element characteristics, such as those of tunnel diodes, silicon-controlled rectifiers, and diode breakdown regions. The piecewise linear function can be described by specifying measured data points. Although the device characteristic is described by some data points, Star-Hspice automatically smooths the corners to ensure derivative continuity and, as a result, better convergence.

A parameter DELTA is provided to control 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 quite separated, specify the DELTA to a proper value. You can specify up to 100 point pairs. At least two point pairs (four coefficients) must be specified.

In order to model bidirectional switch or transfer gates, the functions NPWL and PPWL are provided for G Elements. The NPWL and PPWL function like NMOS and PMOS transistors.

The piecewise linear function also models multi-input AND, NAND,OR, and NOR gates. In this case, only one input determines the state of the output. In AND / NAND gates, the input with the smallest value is used in the piecewise linear function to determine the corresponding output of the gates. In the OR / NOR gates, the input with the largest value is used to determine the corresponding output of the gates.

Voltage Dependent Voltage Sources — E Elements

E Element syntax statements are described in the following paragraphs. The parameters are defined in the following section.

Voltage Controlled Voltage Source (VCVS)

Syntax

Linear

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

Polynomial

```
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=vals>
```

Piecewise Linear

```
Exxx n+ n- <VCVS> PWL(1) in+ in-    <DELTA=val>
+ <SCALE=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ...
+ x100,y100 <IC=val>
```

Multi-Input Gates

```
Exxx n+ n- <VCVS> gatetype(k) in1+ in1- ... ink+ ink-
+ <DELTA=val> <TC1=val> <TC2=val> <SCALE=val>
+ x1,y1  ...   x100,y100 <IC=val>
```

Delay Element

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

Behavioral Voltage Source

The syntax is:

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

Ideal Op-Amp

The syntax is:

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

Ideal Transformer

The syntax is:

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

Parameter Definitions

ABS	Output is absolute value if ABS=1.
DELAY	Keyword for the delay element. The delay element is the same as voltage controlled voltage source, except it is associated by a propagation delay TD. This element facilitates the adjustment of propagation delay in the macro-modelling process. **Note:** DELAY is a reserved word and should not be used as a node name.
DELTA	Controls the curvature of the piecewise linear corners. The parameter defaults to one-fourth of the smallest breakpoint distances. The maximum is limited to one-half of the smallest breakpoint distances.

Exxx	Voltage controlled element name. The parameter must begin with an E, followed by up to 1023 alphanumeric characters.
gain	Voltage gain
gatetype(k)	Can be one of AND, NAND, OR, or NOR. ■ (k) represents the number of inputs of the gate. ■ The x's and y's represent the piecewise linear variation of output, as a function of input. In the multi-input gates, only one input determines the state of the output.
IC	Initial condition: the initial estimate of the value(s) of the controlling voltage(s). If IC is not specified, the default=0.0.
in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
k	Ideal transformer turn ratio: $V(in+,in-) = k \cdot V(n+,n-)$ or, number of gates input
MAX	Maximum output voltage value. The default is undefined, and sets no maximum value.
MIN	Minimum output voltage value. The default is undefined, and sets no minimum value.
n+/-	Positive or negative node of a controlled element
NDIM	Polynomial dimensions. If POLY(NDIM) is not specified, Star-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 10 or the smaller of TD/tstep and tstop/tstep That is, $\mathrm{NPDELAY}_{\mathrm{default}} = \max\left[\frac{\min\langle \mathrm{TD}, \mathrm{tstop}\rangle}{\mathrm{tstep}}, 10\right]$ The .TRAN statement specifies the values of tstep and tstop.
OPAMP	The keyword for ideal op-amp element. OPAMP is a reserved word; do not use it as a node name.
P0, P1 …	The polynomial coefficients. When you specify one coefficient, Star-Hspice assumes it is P1 (P0=0.0), and the element is linear. When you specify more than one polynomial coefficient, the element is nonlinear, and P0, P1, P2 ... represent them (see Polynomial Functions on page 5-26).
POLY	Polynomial keyword function
PWL	Piecewise linear keyword function
SCALE	Element value multiplier
TC1,TC2	First and second order temperature coefficients. The SCALE is updated by temperature: $\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + \mathrm{TC1} \cdot \Delta t + \mathrm{TC2} \cdot \Delta t^2)$
TD	Time delay keyword
TRANSFORMER	Keyword for ideal transformer. TRANS is a reserved word; do *not* use it as a node name.

VCVS	Keyword for voltage-controlled voltage source. VCVS is a reserved word; do not use it as a node name.
x1,...	Controlling voltage across nodes in+ and in-. The x values must be in increasing order.
y1,...	Corresponding element values of x

Example

Ideal OpAmp

A voltage amplifier with supply limits can be built with the voltage controlled voltage source. The output voltage across nodes 2,3 = v(14,1) * 2. The voltage gain parameter, 2, is also given. The MAX and MIN parameters specify a maximum E1 voltage of 5 V and a minimum E1 voltage output of -5 V. If, for instance, V(14,1) = -4V, E1 would be set to -5 V and not -8 V, as the equation would produce.

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

A user-defined parameter can be used in the following format to specify a value for polynomial coefficient parameters:

```
.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) and V(17,0). It describes a voltage source with the value:

$$V(13,0) + V(15,0) + V(17,0)$$

This example represents an ideal voltage summer. The three controlling voltages are initialized for a DC operating point analysis to 1.5, 2.0, and 17.25 V, respectively.

```
EX 17 0 POLY(3) 13 0 15 0 17 0 0 1 1 1 IC=1.5,2.0,17.25
```

Polynomial Function

The voltage controlled source also can output a nonlinear function using the one-dimensional polynomial. Since the POLY parameter is not specified, a one-dimensional polynomial is assumed—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
```

Zero Delay Inverter Gate

You can build a simple inverter with no delay with a piecewise linear transfer function.

```
Einv out 0 PWL(1) in 0 .7v,5v 1v,0v
```

Ideal Transformer

With the turn ratio 10 to 1, the voltage relationship is V(out)=V(in)/10.

```
Etrans out 0 TRANSFORMER in 0 10
```

Voltage Controlled Oscillator (VCO)

Use the VOL keyword to define a single-ended input that controls the output of a VCO.

In the following example, the frequency of the sinusoidal output voltage at node “out” is controlled by the voltage at node “control”. Parameter “v0” is the DC offset voltage and “gain” is the amplitude. The output is a sinusoidal voltage with a frequency of “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). This equation does not apply if you sweep the control voltage.

Voltage Dependent Current Sources — G Elements

G Element syntax statements are described in the following pages. The parameters are defined in the following section.

Voltage Controlled Current Source (VCCS)

Syntax

Linear

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

Polynomial

```
Gxxx n+ n- <VCCS> POLY(NDIM) in1+ in1- ...
+ <inndim+ inndim-> MAX=val> <MIN=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> <ABS=1> P0<P1…> <IC=vals>
```

Piecewise Linear

```
Gxxx n+ n- <VCCS> PWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>

Gxxx n+ n- <VCCS> NPWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val><TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>

Gxxx n+ n- <VCCS> PPWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
```

Multi-Input Gates

```
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>
```

Delay Element

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

Behavioral Current Source

Syntax

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

Voltage Controlled Resistor (VCR)

Syntax

Linear

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

Polynomial

```
Gxxx n+ n- VCR POLY(NDIM) in1+ in1- ...
+ <inndim+ inndim-> <MAX=val> <MIN=val><SCALE=val>
+ <M=val> <TC1=val> <TC2=val>    P0 <P1…> <IC=vals>
```

Piecewise Linear

```
Gxxx n+ n- VCR PWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>
```

```
Gxxx n+ n- VCR NPWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>
```

```
Gxxx n+ n- VCR PPWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>
```

Multi-Input Gates

```
Gxxx n+ n- VCR gatetype(k) in1+ in1- ... ink+ ink-
+ <DELTA=val> <TC1=val> <TC2=val> <SCALE=val> <M=val>
+ x1,y1 ... x100,y100 <IC=val>
```

Voltage Controlled Capacitor (VCCAP)

Syntax (Piecewise Linear)

```
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>
```

The two functions NPWL and PPWL allow the interchange of the “n+” and “n-” nodes while keeping the same transfer function. This action is summarized as follows:

NPWL Function

For node “in-” 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 node “in-” 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 node “in-” connected to “n-”:

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

For node “in-” connected to “n+”:

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

Parameter Definitions

ABS	Output is absolute value if ABS=1.
CUR, VALUE	Current output that flows from n+ to n-. The equation that you define can be a function of node voltages, branch currents, TIME, temperature (TEMPER), and frequency (HERTZ).
DELAY	Keyword for the delay element. The delay element is the same as voltage controlled current source except it is associated by a propagation delay TD. This element facilitates the adjustment of propagation delay in the macromodel process. **Note:** Because DELAY is a Star-Hspice keyword, it should not be used as a node name.
DELTA	Used to control the curvature of the piecewise linear corners. The parameter defaults to 1/4 of the smallest breakpoint distances. The maximum is limited to 1/2 of the smallest breakpoint distances.
Gxxx	Voltage controlled element name. This parameter must begin with a “G” followed by up to 1023 alphanumeric characters.
gatetype(k)	Can be one of AND, NAND, OR, or NOR. The parameter (k) represents the number of inputs of the gate. The x’s and y’s represent the piecewise linear variation of output as a function of input. In the multi-input gates, only one input determines the state of the output.
IC	Initial condition. The initial estimate of the value(s) of the controlling voltage(s). If IC is not specified, the default=0.0.

in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
M	Number of element in parallel
MAX	Maximum current or resistance value. The default is undefined and sets no maximum value.
MIN	Minimum current or resistance value. The default is undefined and sets no minimum value.
n+/-	Positive or negative node of controlled element
NDIM	Polynomial dimensions. If POLY(NDIM) is not specified, a one-dimensional polynomial is assumed. NDIM must be a positive number.
NPDELAY	Sets the number of data points to be used in delay simulations. The default value is the larger of 10 or the smaller of TD/tstep and tstop/tstep That is, $\mathrm{NPDELAY}_{\mathrm{default}} = \max\left[\frac{\min\langle \mathrm{TD}, \mathrm{tstop}\rangle}{\mathrm{tstep}}, 10\right]$ The values of tstep and tstop are specified in the .TRAN statement.
NPWL	Models the symmetrical bidirectional switch or transfer gate, NMOS
P0, P1 …	The polynomial coefficients. When one coefficient is specified, Star-Hspice assumes it to be P1 (P0=0.0), and the element is linear. When more than one polynomial coefficient is specified, the element is nonlinear, and P0, P1, P2 ... represent them (see Polynomial Functions on page 5-26).

POLY	Polynomial keyword function
PWL	Piecewise linear keyword function
PPWL	Models the symmetrical bidirectional switch or transfer gate, PMOS
SCALE	Element value multiplier
SMOOTH	For piecewise linear dependent source elements, SMOOTH selects the curve smoothing method. A curve smoothing method simulates exact data points you provide. This method can be used to make Star-Hspice simulate specific data points that correspond to measured data or data sheets, for example. Choices for SMOOTH are 1 or 2: 1. Selects the smoothing method used in Hspice releases prior to release H93A. Use this method to maintain compatibility with simulations done using releases older than H93A. 2. Selects the smoothing method that uses data points you provide. This is the default for Hspice releases starting with H93A.
TC1,TC2	First and second order temperature coefficients. The SCALE is updated by temperature: $CALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2$
TD	Time delay keyword
transconductance	Voltage-to-current conversion factor
transfactor	Voltage-to-resistance conversion factor

VCCAP	Keyword for voltage controlled capacitance element. VCCAP is a reserved word and should not be used as a node name.
VCCS	Keyword for voltage controlled current source. VCCS is a reserved word and should not be used as a node name.
VCR	Keyword for voltage controlled resistor element. VCR is a reserved word and should not be used as a node name.
x1,...	Controlling voltage across nodes in+ and in-. The x values must be in increasing order.
y1,...	Corresponding element values of x

Example

Switch

A voltage-controlled resistor represents a basic switch characteristic. The resistance between nodes 2 and 0 varies linearly from 10 meg to 1 m ohms when voltage across nodes 1 and 0 varies between 0 and 1 volt. Beyond the voltage limits, the resistance remains at 10 meg and 1 m ohms, respectively.

```
Gswitch 2 0 VCR PWL(1) 1 0 0v,10meg 1v,1m
```

Switch-Level MOSFET

Model a switch level n-channel MOSFET by the N-piecewise linear resistance switch. The resistance value does not change when the node d and s positions are switched.

```
Gnmos d s VCR NPWL(1) g s LEVEL=1 0.4v,150g
+ 1v,10meg 2v,50k 3v,4k 5v,2k
```

Voltage Controlled Capacitor

The capacitance value across nodes (out,0) varies linearly from 1 p to 5 p when voltage across nodes (ctrl,0) varies between 2 v and 2.5 v. Beyond the voltage limits, the capacitance value remains constant at 1 picofarad and 5 picofarads respectively.

```
Gcap out 0 VCCAP PWL(1) ctrl 0 2v,1p 2.5v,5p
```

Zero Delay Gate

Implement a two-input AND gate using an expression and a piecewise linear table. The inputs are voltages at nodes a and b, and the output is the current flow from node out to 0. The current is multiplied by the SCALE value, which in this example is specified as the inverse of the load resistance connected across the nodes (out,0).

```
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. A transmission line, on the other hand, has an infinite frequency response. A glitch input to a G delay is attenuated similarly to a buffer circuit. In this example, the output of the delay element is the current flow from node *out* to node *1* with a value equal to the voltage across nodes (*in*, *0*) multiplied by SCALE value and delayed by TD value.

```
Gdel out 0 DELAY in 0 TD=5ns SCALE=2 NPDELAY=25
```

Diode Equation

Model forward bias diode characteristic from node 5 to ground with 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

Model a diode breakdown region to forward region using the following example. When voltage across the diode goes beyond 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
```

Triode

Both the following voltage controlled current sources implement a basic triode. The first uses the poly(2) operator to multiply the anode and grid voltages together and scale by .02. The next example uses the explicit behavioral algebraic description.

```
gt i_anode cathode poly(2) anode,cathode
+ grid,cathode 0 0 0 0 .02 gt i_anode cathode
+ cur='20m*v(anode,cathode)*v(grid,cathode)'
```

Current Dependent Voltage Sources — H Elements

H Element syntax statements are described in the following paragraphs. The parameters are defined in the following section.

Current Controlled Voltage Source — (CCVS)

Syntax

Linear

```
Hxxx n+ n- <CCVS> vn1 transresistance <MAX=val>
+ <MIN=val> <SCALE=val> <TC1=val><TC2=val> <ABS=1>
+ <IC=val>
```

Polynomial

```
Hxxx n+ n- <CCVS> POLY(NDIM) vn1 <... vnndim>
+ <MAX=val>MIN=val> <TC1=val> <TC2=val> <SCALE=val>
+ <ABS=1> P0  <P1…> <IC=vals>
```

Piecewise Linear

```
Hxxx n+ n- <CCVS> PWL(1) vn1 <DELTA=val> <SCALE=val>
+ <TC1=val> <TC2=val> x1,y1  ...   x100,y100 <IC=val>
```

Multi-Input Gates

```
Hxxx n+ n- gatetype(k) vn1, ... vnk <DELTA=val>
+ <SCALE=val> <TC1=val> <TC2=val> x1,y1 ...  x100,y100
+ <IC=val>
```

Delay Element

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

Parameter Definitions

ABS	Output is absolute value if ABS=1.
CCVS	Keyword for current controlled voltage source. CCVS is a reserved word and should not be used as a node name.
DELAY	Keyword for the delay element. The delay element is the same as a current controlled voltage source except it is associated by a propagation delay TD. This element facilitates the adjustment of propagation delay in the macromodel process. DELAY is a reserved word and should not be used as a node name.
DELTA	Used to control the curvature of the piecewise linear corners. The parameter defaults to 1/4 of the smallest breakpoint distances. The maximum is limited to 1/2 of the smallest breakpoint distances.
gatetype(k)	Can be one of AND, NAND, OR, NOR. (k) represents the number of inputs of the gate. The x's and y's represent the piecewise linear variation of output as a function of input. In the multi-input gates only one input determines the state of the output.
Hxxx	Current controlled voltage source element name. The parameter must begin with an "H" followed by up to 1023 alphanumeric characters.
IC	Initial condition. This is the initial estimate of the value(s) of the controlling current(s) in amps. If IC is not specified, the default=0.0.
MAX	Maximum voltage value. The default is undefined and sets no maximum value.

MIN	Minimum voltage value. The default is undefined and sets no minimum value.
n+/-	Positive or negative controlled source connecting nodes
NDIM	Polynomial dimensions. If POLY(NDIM) is not specified, a one-dimensional polynomial is assumed. NDIM must be a positive number.
NPDELAY	Sets the number of data points to be used in delay simulations. The default value is the larger of 10 or the smaller of TD/tstep and tstop/tstep That is, $\mathrm{NPDELAY}_{\mathrm{default}} = \max\left[\frac{\min\langle \mathrm{TD}, \mathrm{tstop}\rangle}{\mathrm{tstep}}, 10\right]$ The values of tstep and tstop are specifed in the .TRAN statement.
P0, P1 . . .	When one polynomial coefficient is specified, the source is linear, and the polynomial is assumed to be P1 (P0=0.0). When more than one polynomial coefficient is specified, the source is nonlinear, with the polynomials assumed as P0, P1, P2 …
POLY	Polynomial keyword function
PWL	Piecewise linear keyword function
SCALE	Element value multiplier
TC1,TC2	First and second order temperature coefficients. The SCALE is updated by temperature: $\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + \mathrm{TC1} \cdot \Delta t + \mathrm{TC2} \cdot \Delta t^2)$
TD	Time delay keyword

transresistance	Current to voltage conversion factor
vn1 …	Names of voltage sources through which the controlling current flows. One name must be specified for each dimension.
x1,...	Controlling current through vn1 source. The x values must be in increasing order.
y1,...	Corresponding output voltage values of x

Example

```
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. The defining equation of the CCVS is:

$$HX = 1000 \cdot VCUR$$

The defining equation specifies that the voltage output of HX is 1000 times the value of current flowing through CUR. If the equation produces a value of HX >+10 V, or <-10 V, then the MAX= and MIN= parameters set HX to 10 V or -10 V, respectively. CUR is the independent voltage source through which the controlling current flows. If the controlling current does not flow through an independent voltage source, insert a dummy independent voltage source.

```
.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, and P4=1. The controlling current for flowing through VIN1 is initialized at .5 mA. For VIN2, the initial current is 1.3 mA.

Positive controlling current flows from the positive node, through the source, to the negative node of vnam (linear). The polynomial (nonlinear) specifies the source voltage as a function of the controlling current(s).

Current Dependent Current Sources — F Elements

F Element syntax statements are described in the following paragraphs. The parameter definitions follow.

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

```
Fxxx n+ n- <CCCS> POLY(NDIM) vn1 <... vnndim> <MAX=val>
<MIN=val> <TC1=val> <TC2=val> <SCALE=vals> <M=val>
<ABS=1> P0  <P1…> <IC=vals>
```

Piecewise Linear

```
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>
```

Delay Element

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

Parameter Definitions

ABS	Output is absolute value if ABS=1.
CCCS	Keyword for current controlled current source. Note that CCCS is a reserved word and should not be used as a node name.
DELAY	Keyword for the delay element. The delay element is the same as a current controlled current source except it is associated by a propagation delay TD. This element facilitates the adjustment of propagation delay in the macromodel process. *Note:* DELAY is a reserved word and should not be used as a node name.
DELTA	Used to control the curvature of the piecewise linear corners. The parameter defaults to 1/4 of the smallest breakpoint distances. The maximum is limited to 1/2 of the smallest breakpoint distances.
Fxxx	Current controlled current source element name. The parameter must begin with an “F”, followed by up to 1023 alphanumeric characters.
gain	Current gain
gatetype(k)	Can be one of AND, NAND, OR, or NOR. (k) represents the number of inputs of the gate. The x’s and y’s represent the piecewise linear variation of output as a function of input. In the multi-input gates, only one input determines the state of the output. The above keyword names should not be used as a node name.
IC	Initial condition: the initial estimate of the value(s) of the controlling current(s) in amps. If IC is not specified, the default=0.0.

M	Number of element in parallel
MAX	Maximum output current value. The default is undefined and sets no maximum value.
MIN	Minimum output current value. The default is undefined and sets no minimum value.
n+/-	Positive or negative controlled source connecting nodes
NDIM	Polynomial dimensions. If POLY(NDIM) is not specified, a one-dimensional polynomial is assumed. NDIM must be a positive number.
NPDELAY	Sets the number of data points to be used in delay simulations. The default value is the larger of 10 or the smaller of TD/tstep and tstop/tstep. That is, $\mathrm{NPDELAY}_{\mathrm{default}} = \max\left[\frac{\min\langle \mathrm{TD}, \mathrm{tstop}\rangle}{\mathrm{tstep}}, 10\right]$ The values of tstep and tstop are specifed in the .TRAN statement.
P0, P1 …	When one polynomial coefficient is specified, Star-Hspice assumes it to be P1 (P0=0.0) and the source is linear. When more than one polynomial coefficient is specified, the source is nonlinear, and P0, P1, P2 … represent them.
POLY	Polynomial keyword function
PWL	Piecewise linear keyword function
SCALE	Element value multiplier
TC1,TC2	First and second order temperature coefficients. The SCALE is updated by temperature: $SCALEeff = SCALE \cdot (1 + \mathrm{TC1} \cdot \Delta t + \mathrm{TC2} \cdot \Delta t^2)$

TD	Time delay keyword
vn1 …	Names of voltage sources through which the controlling current flows. One name must be specified for each dimension.
x1,...	Controlling current through vn1 source. The x values must be in increasing order.
y1,...	Corresponding output current values of x

Example

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

The above example describes a current controlled current source connected between nodes 13 and 5. The current that controls the value of the controlled source flows through the voltage source named VSENS (to use a current controlled current source, a dummy independent voltage source is often placed into the path of the controlling current). The defining equation is:

$$I(F1) = 5 \cdot I(VSENS)$$

The current gain is 5, the maximum current flow through F1 is 3 A, and the minimum current flow is -3 A. If I(VSENS) = 2 A, I(F1) would be set to 3 amps and not 10 amps as would be suggested by the equation. A user-defined parameter can be specified for the polynomial coefficient(s), as shown below.

```
.PARAM VU = 5
F1 13 5 VSENS MAX=+3 MIN=-3 VU
```

The next example describes a current controlled current source with the value:

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

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

Current flow is from the positive node through the source to the negative node. The direction of positive controlling current flow is from the positive node through the source to the negative node of vnam (linear), or to the negative node of each voltage source (nonlinear).

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

The above example is a delayed current controlled current source.

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

The final example is a piecewise, linear current-controlled current source.

Digital and Mixed Mode Stimuli

Star-Hspice input netlists support two types of digital stimuli: U Element digital input files, and vector input files. This section describes both types.

U Element Digital Input Elements and Models

The U Element can reference digital input and digital output models for mixed mode simulation. Viewlogic's Viewsim mixed mode simulator uses Star-Hspice with digital input from Viewsim. The state information comes from a digital file if Star-Hspice is being run in standalone mode. Digital outputs are handled in a similar fashion. In digital input file mode, the input file is <*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. This is needed because the first line of a digital file, which contains the signal name list, is often longer than the maximum line length accepted by some text editors.

A digital D2A file must not have a blank first line. If the first line of a digital file is blank, Star-Hspice issues an error message.

The following example demonstrates the use of the "\" line continuation character to format an input file for text editing. The 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
```

The general syntax for a U Element digital source in a Star-Hspice netlist is:

General Form

```
Uxxx interface nlo nhi mname SIGNAME = sname IS = val
```

The arguments are defined as

Uxxx	Digital input element name. Must begin with a “U”, which can be followed by up to 1023 alphanumeric characters.
interface	Interface node in the circuit to which the digital input is attached.
nlo	Node connected to low level reference
nhi	Node connected to high level reference
mname	Digital input model reference (U model)
SIGNAME=sname	Signal name as referenced in the digital output file header, can be a string of up to eight alphanumeric characters.
IS=val	Initial state of the input element, must be a state defined in the model.

Model Syntax

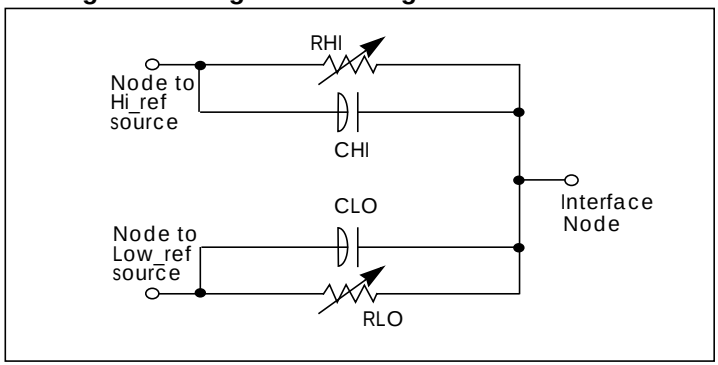
```
.MODEL mname U LEVEL=5 <parameters...>
```

Digital-to-Analog Input Model 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, can be 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, can be a string of up to four alphanumerical characters.
S1TSW	sec		State “1” switching time
S1RLO	ohm		State “1”resistace to low level node
S1RHI	ohm		State “1” resistance to high level node
S19NAME			State “19” character abbreviation, can be 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		Digital input file step size (digital files only)

Up to 20 different states may be defined in the model definition by the SnNAME, SnTSW, SnRLO and SnRHI parameters, where n ranges from 0 to 19. Figure 5-7 shows the circuit representation of the element.

Figure 5-7: Digital-to-Analog Converter Element



Example

The following example shows the usage of the U Element and model as a digital input for a Star-Hspice netlist.

```
* EXAMPLE OF U-ELEMENT DIGITAL INPUT
UC carry-in VLD2A VHD2A D2A SIGNAME=1 IS=0
VLO VLD2A GND DC 0
VHI VHD2A GND DC 1
.MODEL D2A U LEVEL=5 TIMESTEP=1NS,
+ S0NAME=0 S0TSW=1NS S0RLO = 15, S0RHI = 10K,
+ S2NAME=x S2TSW=3NS S2RLO = 1K, S2RHI = 1K
+ S3NAME=z S3TSW=5NS S3RLO = 1MEG,S3RHI = 1MEG
+ S4NAME=1 S4TSW=1NS S4RLO = 10K, S4RHI = 60
.PRINT V(carry-in)
.TRAN 1N 100N
.END
```

where 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
```

Digital Outputs

The general syntax for including a digital output in a Star-Hspice output is:

General Form

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

Uxxx	Digital output element name. Must begin with a “U”, which can be followed by up to 1023 alphanumeric characters.
interface	Interface node in the circuit at which the digital output is measured
reference	Node used as a reference for the output
mname	Digital output model reference (U model)
SIGNAME= sname	Signal name as referenced in the digital output file header, can be a string of up to eight alphanumeric characters.

Model Syntax

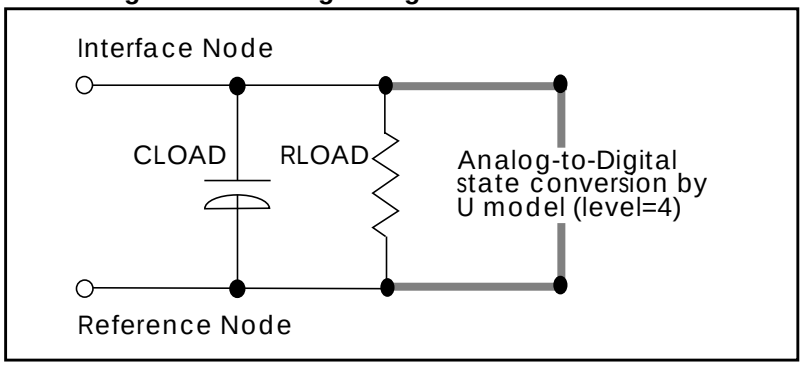
```
.MODEL mname U LEVEL=4 <parameters...>
```

Analog-to-Digital Output Model Parameters

Name (Alias)	Units	Default	Description
RLOAD	ohm	1/gmin	Output resistance.
CLOAD	farad	0	Output capacitance.
S0NAME			State “0” character abbreviation, can be 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, can be a string of up to four alphanumerical characters.
S1VLO	volt		State “1” low level voltage.
S1VHI	volt		State “1” high level voltage.
S19NAME			State “19” character abbreviation, can be 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	Digital input file step size.
TIMESCALE			Scale factor for time.

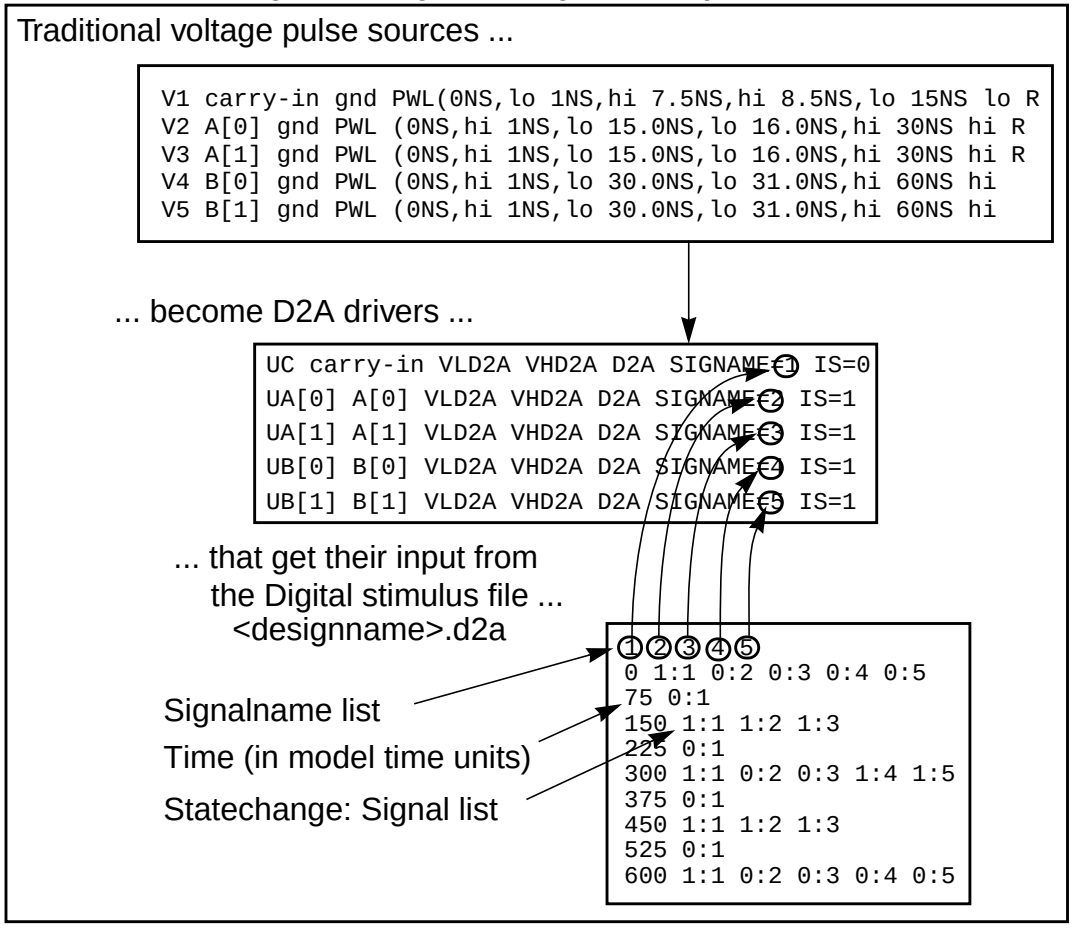
You can define up to 20 different states in the model definition, using the SnNAME, SnVLO and SnVHI parameters, where n ranges from 0 to 19. Figure 5-8 shows the circuit representation of the element.

Figure 5-8: Analog-to-Digital Converter Element



Replacing Sources With Digital Inputs

Figure 5-9: Digital File Signal Correspondence



Example

```
* EXAMPLE OF U-ELEMENT DIGITAL OUTPUT

VOUT carry_out GND PWL 0N 0V 10N 0V 11N 5V 19N 5V 20N 0V
+ 30N 0V 31N 5V 39N 5V 40N 0V

VREF REF GND DC 0.0V

UCO carry-out REF A2D SIGNAME=12
* DEFAULT DIGITAL OUTPUT MODEL (no “X” value)

.MODEL A2D U LEVEL=4 TIMESTEP=0.1NS TIMESCALE=1
+ S0NAME=0 S0VLO=-1 S0VHI= 2.7
+ S4NAME=1 S4VLO= 1.4 S4VHI=9.0
+ CLOAD=0.05pf

.TRAN 1N 50N

.END
```

and the digital output file should look like:

```
12
0       0:1
105     1:1
197     0:1
305     1:1
397     0:1
```

where the “12” represents the signal name, the first column is the time in units of 0.1 nanoseconds, and the second column has the signal value:name pairs. Subsequent outputs would be represented in the same file by more columns.

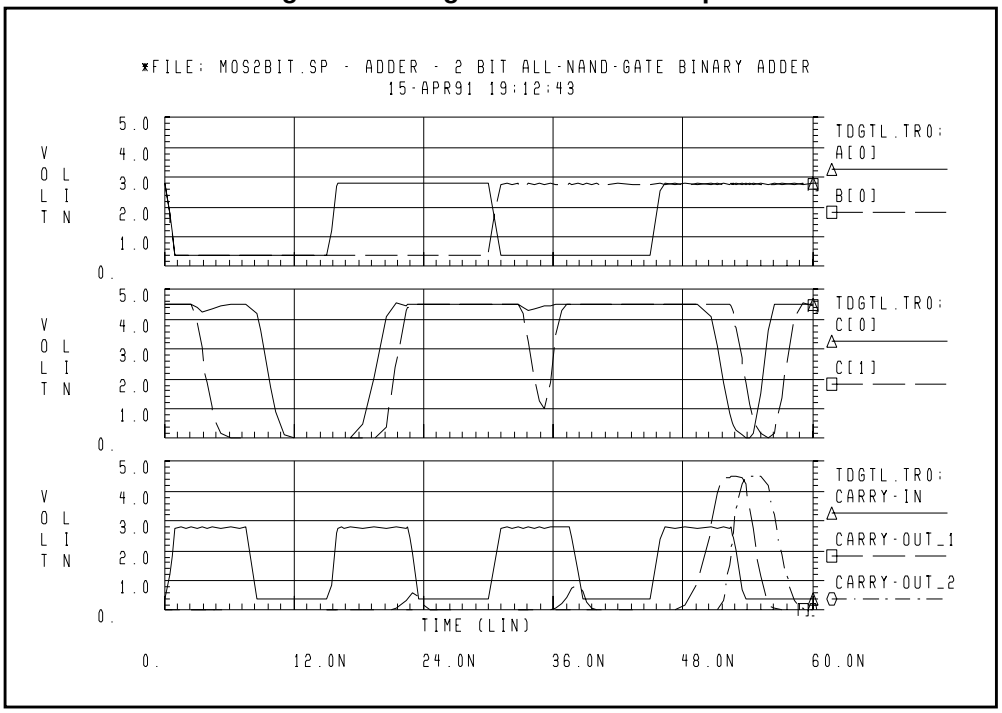
The following two-bit MOS adder uses the digital input file. In the following plot, nodes ‘A[0], A[1], B[0], B[1], and CARRY-IN’ all come from a digital file input (see Figure 5-9). SPICE outputs a digital file.

```
FILE: MOS2BIT.SP - ADDER - 2 BIT ALL-NAND-GATE
+ BINARY ADDER
*

.OPTIONS ACCT NOMOD FAST scale=1u gmindc=100n post
.param lmin=1.25 hi=2.8v lo=.4v vdd=4.5
.global vdd
*

.TRAN .5NS 60NS
```

```
.MEAS PROP-DELAY TRIG V(carry-in) TD=10NS VAL='vdd*.5'
+ RISE=1 TARG V(c[1]) TD=10NS VAL='vdd*.5' RISE=3
*

.MEAS PULSE-WIDTH TRIG V(carry-out_1) VAL='vdd*.5'
+ RISE=1 TARG V(carry-out_1) VAL='vdd*.5' FALL=1
*

.MEAS FALL-TIME TRIG V(c[1]) TD=32NS VAL='vdd*.9'
+ FALL=1 TARG V(c[1]) TD=32NS VAL='vdd*.1' FALL=1
*

VDD vdd gnd DC vdd
X1 A[0] B[0] carry-in C[0] carry-out_1 ONEBIT
X2 A[1] B[1] carry-out_1 C[1] carry-out_2 ONEBIT
*

* Subcircuit Definitions
.subckt NAND in1 in2 out wp=10 wn=5
     M1 out in1 vdd vdd P W=wp L=lmin ad=0
     M2 out in2 vdd vdd P W=wp L=lmin ad=0
     M3 out in1 mid gnd N W=wn L=lmin as=0
     M4 mid in2 gnd gnd N W=wn L=lmin ad=0
     CLOAD out gnd 'wp*5.7f'
.ends
*

.subckt ONEBIT in1 in2 carry-in out carry-out
     X1 in1 in2 #1_nand NAND
     X2 in1 #1_nand 8 NAND
     X3 in2 #1_nand 9 NAND
     X4 8 9 10 NAND
     X5 carry-in 10 half1 NAND
     X6 carry-in half1 half2 NAND
     X7 10 half1 13 NAND
     X8 half2 13 out NAND
     X9 half1 #1_nand carry-out NAND
.ENDS ONEBIT
*

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

```
uc0 c[0] vrefa2d a2d signame=10
uc1 c[1] vrefa2d a2d signame=11
uco carry-out_2 vrefa2d a2d signame=12
uci carry-in vrefa2d a2d signame=13
*

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

.MODEL P PMOS LEVEL=3 VTO=-0.8 UO=150 KAPPA=.25 KP=15U
+ ETA=.015 THETA=.04 VMAX=5E4 NSUB=1.8E16 TOX=400
+ GAMMA=.672 PB=0.6 JS=.1M XJ=0.5U LD=0.15U NFS=1E11
+ NSS=2E10 RSH=80 CJ=.3M MJ=0.5 CJSW=.1N MJSW=0.3
+ acm=2 capop=4
*

* Default Digital Input Interface Model
.MODEL D2A U LEVEL=5 TIMESTEP=0.1NS,
+ S0NAME=0 S0TSW=1NS S0RLO = 15, S0RHI = 10K,
+ S2NAME=x S2TSW=5NS S2RLO = 1K, S2RHI = 1K
+ S3NAME=z S3TSW=5NS S3RLO = 1MEG,S3RHI = 1MEG
+ S4NAME=1 S4TSW=1NS S4RLO = 10K, S4RHI = 60
VLD2A VLD2A 0 DC lo
VHD2A VHD2A 0 DC hi
*

* Default Digital Output Model (no “X” value)
.MODEL A2D U LEVEL=4 TIMESTEP=0.1NS TIMESCALE=1
+ S0NAME=0 S0VLO=-1 S0VHI= 2.7
+ S4NAME=1 S4VLO= 1.4 S4VHI=6.0
+ CLOAD=0.05pf

VREFA2D VREFA2D 0 DC 0.0V

.END
```

Figure 5-10: Digital Stimulus File Input



Specifying a Digital Vector File

The digital vector file consists of three parts:

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

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

```
.VEC 'digital_vector_file'
```

Defining Vector Patterns

The *Vector Pattern Definition* section defines the vectors—their names, sizes, signal direction, and so on—and must occur first in the digital vector file. A sample Vector Pattern Definition section follows:

```
radix 1111 1111
vname a b c d e f g h
io iiii iiii
tunit ns
```

Keywords such as *radix*, *vname* are explained in Defining Tabular Data on page 5-68.

Defining Waveform Characteristics

The *Waveform Characteristics* section defines various attributes for signals, such as the rise or fall time, thresholds for logic ‘high’ or ‘low’, and so on. A sample Waveform Characteristics section follows:

```
trise 0.3 137F 0000
tfall 0.5 137F 0000
vih 5.0 137F 0000
vil 0.0 137F 0000
```

Using Tabular Data

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

Example

An example of tabular data follows:

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

Comment Lines

A line beginning with a semi-colon “;” is considered a comment line. Comments may also start at any point along a line. Star-Hspice ignores characters following a semi-colon.

Example

An example of usage follows:

```
; This is a comment line
radix 1 1 4 1234 ; This is a radix line
```

Continuing a Line

Like netlists, a line beginning with a plus sign “+” is a continuation from the previous line.

Digital Vector File Example

An example of a vector pattern definition follows:

```
; specifies # of bits associated with each vector
radix 1 2 444
;*************************************************
; defines name for each vector. For multi-bit
; vectors, innermost [] provide the bit index range,
; MSB:LSB
vname v1 va[[1:0]] vb[12:1]
;actual signal names: v1, va[0], va[1], vb1 ... vb12
;*************************************************
; defines vector as input, output, or bi-direc
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 delay of 1 ns
; Note: do not put unit (e.g., ns) again here because
; this value will be multiplied by the unit specified
; in the ‘tunit’ line.
tdelay 1.0
; signals va1 and va0 have delays of 1.5ns
tdelay 1.5 0 3 000
;*************************************************
; specify input rise and fall times (if you want
; different rise and fall times, use trise/
```

```
; tfallstmt.)
; Note: do not put unit (e.g., ns) again here because
; this value will be multiplied by the unit specified
; in the 'tunit' line.
slope 1.2
;*****************************************************
; specify the logic 'high' voltage for input signals
vih 3.3 1 0 000
vih 5.0 0 0 FFF
; likewise, may specify logic 'low' with '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
.
.
.
```

Defining Tabular Data

Although this section generally appears last in a digital vector file, following 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. 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...
.
.
```

The set of values for a particular signal over all times is a *vector*, a vertical column in the tabular data and vector table. Thus, the set of all *signal1_valuex* constitute one vector. Signal values may have the legal states described in the following section.

Rows in the tabular data section must appear in chronological order because row placement carries sequential timing information.

Example

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

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

Input Stimuli

Star-Hspice converts each input signal into a PWL (piecewise linear) voltage source and a series resistance. The legal states for an input signal are:

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

For the 0, 1, X, x, U, u states, the resistance value is set to zero; for the L, H states, the resistance value is defined by the *out* (or *outz*) statement; and for the Z, z states, the resistance value is defined by the *triz* statement.

Expected Output

Star-Hspice converts each output signal into a *.DOUT* statement in the netlist. During simulation, Star-Hspice compares the actual results with the expected output vector(s), and if the states are different, an error message appears. The legal states for expected outputs include:

0	Expect ZERO
1	Expect ONE
X, x	Don't care
U, u	Don't care
Z, z	Expect HIGH IMPEDANCE (don't care) Z, z are treated as "don't care" because Star-Hspice cannot detect a high impedance state.

Example

An example of usage follows:

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

Verilog Value Format

Star-Hspice also accepts Verilog *sized* format for number specification:

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

The <*size*> specifies (in decimal) the number of bits, and <*base format*> indicates binary ('b or 'B), octal (*'o* or *'O*), or hexadecimal (*'h* or *'H*). Valid <*number*> fields are combinations of the characters *0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F*. Depending on the <*base format*> chosen, only a subset of these characters may be legal.

You may 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, and one bit in the binary base.

If the most significant bit of a number is 0, X, or Z, the number is automatically extended (if necessary) to fill the remaining bits with (respectively) 0, X, or Z. If the most significant bit is 1, it is extended with 0.

Example

```
4'b1111
12'hABx
32'bZ
8'h1
```

This example specifies values for: a 4-bit signal in binary, a 12-bit signal in hexadecimal, a 32-bit signal in binary, and an 8-bit signal in hexadecimal.

Equivalents of these lines in non-Verilog format are:

```
1111
AB xxxx
ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ
1000 0000
```

Periodic Tabular Data

Very often tabular data is periodic, so it is unnecessary to specify the absolute time at every time point. When a user specifies the *period* statement, the *tabular data* section omits the absolute times (see Using Tabular Data on page 5-66 for details).

Example

```
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
```

Defining Vector Patterns

The *Vector Pattern Definition* section defines the sequence or order for each vector stimulus, as well as any individual characteristics. The statements in this section (except the *radix* statement) might appear in any order, and all keywords are case-insensitive.

Radix Statement

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

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

Only one *radix* statement must appear in the file, and it must be the first noncomment line.

Example

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

```
; start of vector pattern definition section
radix 1 1 4 1234 1111 1111
```

Vname Statement

The *vname* statement defines the name of each vector. If not specified, a default name will be given to each signal: V1, V2, V3, and so on. If you define more than one *vname* statement, the last one overrules the previous one.

```
radix 1 1 1 1 1 1 1 1 1 1 1 1
vname V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12
```

Provide the range of the bit indices with a square bracket [] and a colon syntax:

```
[starting_index : ending_index]
```

The *vname* name is required for each bit, and a single name may be associated with multiple bits (*such as* bus notation).

The bit order is MSB:LSB. This bus notation syntax may also be nested inside other grouping symbols such as <>, (), [], etc. The name of each bit will be *vname* with the index suffix appended.

Example 1

If you specify:

```
radix 2 4
vname VA[0:1] VB[4:1]
```

the resulting names of the voltage sources generated are:

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

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

Example 2

If you specify:

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

the resulting names of the voltage sources are:

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

Example 3

This example shows how to specify a single bit of a bus:

```
vname VA[[2:2]]
```

Example 4

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

```
radix 444444
vname A[0:23]
```

IO Statement

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

i	Input used to stimulate the circuit.
o	Expected output used to compare with the simulated outputs.
b	Star-Hspice ignores.

Example

If the *io* statement is not specified, all signals are assumed input signals. If you define more than one *io* statements, the last one overrules previous ones.

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

Tunit Statement

The *tunit* statement defines the time unit in the digital vector file for *period*, *tdelay*, *slope*, *trise*, *tfall*, and *absolute time*. It must be:

fs	femto-second
ps	pico-second
ns	nano-second
us	micro-second
ms	milli-second

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

Example

The *tunit* statement in this example specifies that the absolute times in the *tabular data* section are 11.0ns, 20.0ns, and 33.0ns, respectively.

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

Period and Tskip Statements

The *period* statement defines the time interval for the *tabular data* section so that specifying the absolute time at every time point is not necessary. Thus, if a *period* statement is provided alone (without the *tskip* statement), the *tabular data* section contains only signal values, not absolute times. The time unit of *period* is defined by the *tunit* statement.

Example

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

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

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

Example

If your netlist contains:

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

then Star-Hspice ignores the absolute times 11.0, 20.0 and 33.0.

Enable Statement

The *enable* statement specifies the controlling signal(s) of bidirectional signals and is absolutely required for all bidirectional signals. If more than one *enable* statement exists, the last value will overrule the previous ones, and a warning message will be issued.

The syntax is a keyword *enable*, followed by the controlling signal name and the mask that defines the (bidirectional) signals to which *enable* applies.

The controlling signal of bi-directional signals must be an input signal with radix of 1. The bidirectional signals become output when the controlling signal is at state 1 (or high). If you wish to reverse this default control logic, you must start the control signal name with '~'.

Example

In this example, signals *x* and *y* are bidirectional, as defined by the 'b' in the *io* line. The first enable statement indicates that *x* (as defined by the position of 'F') becomes output when signal *a* is 1. The second enable specifies that bidirectional bus *y* becomes output when signal *a* is 0.

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

Modifying Waveform Characteristics

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

Tdelay, Idelay, and Odelay Statements

The *tdelay*, *idelay* and *odelay* statements define the delay time of the signal relative to the absolute time of each row in the *tabular data* section; *idelay* applies to the input signals, *odelay* applies to the output signals, while *tdelay* applies to both input and output signals.

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

The time unit of *tdelay*, *idelay* and *odelay* is defined by the *tunit* statement. Normally, you only need to use the *tdelay* statement; only use the *idelay* and *odelay* statements to specify different input and output delay times for bi-directional signals. *idelay* settings on output signals (or *odelay* settings on input signals) are ignored with warning message issued.

More than one *tdelay* (*idelay*, *odelay*) statement can be specified. If more than one *tdelay* (*idelay*, *odelay*) statement is applied to a signal, the last value will overrule the previous ones, and a warning will be given. If you do not specify the signal delays by a *tdelay* (*idelay* or *odelay*) statement, Star-Hspice defaults to zero.

Example

The first *tdelay* statement indicates that all signals have the same delay time 1.0. The delay time of some signals are overruled by the subsequent *tdelay* statements. The V2 and Vx signals have delay time -1.2, and V4 V5[0:1] V6[0:2] have a delay of 1.5. The V7[0:3] signals have an input delay time of 2.0 and an output delay time of 3.0.

```
radix 1 1 4 1234 11111111
io i i o iiib iiiiiiii
vname V1 V2 VX[3:0] V4 V5[1:0] V6[0:2] V7[0:3]
+V8 V9 V10 V11 V12 V13 V14 V15
tdelay 1.0
tdelay -1.2 0 1 1 0000 00000000
tdelay 1.5 0 0 0 1370 00000000
idelay 2.0 0 0 0 000F 00000000
odelay 3.0 0 0 0 000F 00000000
```

Slope Statement

The *slope* statement specifies input signal rise/fall time, with the time unit defined by the *tunit* statement. You can specify the signals to which the *slope* applies using a mask. If the *slope* statement is not provided, the default slope value is 0.1 ns.

If you specify more than one *slope* statement, the last value will overrule the previous ones, and a warning message will be issued. The *slope* statement has no effect on the expected output signals. The rising time and falling time of a signal will be overruled if *trise* and *tfall* are specified.

Example

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

```
slope 1.2
slope 1.1 1100 0110
```

Trise Statement

The *trise* statement specifies the rise time of each input signal (for which the mask applies). The time unit of *trise* is defined by the *tunit* statement.

Example

If you do not specify the rising time of the signals by any *trise* statement, the value defined by the *slope* statement is used. If you apply more than one *trise* statements to a signal, the last value overrules the previous ones, and Star-Hspice issues a warning message.

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

The *trise* statements have no effect on the expected output signals.

Tfall Statement

The *tfall* statement specifies the falling time of each input signal (for which the mask applies). The time unit of *tfall* is defined by the *tunit* statement.

Example

If you do not specify the falling time of the signals by a *tfall* statement, Star-Hspice uses the value defined by the *slope* statement. If you specify more than one *tfall* statement to a signal, the last value overrules the previous ones, and Star-Hspice issues a warning message.

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

The *tfall* statements have no effect on the expected output signals.

Out /Outz Statements

The keywords *out* and *outz* are equivalent and specify the output resistance of each signal (for which the mask applies); *out* (or *outz*) applies to the input signals only.

Example

If you do not specify the output resistance of a signal by an *out* (or *outz*) statement, Star-Hspice uses the default (zero). If you specify more than one *out* (or *outz*) statement to a signal, Star-Hspice overrules the last value with the previous ones, and issues a warning message.

```
out 15.1
out 150 1 1 1 0000 00000000
outz 50.5 0 0 0 137F 00000000
```

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

Triz Statement

The *triz* statement specifies the output impedance when the signal (for which the mask applies) is in *tristate*; *triz* applies to the input signals only.

Example

If you do not specify the *tristate* impedance of a signal by a *triz* statement, Star-Hspice assumes 1000M. If you apply more than one *triz* statement to a signal, the last value overrules the previous ones, and Star-Hspice issues a warning.

```
triz 15.1M
triz 150M 1 1 1 0000 00000000
triz 50.5M 0 0 0 137F 00000000
```

The *triz* statements have no effect on the expected output signals.

Vih Statement

The *vih* statement specifies the logic high voltage of each input signal to which the mask applies.

Example

If you specify the logic high voltage of the signals by a *vih* statement, Star-Hspice assumes 3.3. If you apply more than one *vih* statements to a signal, the last value overrules the previous ones, and Star-Hspice issues a warning.

```
vih 5.0
vih 5.0 1 1 1 137F 00000000
vih 3.5 0 0 0 0000 11111111
```

The *vih* statements have no effect on the expected output signals.

Vil Statement

The *vil* statement specifies the logic low voltage of each input signal to which the mask applies.

Example

If you specify the logic low voltage of the signals by a *vil* statement, Star-Hspice assumes 0.0. If you apply more than one *vil* statement to a signal, the last value overrules the previous ones, and Star-Hspice issues a warning.

```
vil 0.0
vil 0.0 1 1 1 137F 11111111
```

The *vil* statements have no effect on the expected output signals.

Vref Statement

Similar to the *tdelay* statement, the *vref* statement specifies the name of the reference voltage for each input vector to which the mask applies; *vref* applies to the input signals only.

Example

If your netlist contains:

```
vname v1 v2 v3 v4 v5[1:0] v6[2:0] v7[0:3] v8 v9 v10
vref 0
vref 0 111 137F 000
vref vss 0 0 0 0000 111
```

When Star-Hspice implements it into the netlist, the voltage source realizes *v1*:

```
v1 V1 0 pwl(......)
```

as do *v2*, *v3*, *v4*, *v5*, *v6*, and *v7*. However, *v8* is realized by

```
V8 V8 vss pwl(......)
```

as is *v9* and *v10*.

If you do not specify the reference voltage name of the signals by a *vref* statement, Star-Hspice assumes 0. If you apply more than one *vref* statement, the last value will overrule the previous ones, and Star-Hspice issues a warning. The *vref* statements have no effect on the output signals.

Vth Statement

Similar to the *tdelay* statement, the *vth* statement specifies the logic threshold voltage of each signals to which the mask applies; *vth* applies to the output signals only. The threshold voltage is used to decide the logic state of Star-Hspice's output signals for comparison with the expected output signals.

Example

If you do not specify the threshold voltage of the signals by a *vth* statement, Star-Hspice assumes 1.65. If you apply more than one *vth* statements to a signal, the last value overrules the previous ones, and Star-Hspice issues a warning.

```
vth 1.75
vth 2.5 1 1 1 137F 00000000
vth 1.75 0 0 0 0000 11111111
```

The *vth* statements have no effect on the input signals.

Voh Statement

The *voh* statement specifies the logic high voltage of each output signal to which the mask applies.

Example

If you do not specify the logic high voltage by a *voh* statement, Star-Hspice assumes 3.3. If you apply more than one *voh* statements to a signal, the last value overrules the previous ones, and Star-Hspice issues a warning.

```
voh 4.75
voh 4.5 1 1 1 137F 00000000
voh 3.5 0 0 0 0000 11111111
```

The *voh* statements have no effect on input signals.

Note: If you do not define either *voh* or *vol*, Star-Hspice uses *vth* (default or defined).

Vol Statement

The *vol* statement specifies the logic low voltage of each output signal to which the mask applies.

Example

If you do not specify the logic low voltage by a *vol* statement, Star-Hspice assumes 0.0. If you apply more than one *vol* statements to a signal, the last value overrules the previous ones, and Star-Hspice issues a warning.

```
vol 0.5
vol 0.5 1 1 1 137F 11111111
```

The *vol* statements have no effect on input signals.

Note: If you do not define either *voh* or *vol*, Star-Hspice uses *vth* (default or defined).

Avant!

Chapter 6
Multi-Terminal Networks

This chapter covers various topics related to the S Element:

- Using Scattering Parameter Element

Using Scattering Parameter Element

A new S Element, in conjunction with the generic frequency-domain model (`.MODEL SP`), provides a convenient way to describe a multi-terminal network. Currently, S (scattering) and Y parameters are supported. S Element can be used in AC and DC analyses.

In particular, the S parameter in S Element represents the generalized scattering parameter **S** for a multi-terminal network, which is defined as:

$$\mathbf{v_{ref}} = \mathbf{S} \cdot \mathbf{v_{inc}} \ .$$

where boldface lower-case and upper-case symbols denote vectors and matrices, respectively. $\mathbf{v_{inc}}$ and $\mathbf{v_{ref}}$ are the incident and reflected voltage wave vectors (see Figure 6-1). The S parameter can be converted to Y parameter using the following formula:

$$\mathbf{Y} = \mathbf{Y_r}(\mathbf{I} - \mathbf{S})(\mathbf{I} + \mathbf{S})^{-1}.$$

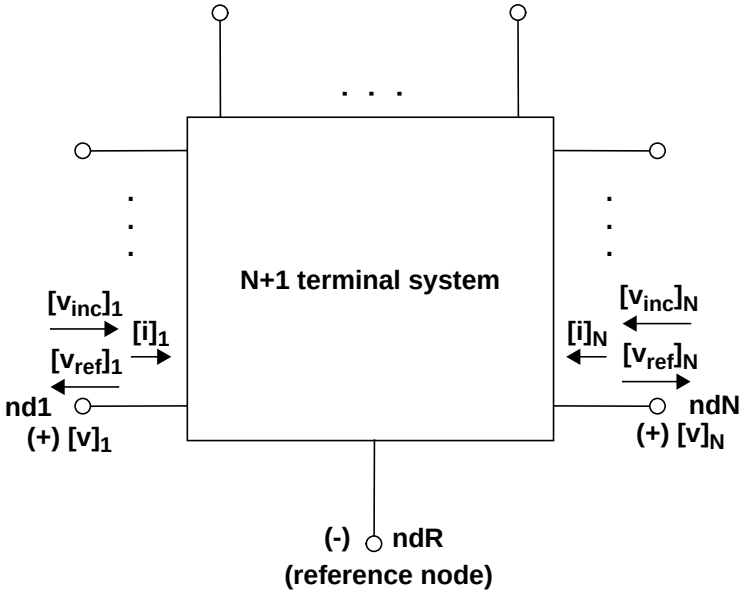
where $\mathbf{Y_r}$ is the characteristic admittance matrix of the reference system, which is related to the characteristic impedance matrix $\mathbf{Z_r}$ by:

$$\mathbf{Y_r} = \mathbf{Z_r^{-1}} \ .$$

Similarly, Y parameter can be converted to S parameter as follows:

$$\mathbf{S} = (\mathbf{Y_r} + \mathbf{Y})^{-1}(\mathbf{Y_r} - \mathbf{Y}) \ .$$

Figure 6-1: Terminal Node Notation



Syntax

The syntax of the S Element is:

```
Sxxx nd1 nd2 ... ndN ndR FQMODEL=name [TYPE=val Zo=val
   Zof=name]
```

`nd1 nd2 ... ndN`	N signal nodes (see Figure 6-1).
ndR	Reference node.
FQMODEL	`.MODEL` statement of type sp, which defines the frequency behavior of S or Y parameter.
TYPE	Parameter type: ■ S: scattering parameter (default) ■ Y: Y parameter

Zo	Characteristic impedance value of the reference line (frequency-independent). For multi-terminal cases (N>1), the characteristic impedance matrix of the reference lines is assumed to be diagonal and its diagonal values are set to Zo. More general types of a reference line system can be specified using Zof. Default=50Ω.
Zof	Name of the frequency-varying model that defines the frequency behavior of the reference system. If both Zo and Zof have been defined, then Zof has precedence.

Frequency Table Model

The Frequency Table Model is a generic model that can be used to describe frequency-varying behavior. Currently, it is used by S Element and .NOISENPT.

Syntax

The syntax of the .MODEL model card is:

```
.MODEL name sp [N=val FSTART=val FSTOP=val NI=val
+ SPACING=val MATRIX=val VALTYPE=val INFINITY=matrixval
+ INTERPOLATION=val EXTRAPOLATION=val]
+ [DATA=(npts ...)] [DATAFILE=filename]
```

name	Model name.
N	Matrix dimension (number of signal terminals). Values other than 1 must be specified before setting INFINITY and DATA. Default=1.
FSTART	Starting frequency point for data. Default=0.

FSTOP	Final frequency point for data (used only for the LINEAR and LOG spacing formats).
NI	Number of frequency points per interval. Used only for the DEC and OCT spacing formats. Default=10.
npts	Number of data points.
SPACING	Data sample spacing format: ■ LIN (LINEAR): uniform spacing with the frequency step of (FSTOP-FSTART)/(npts-1). Default. ■ OCT: octave variation with FSTART as the starting frequency and NI points per octave. npts determines the final frequency. ■ DEC: decade variation with FSTART as the starting frequency and NI points per decade. npts determines the final frequency. ■ LOG: logarithmic spacing with FSTART and FSTOP as the starting and final frequencies. ■ POI(NONUNIFORM): nonuniform spacing. Data points are paired with frequency points.
MATRIX	Matrix (data point) format: ■ SYMMETRIC: symmetric matrix. Only the lower-half triangle portion of a matrix is specified. Default. ■ HERMITIAN: similar to SYMMETRIC but off-diagonal terms are complex-conjugate of each other. ■ NONSYMMETRIC: nonsymmetric matrix. A full matrix is specified.

VALTYPE	Data type of matrix elements: ■ REAL: real entry ■ CARTESIAN: complex number in real/ imaginary format. Default. ■ POLAR: complex number in polar format. Angles are specified in radian.
INFINITY	Data point at infinity. Typically real-valued. Data format must be consistent with MATRIX and VALTYPE specifications. This point is not counted in npts.
DC	Data port at DC. Typically real-valued. Data format must be consistent with MATRIX and VALTYPE specifications. This point is not counted in npts. It is required to specify DC point or data point at frequency=0.
INTERPOLATION	Interpolation scheme: ■ STEP: piecewise step. Default. ■ LINEAR: piecewise linear ■ SPLINE: b-spline curve fit
EXTRAPOLATION	Extrapolation scheme during simulation: ■ NONE: no extrapolation is allowed. Star-Hspice will terminate if data point is required outside of the specified range. ■ STEP: the last boundary point is used. Default. ■ LINEAR: linear extrapolation using the last two boundary points. If the data point at the infinity is specified, then no extrapolation is performed and the infinity value is used.

DATA

Specifies data points. Syntax for LIN spacing:

```
.MODEL name sp SPACING=LIN
+ [N=dim] FSTART=f0 DF=f1
+ DATA=npts d1 d2 ...
```

Syntax for OCT or DEC spacing:

```
.MODEL name sp SPACING=DEC or OCT
+ [N=dim] FSTART=f0 NI=n_per_intval
+ DATA=npts d1 d2 ...
```

Syntax for POI spacing:

```
.MODEL name sp
+ SPACING=NONUNIFORM [N=dim]
+ DATA=npts f1 d1 f2 d2 ...
```

DATAFILE

This option allows users to specify data points in an external file. The content of this file must be only raw numbers without any suffixes, comments or continuation letters. The order of data must follow the DATA statement.
This feature is useful as this data file does not have limitation on line length so users can enter a large number of data points.

Note: The interpolation and extrapolation are performed after S parameter data are converted to Y parameter internally.

Example

The two outputs from the resistor and S parameter modeling should be matched exactly in this example. See Table 6-1 for the input file listing and Figure 6-2 for an illustration of a transmission line with a resistive termination.

Figure 6-2: Transmission line with a resistive termination

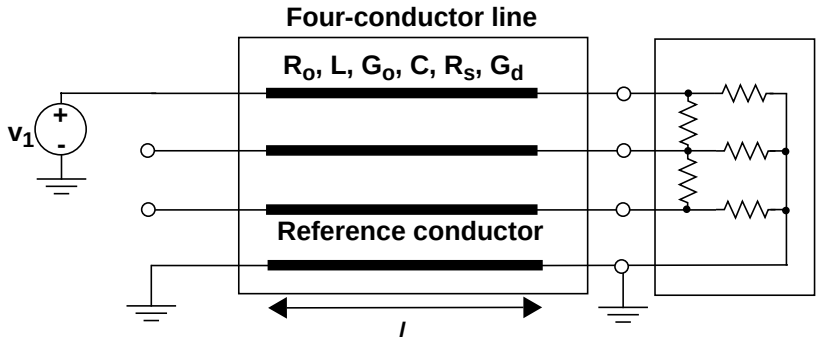


Table 6-1: Input File Listing

Header, options, and sources	`*S parameter ex1: x-line with a resistive`
	`+ termination`
	`.OPTIONS POST`
	`V1 i1 0 ac=1v`

Analysis	`.AC lin 500 0Hz 30MegHz` `.DC v1 0v 5v 1v`

Transmission line (W element)	`W1 i1 i2 i3 0 o1 o2 o3 0 RLGCMODEL=wrlgc N=3`
	`+ L=0.97`
	`.MODEL wrlgc sp MODELTYPE=RLGC N=3`
	`+ Lo = 2.78310e-07`
	`+ 8.75304e-08 3.29391e-07`
	`+ 3.65709e-08 1.15459e-07 3.38629e-07`
	`+ Co = 1.41113e-10`
	`+ -2.13558e-11 9.26469e-11`
	`+ -8.92852e-13 -1.77245e-11 8.72553e-11`

Termination	`x1 o1 o2 o3 0 terminator`

Frequency model definition	`.MODEL fmod sp N=3 FSTOP=30MegHz`
	`+ DATA= 1`
	`+ -0.270166 0.0`
	`+ 0.322825 0.0 -0.41488 0.0`
	`+ 0.17811 0.0 0.322825 0.0 -0.270166 0.0`

Resistor elements	.SUBCKT terminator n1 n2 n3 ref R1 n1 ref 75 R2 n2 ref 75 R3 n3 ref 75 R12 n1 n2 25 R23 n2 n3 25 .ends terminator
Equivalent S parameter element	.ALTER S parameter case .SUBCKT terminator n1 n2 n3 ref S1 n1 n2 n3 ref FQMODEL=fmod .ENDS terminator .END

This is an example of a transmission line with a capacitive network termination.

Frequency model definition	.MODEL fmod sp N=3 FSTOP=30MegHz + DATA= 2 + 1.0 0.0 + 0.0 0.0 1.0 0.0 + 0.0 0.0 0.0 0.0 1.0 0.0 + 0.97409 -0.223096 + 0.00895303 0.0360171 0.964485 -0.25887 + -0.000651487 0.000242442 0.00895303 + 0.0360171 0.97409 -0.223096
Using capacitive elements	.SUBCKT terminator n1 n2 n3 ref C1 n1 ref 10pF C2 n2 ref 10pF C3 n3 ref 10pF C12 n1 n2 2pF C23 n2 n3 2pF .ENDS terminator

The two outputs from the resistor and S parameter modeling differ slightly, due to the linear frequency dependency related to the capacitor. This difference can be removed by using the linear interpolation scheme in .MODEL.

This is an example of a transmission line with S parameter.

Figure 6-3: 3-Conductor Transmission line

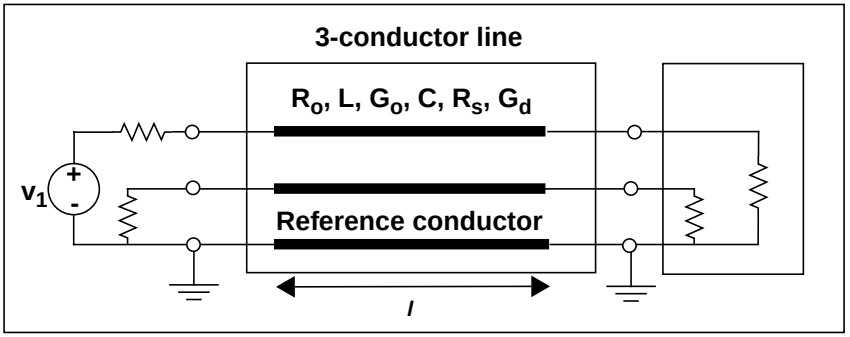


Table 6-2: Input File Listing

Header, options and sources	`*S parameter ex3: modeling x-line using`
	`+ S parameter`
	`.OPTIONS POST`
	`vin in0 0  ac=1`

Analysis	`.AC lin 100 0 1000meg` `.DC vin 0 1v 0.2v`
Transmission line	`X1  in1 in2 out1 out2 0 x-line`

Termination	`R1  in0   in1   28`
	`R2  in2   0     28`
	`R3  out1  0     28`
	`R4  out2  0     28`

W Element RLGC model definition	`.MODEL m2 W ModelType=RLGC, N=2`
	`+ Lo= 0.178e-6`
	`+     0.0946e-7 0.178e-6`
	`+ Co= 0.23e-9`
	`+    -0.277e-11 0.23e-9`
	`+ Ro= 0.97`
	`+     0         0.97`
	`+ Go= 0`
	`+     0         0`
	`+ Rs= 0.138e-3`
	`+     0         0.138e-3`
	`+ Gd= 0.29e-10`
	`+     0         0.29e-10`

Frequency model definition	.MODEL SM2 sp N=4 FSTART=0 FSTOP=1e+09 + SPACING=LINEAR + DATA= 60 + 0.00386491 0 + 0 0 0.00386491 0 + 0.996135 0 0 0 0.00386491 0 + 0 0 0.996135 0 0 0 0.00386491 0 + -0.0492864 -0.15301 + 0.00188102 0.0063569 -0.0492864 + -0.15301 0.926223 -0.307306 0.000630484 + -0.00154619 0.0492864 -0.15301 + 0.000630484 -0.00154619 0.926223 + -0.307306 0.00188102 0.0063569 + -0.0492864 -0.15301 -0.175236 -0.241602 + 0.00597 0.0103297 -0.175236 -0.241602 + 0.761485 -0.546979 0.00093508 + -0.00508414 -0.175236 -0.241602 + 0.00093508 -0.00508414 0.761485 + -0.546979 0.00597 0.0103297 -0.175236 + -0.241602 + ...
Equivalent S parameter element	.SUBCKT terminator n1 n2 n3 ref S1 n1 n2 n3 ref FQMODEL=SM2 .ENDS terminator

Avant!

Chapter 7
Parameters and Functions

Parameters are similar to variables found in most programming languages. They hold a value that is either assigned at design time or calculated during the simulation based on circuit solution values. Parameters are used for storage of static values for a variety of quantities (resistance, source voltage, rise time, and so on), or used in sweep or statistical analysis.

This chapter describes the basic usage of parameters within a Star-Hspice netlist:

Using Parameters in Simulation (.PARAM)

Parameter Definition

Parameters in Star-Hspice are names that have associated numeric values. You can use any of the following methods to define parameters:

Simple assignment	`.PARAM <SimpleParam> = 1e-12`
Algebraic definition	`.PARAM <AlgebraicParam> = 'SimpleParam*8.2'`
User-defined function	`.PARAM <MyFunc( x, y )> = 'Sqrt((x*x)+(y*y))'`
Subcircuit default	`.SUBCKT <SubName> <ParamDefName> = + <Value>`
Predefined analysis function	`.PARAM <mcVar> = Agauss(1.0,0.1)` (see "Statistical Analysis and Optimization" on page Chapter 131)
.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 Chapter 838)

A parameter definition in Star-Hspice always takes the last value found in the Star-Hspice input (subject to local versus global parameter rules). Thus, the definitions below assign the value 3 to the parameter DupParam.

```
.PARAM DupParam = 1
...
.PARAM DupParam = 3
```

The value 3 will be substituted for all instances of DupParam, including instances that occur earlier in the input than the .PARAM DupParam = 3 statement.

All parameter values in Star-Hspice are IEEE double floating point numbers.

Parameter resolution order is as follows:

1. Resolve all literal assignments
2. Resolve all expressions
3. Resolve all function calls

Parameter passing order is shown in Table 7-1.

Table 7-1: 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)

Parameter Assignment

A constant real number or an algebraic expression of real values, predefined function, user-defined function, or circuit or model values can be assigned to parameters. A complex expression must be enclosed in single quotes in order to invoke the Star-Hspice algebraic processor. A simple expression consists of a single parameter name. The parameter keeps the assigned value unless there is a later definition that changes its value, or it is assigned a new value by an algebraic expression during simulation. There is no warning if a parameter is reassigned.

Syntax

```
.PARAM <ParamName> = <RealNumber>
.PARAM <ParamName> = '<Expression>' $ Quotes are mandatory
.PARAM <ParamName1> = <ParamName2> $ Cannot be recursive!
```

Numerical Example

```
.PARAM TermValue = 1g
     rTerm Bit0 0 TermValue
     rTerm Bit1 0 TermValue

...
```

Expression Example

```
.PARAM Pi = '355/113'
.PARAM Pi2 = '2*Pi'

.PARAM npRatio = 2.1
.PARAM nWidth = 3u
.PARAM pWidth = 'nWidth * npRatio'
Mp1  ... <pModelName> W = pWidth
Mn1  ... <nModelName> W = nWidth
...
```

Inline Assignments

To define circuit values by a direct algebraic evaluation:

```
r1 n1 0 R = '1k/sqrt(HERTZ)' $ Resistance related to frequency
```

Parameters in Output

To use an algebraic expression as an output variable in a .PRINT, .PLOT, or .PROBE statement, use the PAR keyword (see “Specifying Simulation Output” on page Chapter 81 for more information on simulation output). For example:

```
.PRINT DC v(3) gain = PAR('v(3)/v(2)') PAR('v(4)/v(2)')
```

User-Defined Function Parameters

A user-defined function can be defined similar to the parameter assignment except for the fact that it cannot be nested more than three deep.

Syntax

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

Example

```
.PARAM CentToFar (c)        = '(((c*9)/5)+32)'
.PARAM F(p1,p2)             = 'Log(Cos(p1)*Sin(p2))'
.PARAM SqrdProd (a,b)       = '(a*a)*(b*b)'
```

Subcircuit Default Definitions

When you use hierarchical sub-circuits, you can pick default values for circuit elements. You typically use this in cell definitions, to simulate the circuit with typical values (see Using Subcircuits on page 3-50).

Syntax

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

where <*SubDefaultsList*> is

```
<SubParam1> = <Expression> [<SubParam2> = <Expression> ...]
```

Subcircuit Parameter Example

This example implements an inverter with a Strength parameter. By default, the inverter can drive three devices. Enter a new value for the Strength parameter in the element line, to select larger or smaller inverters for the application.

```
.SUBCKT Inv a y Strength = 3
     Mp1 <MosPinList> pMosMod L = 1.2u W = 'Strength * 2u'
     Mn1 <MosPinList> nMosMod L = 1.2u W = 'Strength * 1u'
.ENDS

...
xInv0 a y0 Inv                $ Default devices:
                              p device = 6u,n device = 3u
```

```
xInv1 a y1 Inv Strength = 5   $ p device = 10u,
                              n device = 5u
xInv2 a y2 Inv Strength = 1   $ p device =  2u,
                              n device = 1u
...
```

Parameter Scoping Example

The following example shows explicitly the difference between local and global scoping for parameter usage in sub-circuits.

Given the input netlist fragment:

```
.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
```

with the global parameter scoping option .OPTION PARHIER = GLOBAL set, and the following input statements

```
...
xInv0 a y0 Inv                        $ Xinv0.Mp1 width = 1u
xInv1 a y1 Inv DefPwid = 5u           $ Xinv1.Mp1 width = 5u
.MEASURE TRAN Wid0 PARAM = 'lv2(xInv0.Mp1)' $ lv2 is the
.MEASURE TRAN Wid1 PARAM = 'lv2(xInv1.Mp1)' $ template for the
                                      $ channel width
                                      $'lv2(xInv1.Mp1)'

 ...
```

the following results are produced in the listing file:

```
    wid0   =    1.0000E-06
    wid1   =    1.0000E-06
```

With the local parameter scoping option .OPTION PARHIER = LOCAL set, and the following statements

```
...
xInv0 a y0 Inv                        $ Xinv0.Mp1 width = 1u
xInv1 a y1 Inv DefPwid = 5u           $ Xinv1.Mp1 width = 1u:
.MEASURE TRAN Wid0 PARAM = 'lv2(xInv0.Mp1)' $ override the
                                      global .PARAM
.MEASURE TRAN Wid1 PARAM = 'lv2(xInv1.Mp1)'
...
```

the following results are produced in the listing file:

```
wid0   =    2.0000E-06
wid1   =    5.0000E-06
```

Predefined Analysis Function

Star-Hspice has specialized analysis types, primarily Optimization and Monte Carlo, that require a method of controlling the analysis. The parameter definitions related with these analysis types are described in “Statistical Analysis and Optimization” on page Chapter 131.

Measurement Parameters

.MEASURE statements in Star-Hspice produce a type of parameter called a measurement parameter. In general, the rules for measurement parameters are the same as the rules for standard parameters, with one exception: measurement parameters are not defined in a .PARAM statement, but are defined directly in a .MEASURE statement. The detailed syntax and usage of the .MEASURE statement is described in “Specifying User-Defined Analysis (.MEASURE)” on page Chapter 838.

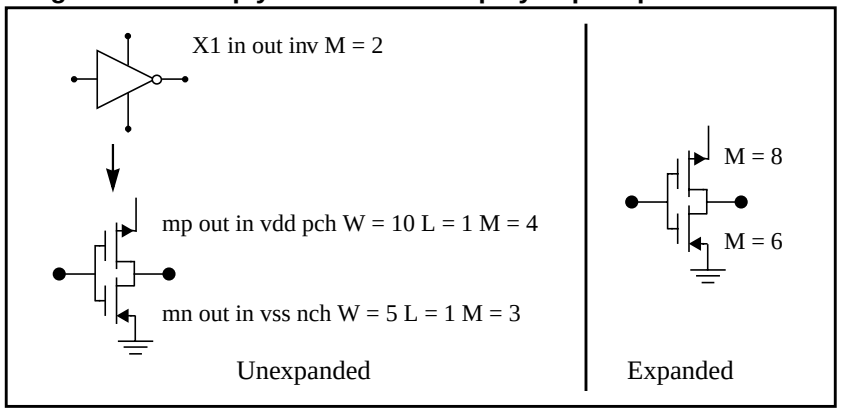
Multiply Parameter

The multiply parameter M is a special keyword common to all elements (except for voltage sources) and sub-circuits. It multiplies the internal component values to give the effect of making parallel copies of the element or subcircuit. To simulate the effect of 32 output buffers switching simultaneously, only one subcircuit call needs to be placed, such as:

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

Multiply works hierarchically. A subcircuit within a subcircuit is multiplied by the product of the multiply parameters at both levels.

Figure 7-1: Multiply Parameters Simplify Flip-Flop Initialization



Using Algebraic Expressions[1]

You can replace any parameter defined in the netlist by an algebraic expression with quoted strings. Then, these expressions may be used as output variables in the .PLOT, .PRINT, and .GRAPH statements. Using algebraic expressions can expand your options in the input netlist file. Some usages 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)')
```

Algebraic Expressions for Output

Syntax

```
PAR('algebraic expression')
```

In addition to using quotations, the expression must be defined inside the PAR( ) statement for output.The continuation character for quoted parameter strings is a double backslash, "\\". (Outside of quoted strings, the single backslash, "\", is the continuation character.)

---

[1] *Star-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, since constants are only single-precision numbers (7 digits).*

Built-In Functions

In addition to simple arithmetic operations (+, -, *, /), Star-Hspice provides a number of built-in functions that you can use in expressions. The Star-Hspice built-in functions are listed in Table 7-2.

Table 7-2: Star-Hspice Built-in Functions *(Sheet 1 of 5)*

HSPICE Form	Function	Class	Description
sin(x)	sine	trig	Returns the sine of x (radians)
cos(x)	cosine	trig	Returns the cosine of x (radians)
tan(x)	tangent	trig	Returns the tangent of x (radians)
asin(x)	arc sine	trig	Returns the inverse sine of x (radians)
acos(x)	arc cosine	trig	Returns the inverse cosine of x (radians)
atan(x)	arc tangent	trig	Returns the inverse tangent of x (radians)
sinh(x)	hyperbolic sine	trig	Returns the hyperbolic sine of x (radians)
cosh(x)	hyperbolic cosine	trig	Returns the hyperbolic cosine of x (radians)
tanh(x)	hyperbolic tangent	trig	Returns the hyperbolic tangent of x (radians)
abs(x)	absolute value	math	Returns the absolute value of x: |x|
sqrt(x)	square root	math	Returns the square root of the absolute value of x: sqrt(-x) = -sqrt(|x|)

Table 7-2: Star-Hspice Built-in Functions *(Sheet 2 of 5)*

HSPICE Form	Function	Class	Description
pow(x,y)	absolute power	math	Returns the value of x raised to the integer part of y: $x^{(\text{integer part of y})}$
pwr(x,y)	signed power	math	Returns the absolute value of x raised to the y power, with the sign of x: (sign of x)$\|x\|^y$
log(x)	natural logarithm	math	Returns the natural logarithm of the absolute value of x, with the sign of x: (sign of x)log(|x|)
log10(x)	base 10 logarithm	math	Returns the base 10 logarithm of the absolute value of x, with the sign of x: (sign of x)$\log_{10}(\|x\|)$
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.
sgn(x)	return sign	math	Returns -1 if x is less than 0, 0 if x is equal to 0, and 1 if x is greater than 0
sign(x,y)	transfer sign	math	Returns the absolute value of x, with the sign of y: (sign of y)|x|
min(x,y)	smaller of two args	control	Returns the numeric minimum of x and y

Table 7-2: Star-Hspice Built-in Functions *(Sheet 3 of 5)*

HSPICE Form	Function	Class	Description
max(x,y)	larger of two args	control	Returns the numeric maximum of x and y
lv(<Element>) or lx(<Element>)	element templates	various	Returns various element values during simulation. See “Element Template Output” in Chapter 7 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 Chapter 821 for more information.
(*cond*) ? x : y	ternary operator		Returns *x* if *cond* is not zero. Otherwise, returns *y.* **Syntax:** .para x=(condition) ?y:z
<	relational operator (less than)		Returns 1 if the left eft operand is less than the right operand. Otherwise, returns 0. **Syntax:** .para x=y<z (y less than z)
<=	relational operator (less than or equal)		Returns 1 if the left eft operand is less than or equal to the right operand. Otherwise, returns 0. **Syntax:** .para x=y<=z (y less than or equal to z)

Table 7-2: Star-Hspice Built-in Functions *(Sheet 4 of 5)*

HSPICE Form	Function	Class	Description
>	relational operator (greater than)		Returns 1 if the left eft operand is greater than the right operand. Otherwise, returns 0. **Syntax:** .para x=y>z (y greater than z)
>=	relational operator (greater than or equal)		Returns 1 if the left eft operand is greater than or equal to the right operand. Otherwise, returns 0. **Syntax:** .para x=y>=z (y greater than or equal to z)
==	equality		Returns 1 if the operands are equal. Otherwise, returns 0. **Syntax:** .para x=y==z (y equal to z)
!=	inequality		Returns 1 if the operands are not equal. Otherwise, returns 0. **Syntax:** .para x=y!=z (y not equal to z)
&&	Logical AND		Returns 1 if neither operand is zero. Otherwise, returns 0. **Syntax:** .para x=y&&z (y AND z)

Table 7-2: Star-Hspice Built-in Functions *(Sheet 5 of 5)*

HSPICE Form	Function	Class	Description
||	Logical OR		Returns 1 if either or both operands are not zero. Returns 0 only if both operands are zero. **Syntax:** .para x=y||z (y OR z)

Example

```
.parameters p1=4 p2=5 p3=6
r1 1 0 value=(p1 ? p2+1 : p3)
```

Star-Hspice reserves the variable names listed in Table 7-3 for use in elements such as E, G, R, C, and L. You cannot use them for any other purpose in your netlist (in .PARAM statements, for example).

Table 7-3: Star-Hspice Special Variables

HSPICE Form	Function	Class	Description
time	current simulation time	control	Parameterizes the current simulation time during transient analysis.
temper	current circuit temperature	control	Parameterizes the current simulation temperature during transient/temperature analysis.
hertz	current simulation frequency	control	Parameterizes the frequency during AC analysis.

User-Defined Functions

An expression can contain parameters that have not yet been defined. A function must have at least one argument, and not more than two. Functions can be redefined.

Syntax

```
fname1 (arg1, arg2) = expr1 (fname2
(arg1, ...) = expr2) off
```

where:

fname	Specifies function name. This parameter must be distinct from array names and built-in functions. Subsequently defined functions must have all their embedded functions previously defined.
arg1, *arg2*	Specifies variables used in the expression.
off	Voids all user-defined functions.

Example

```
f(a,b) = POW(a,2)+a*b g(d) = SQRT(d) h(e) = e*f(1,2)-
g(3)
```

Parameter Scoping and Passing

Parameterized sub-circuits provide a method of reducing the number of similar cells that must be created to provide enough functionality within your library. Star-Hspice allows you to pass circuit parameters into hierarchical designs, allowing you to configure a cell at runtime.

For example, if you parameterize the initial state of a latch in its subcircuit definition, then you can override this initial default in the instance call. Only one cell needs to be created to handle both initial state versions of the latch.

You also can parameterize a cell to reflect its layout. Parameterize a MOS inverter to simulate a range of inverter sizes with only one cell definition. In addition, you can perform Monte Carlo analysis or optimization on a parameterized cell.

The way you choose to handle hierarchical parameters depends on how you construct and analyze your cells. You can choose to construct a design in which information flows from the top of the design down into the lowest hierarchical levels. Centralizing the control at the top of the design hierarchy involves setting *global* parameters. You can also choose to construct a library of small cells that are individually controlled from within by setting *local* parameters, and build upwards to the block level.

This section describes the scope of Star-Hspice parameter names, and how Star-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.

You can risk poor library integrity when using libraries from different vendors in a single design. Because there is no standardization between vendors on what circuit parameters are named, it is possible that two components can include the same parameter name with different functions. Suppose that the first vendor builds a library that uses the name *Tau* as a parameter to control one or more sub-circuits in their library. Now suppose that the second vendor uses *Tau* to control

a different aspect of their library. Setting a global parameter named *Tau* to control one library also modifies the behavior of the second library, which might not be the intent.

When the *scope* of a higher level parameter is *global* to all sub-circuits at lower levels of the design hierarchy, lower level parameter values are overridden by the values from higher level definitions if the names are the same. The scope of a lower level parameter is *local* to the subcircuit in which the parameter is defined (but global to all sub-circuits that are even lower in the design hierarchy). The local scoping rules in Star-Hspice solve the problem of lower level parameters being overridden by higher level parameters of the same name when that is not desired.

Reusing Cells

Problems with parameter names also occur when different groups collaborate on a design. Because Star-Hspice global parameters prevail over local parameters, all designers are required to 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 standardized libraries. You can avoid this situation by using local parameter scoping that encapsulates all information about a section of a design within that section.

Creating Parameters in a Library

To ensure that critical, user-supplied parameters are present in a Star-Hspice netlist at simulation time, Star-Hspice allows the use of “illegal defaults”—that is, defaults that cause the simulator to abort if there are no overrides for the defaults.

Library cells that include illegal defaults require that the user provide a value for each and every instance of those cells. Failure to do so causes the Star-Hspice simulation to abort.

An example is the use of a default MOSFET width of 0.0. This causes Star-Hspice to abort because this parameter is required by the Star-Hspice MOSFET models.

Example

```
...
* 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

* Invocation of symbols in a design
x1 A Y1 Inv                $ Bad! No widths specified
x2 A Y2 Inv Wid = 1u       $ Overrides illegal value for Wid
```

This simulation would abort on subcircuit instance *x1* because the required parameter *Wid* is never set on the subcircuit instance line. Subcircuit *x2* would simulate correctly. Additionally, the instances of the *Inv* cell are subject to accidental interference because the global parameter *Wid* is exposed outside the domain of the library. Anyone could have specified an alternative value for the parameter in another section of the library or the design, which could have prevented the simulation from catching the condition present on *x1*.

Example

Now consider the effect of a global parameter whose name conflicts with the library internal parameter *Wid*. Such a global parameter could be specified by the user or in a different library. In this example, the user of the library has specified a different meaning for the parameter *Wid* to be used in the definition of 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

* Invocation of 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, the simulation succeeds, although incorrectly. There is no warning message that the inverter x1 has no widths assigned, 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 current sweep's Wid value.

In global scoping, the highest level name prevails in name conflict resolution. In local scoping, the lowest level name is used.

The parameter inheritance method allows you to specify that local scoping rules be used. This feature can cause different results than you have obtained with Star-Hspice releases prior to 95.1 on existing circuits.

With 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), and results in the correct device sizes for x2. This change might affect your simulation results if a circuit like the second one shown above is created intentionally or accidentally.

As an alternative to width testing in the Example 2 netlist, it is also possible to achieve a limited version of library integrity with the .OPTION DEFW. This option specifies the default width for all MOS devices during a simulation. Because part of the definition is still in the domain of the top-level circuit, this method still suffers from the possibility of making unwanted changes to library values without notification by the simulator.

Table 7-4 outlines and compares the three primary methods for configuring libraries to achieve required parameter checking in the case of default MOS transistor widths.

Table 7-4: Methods for Configuring Libraries

Method	Parameter Location	Pros	Cons
Local	On a .SUBCKT definition line	The library is protected from global circuit parameter definitions unless the user wishes to override. Single location for default values.	Cannot be used with releases of Star-Hspice prior to Release 95.1.
Global	At the global level and on .SUBCKT definition lines	Works with older Star-Hspice versions	The library can be changed by indiscrete user or other vendor assignment and by the intervening hierarchy. Cannot override a global value at a lower level.
Special	.OPTION DEFW statement	Simple to do	Third party libraries or other sections of the design might depend on the option DEFW.

Parameter Defaults and Inheritance

Use the .OPTION parameter PARHIER to specify scoping rules.

Syntax

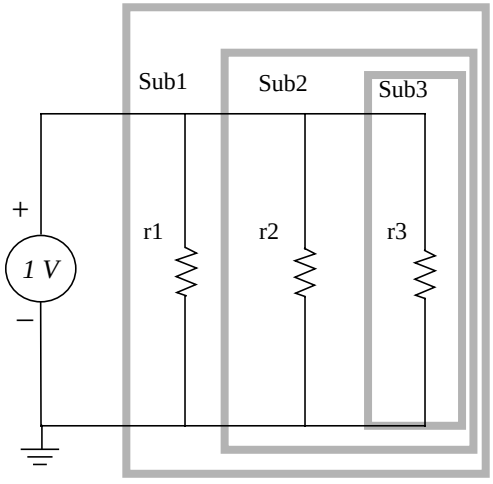
```
.OPTION PARHIER = < GLOBAL | LOCAL >
```

The default setting is GLOBAL, which gives the same scoping rules that Star-Hspice used prior to Release 95.1.

Figure 7-2 shows a flat representation of a hierarchical circuit that contains three resistors.

Each of the three resistors gets its simulation time resistance from the parameter named *Val*. The *Val* parameter is defined in four places in the netlist, with three different values.

Figure 7-2: Hierarchical Parameter Passing Problem




```
TEST OF PARHIER
.OPTION list node post = 2
+ ingold = 2
+ parhier = <Local|Global>
.PARAM Val = 1
x1 n0 0 Sub1
.SubCkt Sub1 n1 n2 Val = 1
   r1 n1 n2 Val
   x2 n1 n2 Sub2
.Ends Sub1
.SubCkt Sub2 n1 n2 Val = 2
   r2 n1 n2 Val
   x3 n1 n2 Sub3
.Ends Sub2
.SubCkt Sub3 n1 n2 Val = 3
   r3 n1 n2 Val
.Ends Sub3
```

There are two possible solutions for the total resistance of the chain: 0.3333 Ω and 0.5455 Ω .

The PARHIER option allows you to specify which parameter value prevails when parameters with the same name are defined at different levels of the design hierarchy.

Under global scoping rules, in the case of name conflicts, the top-level assignment .PARAM Val = 1 overrides the subcircuit defaults, and the total is 0.3333 Ω Under local scoping rules, the lower level assignments prevail, and the total is 0.5455 Ω (one, two and three ohms in parallel).

The example shown in Figure 7-2 produces the results in Table 7-5, based on the setting of the local/global PARHIER option:

Table 7-5: 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 Problems

Changes in scoping rules can cause different simulation results for designs created prior to Star-Hspice Release 95.1 from designs created after that release. Use the following checklist to determine if you will see simulation differences with the new default scoping rules. These checks are especially important if your netlists contain devices from multiple vendors' libraries.

- Check your sub-circuits for parameter defaults on the .SUBCKT or .MACRO line.
- Check your sub-circuits for a .PARAM statement within a .SUBCKT definition.
- Check your circuits for global parameter definitions using the .PARAM statement.
- If any of the names from the first three checks are identical, then set up two Star-Hspice jobs, one with .OPTION PARHIER = GLOBAL and one with .OPTION PARHIER = LOCAL, and look for differences in your output.

Avant!

Chapter 8
Specifying Simulation Output

Use output format statements and variables to display steady state, frequency, and time domain simulation results. These variables also permit you to use behavioral circuit analysis, modeling, and simulation techniques. Display electrical specifications such as rise time, slew rate, amplifier gain, and current density using the output format features.

This chapter discusses the following topics:

Using Output Statements

Output Commands

Star-Hspice output statements are contained in the input netlist file and include .PRINT, .PLOT, .GRAPH, .PROBE, .MEASURE, and .DOUT. Each statement specifies the output variables and type of simulation result to be displayed—for example, .DC, .AC, or .TRAN. With the use of .OPTION POST, all output variables referenced in .PRINT, .PLOT, .GRAPH, .PROBE, .MEASURE, and .DOUT statements are put into the interface files for AvanWaves. AvanWaves allows high resolution, post simulation, and interactive display of waveforms.

Output Statement	Description
.PRINT	Prints numeric analysis results in the output listing file (and post-processor data if .OPTION POST is used).
.PLOT	Generates low-resolution (ASCII) plots in the output listing file (and post-processor data if .OPTION POST is used).
.GRAPH	Generates high-resolution plots for specific printing devices (HP LaserJet for example) or in PostScript format, intended for hard-copy outputs without a using a post-processor.
.PROBE	Outputs data to post-processor output files but not to the output listing (used with .OPTION PROBE to limit output).
.MEASURE	Prints to output listing file the results of specific user-defined analyses (and post-processor data if .OPTION POST is used).

Output Statement	Description
.DOUT	Specifies the expected final state of an output signal.

.Measure Performance

If you specify a large number of `.measure` statements, Star-Hspice simulation can require a long time to run, on the order of several minutes to several hours. Overall simulation run time depends on the number of `.measure` statements to process for each iteration, and the number of iterations required to achieve convergence.

To reduce simulation run time, you should place similar variables together, when you list them in the `.measure` statement.

Examples

Example 1 - Original Case (Slower, due to repeated switching between the `v1` and `v2` variables):

```
.meas tran val1 AVG v(1) FROM=0ms TO=50ms
.meas tran val2 AVG v(2) FROM=0ms TO=50ms
.meas tran val3 AVG v(1) FROM=50ms TO=100ms
.meas tran val4 AVG v(2) FROM=50ms TO=100ms
```

Example 2 - Improved Case (Faster):

```
.meas tran val1 AVG v(1) FROM=0ms TO=50ms
.meas tran val3 AVG v(1) FROM=50ms TO=100ms
.meas tran val2 AVG v(2) FROM=0ms TO=50ms
.meas tran val4 AVG v(2) FROM=50ms TO=100ms
```

In the second example, all `V(1)` variables are listed consecutively, followed by all `v(2)` variables. In this second case, Star-Hspice applies all measurements to a single variable (`v1`) at the same time. This reduces overall simulation run time, compared to switching back to the same variable repeatedly, when you do **not** sort the .measure list by variable name.

To help you automatically sort large numbers of `.measure` statements in this way, you can use the `.option meassort` statement.

Syntax

`.option meassort=0` (the default; does not sort `.measure` statements)

`.option meassort=1` (internally sorts `.measure` statements)

Set this option to `1` only if you use a large number of `.measure` statements, where it is important to list similar variables together, to reduce simulation run time. For a small number of `.measure` statements, turning on internal sorting might slow-down the simulation while sorting, compared to **not** sorting first.

Output Variables

The output format statements require special output variables to print or plot analysis results for nodal voltages and branch currents. There are five groups of output variables: DC and transient analysis, AC analysis, element template, .MEASURE statement, and parametric analysis.

DC and transient analysis displays individual nodal voltages, branch currents, and element power dissipation.

AC analysis displays imaginary and real components of a nodal voltage or branch current, as well as the 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 are user-defined. They represent the electrical specifications measured in a .MEASURE statement analysis.

Parametric analysis variables are mathematically defined expressions operating on user-specified nodal voltages, branch currents, element template variables, or other parameters. You can perform behavioral analysis of simulation results using these variables. See "Using Algebraic Expressions" on page -9 for information about parameters in Star-Hspice.

Displaying Simulation Results

The following section describes the statements used to display simulation results for your specific requirements.

.PRINT Statement

The .PRINT statement specifies output variables for which values are printed.

- The maximum number of variables in a single .PRINT statement is 32. You can use additional .PRINT statements for more output variables.
- To simplify parsing of the output listings, a single "x" printed in the first column indicates the beginning of the .PRINT output data, and a single "y" in the first column indicates the end of the .PRINT output data.

Syntax

```
.PRINT antype ov1 <ov2 … ov32>
```

antype Specifies the type of analysis for outputs. *Antype* is one of the following types: DC, AC, TRAN, NOISE, or DISTO.

ov1 … Specifies the output variables to print. These are voltage, current, or element template variables from a DC, AC, TRAN, NOISE, or DISTO analysis.

You can use the `.option pwildc` statement, to enable including wildcards in `.PRINT` statements:

```
.option pwildc=1
.PRINT TRAN V(9?t*u)
```

- 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, `t*u` matches `tu`, `t2u`, `tbu`, `t&u`, `tofu`, and `toomuchforyou`.

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

Statement Order

Star-Hspice produces different .sw0 and .tr0 files based on the order of the .print and .dc statements. If no analysis type is given for a .print command, then the analysis type will match the last analysis command found in the netlist before the .print. See examples below.

```
CASE 1
.print v(din) i(mxn18)
.dc vdin 0 5.0 0.05
.tran 1ns 60ns

CASE 2
.dc vdin 0 5.0 0.05
.tran 1ns 60ns
.print v(din) i(mxn18)

CASE 3
.dc vdin 0 5.0 0.05
.print v(din) i(mxn18)
.tran 1ns 60ns
```

- If you are replacing the .print statement with

  ```
  .print TRAN v(din) i(mnx)
  ```

  then all 3 cases have identical .sw0 and .tr0 files.

- If you are replacing the .print statement with

  ```
  .print DC v(din) i(mnx)
  ```

  then the .sw0 and .tr0 files will be different.

Example

```
.PRINT TRAN V(4) I(VIN) PAR(`V(OUT)/V(IN)')
```

This example prints out the results of a transient analysis for the nodal voltage named 4 and the current through the voltage source named VIN. The ratio of the nodal voltage at node “OUT” and node “IN” is also printed.

```
.PRINT AC VM(4,2) VR(7) VP(8,3) II(R1)
```

VM(4,2) specifies that the AC magnitude of the voltage difference (or the difference of the voltage magnitudes, depending on the value of the ACOUT option) between nodes 4 and 2 is printed. VR(7) specifies that the real part of the AC voltage between nodes 7 and ground is printed. VP(8,3) specifies that the

phase of the voltage difference between nodes 8 and 3 (or the difference of the phase of voltage at node 8 and voltage at node 3 depending on the value of ACOUT options) is printed. II(R1) specifies that the imaginary part of the current through R1 is printed.

```
.PRINT AC ZIN YOUT(P) S11(DB) S12(M) Z11(R)
```

The above example specifies that the magnitude of the input impedance, the phase of the output admittance, and several S and Z parameters are to be printed. This statement would accompany a network analysis using the .AC and .NET analysis statements.

```
.PRINT DC V(2) I(VSRC) V(23,17) I1(R1) I1(M1)
```

This example specifies that the DC analysis results are to be printed for several different nodal voltages and currents through the resistor named R1, the voltage source named VSRC, and the drain-to-source current of the MOSFET named M1.

```
.PRINT NOISE INOISE
```

In this example, the equivalent input noise is printed.

```
.PRINT DISTO HD3 SIM2(DB)
```

This example prints the magnitude of the third-order harmonic distortion and the decibel value of the intermodulation distortion sum through the load resistor specified in the .DISTO statement.

```
.PRINT AC INOISE ONOISE VM(OUT) HD3
```

In this statement, specifications of NOISE, DISTO, and AC output variables are included on the same .PRINT statement.

```
.PRINT pj1 = par('p(rd) +p(rs)')
```

This statement prints the value of pj1 with the specified function.

Note: Star-Hspice ignores .PRINT statement references to nonexistent netlist part names and prints those names in a warning message.

.PLOT Statement

The .PLOT statement plots output values of one or more variables in a selected analysis. Each .PLOT statement defines the contents of one plot, which can have 1 to 32 output variables.

When no plot limits are specified, Star-Hspice automatically determines the minimum and maximum values of each output variable being plotted and scales each plot to fit common limits. To cause Star-Hspice to set limits for certain variables, set the plot limits to (0,0) for those variables.

To make Star-Hspice find plot limits for each plot individually, select .OPTION PLIM to create a different axis for each plot variable. The PLIM option is similar to the plot limit algorithm in SPICE2G.6. In the latter case, each plot can have limits different from any other plot. The overlap of two or more traces on a plot is indicated by a number from 2 through 9.

When more than one output variable appears on the same plot, the first variable specified is printed as well as plotted. If a printout of more than one variable is desired, include another .PLOT statement.

The number of .PLOT statements you can specify for each type of analysis is unlimited. Plot width is set by the option CO (columns out). For a CO setting of 80, a 50-column plot is produced. If CO is 132, a 100-column plot is produced.

Syntax

```
.PLOT antype ov1 <(plo1,phi1)> … <ov32>
+ <(plo32,phi32)>
```

where:

antype	Type of analysis for the specified plots. Analysis types are: DC, AC, TRAN, NOISE, or DISTO.
ov1 …	Output variables to plot. These are voltage, current, or element template variables from a DC, AC, TRAN, NOISE, or DISTO analysis. See the following sections for syntax.

plo1,phi1 … Lower and upper plot limits. Each output variable is plotted using the first set of plot limits following the output variable. Output variables following a plot limit should have a new plot limit. For example, to plot all output variables with the same scale, specify one set of plot limits at the end of the PLOT statement. Setting the plot limits to (0,0) causes Star-Hspice to set the plot limits.

Example

In the following example, PAR invokes the plot of the ratio of the collector current and the base current of the transistor Q1.

```
.PLOT DC V(4) V(5) V(1) PAR(`I1(Q1)/I2(Q1)')
.PLOT TRAN V(17,5) (2,5) I(VIN) V(17) (1,9)
.PLOT AC VM(5) VM(31,24) VDB(5) VP(5) INOISE
```

The second of the two examples above uses the VDB output variable to plot the AC analysis results of the node named 5 in decibels. Also, NOISE results may be requested along with the other variables in the AC plot.

```
.PLOT AC ZIN YOUT(P) S11(DB) S12(M) Z11(R)
.PLOT DISTO HD2 HD3(R) SIM2
.PLOT TRAN V(5,3) V(4) (0,5) V(7) (0,10)
.PLOT DC V(1) V(2) (0,0) V(3) V(4) (0,5)
```

In the last example above, Star-Hspice sets the plot limits for V(1) and V(2), while 0 and 5 volts are specified as the plot limits for V(3) and V(4).

.PROBE Statement

The .PROBE statement saves output variables into the interface and graph data files. Star-Hspice usually saves all voltages and supply currents in addition to the output variables. Set .OPTION PROBE to save output variables only. Use the .PROBE statement to specify which quantities are to be printed 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 the .PROBE statement to specify which values you want saved in the output listing.

Syntax

```
.PROBE antype ov1 … <ov32>
```

where:

antype	Type of analysis for the specified plots. Analysis types are: DC, AC, TRAN, NOISE, or DISTO.
ov1 …	Output variables to be plotted. These are voltage, current, or element template variables from a DC, AC, TRAN, NOISE, or DISTO analysis. The limit for the number of output variables in a single .PROBE statement is 32. Additional .PROBE statements may be used to deal with more output variables.

Example

```
.PROBE DC V(4) V(5) V(1) beta = PAR(`I1(Q1)/I2(Q1)')
```

.GRAPH Statement

The .GRAPH statement allows high resolution plotting of simulation results. This statement is similar to the .PLOT statement with the addition of an optional model. When a model is specified, you can add or change graphing properties for the graph. The .GRAPH statement generates a *.gr#* graph data file and sends this file directly to the default high resolution graphical device (specified by PRTDEFAULT in the *meta.cfg* configuration file).

Each .GRAPH statement creates a new *.gr#* file, where # ranges first from 0 to 9, and then from a to z. The maximum number of graph files that can exist is 36. If more than 36 .GRAPH statements are used, the graph files are overwritten starting with the *.gr0* file. To overcome this limitation, the option ALT999 or ALT9999 should be used to extend the number of digits allowed in the file name extension to *.gr###* or *.gr####* (in this case # ranges from 0 to 9).

Note: The .GRAPH statement is not provided in the PC version of Star-Hspice.

Syntax

```
.GRAPH antype <MODEL = mname> <unam1 = > ov1,
+ <unam2 = >ov2, … <unam32 = > ov32 (plo,phi)
```

where:

antype	Type of analysis for the specified plots. Analysis types are: DC, AC, TRAN, NOISE, or DISTO.
mname	Plot model name referenced by the .GRAPH statement. .GRAPH and its plot name allow high resolution plots to be made from Star-Hspice directly.
unam1…	User-defined output names, which correspond to output variables ov1…ov32 (*unam1* to *unam32* respectively), are used as labels instead of output variables for a high resolution graphic output.
ov1 …ov2	Output variables to be printed, 32 maximum. They can be voltage, current, or element template variables from a different type analysis. Algebraic expressions also are used as output variables, but they must be defined inside the PAR( ) statement.
plo, phi	Lower and upper plot limits. Set the plot limits only at the end of the .GRAPH statement.

.MODEL Statement for .GRAPH

This section describes the model statement for .GRAPH.

Syntax

```
.MODEL mname PLOT (pnam1 = val1 pnam2 = val2….)
```

mname	Plot model name referenced by the .GRAPH statements
PLOT	Keyword for a .GRAPH statement model
pnam1 = val1…	Each .GRAPH statement model includes a variety of model parameters. If no model parameters are specified, Star-Hspice takes the default values of the model parameters described in the following table. Pnam*n* is one of the model parameters of a .GRAPH statement, and val*n* is the value of pnam*n*. Val*n* can be a number of parameter.

Example

```
.GRAPH DC cgb = lx18(m1) cgd = lx19(m1) cgs = lx20(m1)

.GRAPH DC MODEL = plotbjt
+ model_ib = i2(q1)    meas_ib = par(ib)
+ model_ic = i1(q1)    meas_ic = par(ic)
+ model_beta = par('i1(q1)/i2(q1)')
+ meas_beta = par('par(ic)/par(ib)')(1e-10,1e-1)

.MODEL plotbjt PLOT MONO = 1 YSCAL = 2 XSCAL = 2
+ XMIN = 1e-8 XMAX = 1e-1
```

Model Parameters

Name (Alias)	Default	Description
FREQ	0.0	Plots symbol frequency. Value 0 suppresses plot symbol generation; a value of n generates a plot symbol every n points.
MONO	0.0	Monotonic option. MONO = 1 automatically resets x-axis if any change in x direction.
TIC	0.0	Shows tick marks
XGRID, *YGRID*	0.0	Setting to 1.0 turns on the axis grid lines
XMIN, *XMA*X	0.0	If XMIN is not equal to XMAX, then XMIN and XMAX determines the x-axis plot limits. If XMIN equals XMAX, or if XMIN and XMAX are not set, then the limits are automatically set. These limits apply to the actual x-axis variable value regardless of the XSCAL type.
XSCAL	1.0	Scale for the x-axis. Two common axis scales are: Linear(LIN) (XSCAL = 1) Logarithm(LOG) (XSCAL = 2)
YMIN, *YMAX*	0.0	If YMIN is not equal to YMAX, then YMIN and YMAX determines the y-axis plot limits. The y-axis limits specified in the .GRAPH statement override YMIN and YMAX in the model. If limits are not specified then they are automatically set. These limits apply to the actual y-axis variable value regardless of the YSCAL type.

Name (Alias)	Default	Description
YSCAL	1.0	Scale for the y-axis. Two common axis scales are: Linear(LIN) (YSCAL = 1) Logarithm(LOG) (YSCAL = 2)

Print Control Options

.OPTION CO for Printout Width

The number of output variables printed on a single line of output is a function of the number of columns, set by the option CO. Typical values are CO = 80 for narrow printouts and CO = 132 for wide printouts. CO = 80 is the default. The maximum number of output variables allowed is 5 per 80-column output and 8 per 132-column output with twelve characters per column. Star-Hspice automatically creates additional print statements and tables for all output variables beyond the number specified by the CO option.

.WIDTH Statement

Syntax

```
.WIDTH OUT = {80 |132}
```

where *OUT* is the output print width

Example

```
.WIDTH OUT = 132 $ SPICE compatible style
.OPTION CO = 132  $ preferred style
```

Permissible values for OUT are 80 and 132. OUT can also be set with option CO.

.OPTION ALT999 or ALT9999 for Output File Name Extension

The output files for postprocessor (from .OPTION POST) or .GRAPH statements have unique extensions *.xx#*, where *xx* is a 2-character text string to denote the output type (see “Specifying Simulation Output” on page Chapter 81 for more information), and # is an alphanumeric character that denotes the .ALTER number of the current simulation. This limits the total number of .ALTER statements in a netlist to 36 before the outputs begin overwriting the current files.

The options ALT999 and ALT9999 extend the output file name extension syntax to *.xx###* and *.xx####*, respectively, where # now represents a numerical character only. This syntax allows for 1000 and 10,000 .ALTERs in the input netlist while maintaining a unique file name for the output files.

.OPTION INGOLD for Printout Numerical Format

Variable values are printed in engineering notation by default:

```
F = 1e-15     M = 1e-3
P = 1e-12     K = 1e3
N = 1e-9      X = 1e6
U = 1e-6      G = 1e9
```

In contrast to the exponential format, the engineering notation provides two to three extra significant digits and aligns columns to facilitate comparison. To obtain output in exponential format, specify INGOLD = 1 or 2 with an .OPTION statement.

INGOLD = 0 (default)	Engineering Format	1.234K 123M
INGOLD = 1	G Format (fixed and exponential)	1.234e+03 .123
INGOLD = 2	E Format (exponential SPICE)	1.234e+03 .123e-1

.OPTION POST for High Resolution Graphics

Use an .OPTION POST statement to use AvanWaves to display high resolution plots of simulation results on a graphics terminal or a high resolution laser printer. Use the .OPTION POST to provide output without specifying other parameters. POST has defaults that supply most parameters with usable data.

POST = 0,1,BINARY	Output format is binary
POST = 2,ASCII	Output format is ascii

.OPTION ACCT Summary of Job Statistics

A detailed accounting report is generated using the ACCT option, where:

.OPTION ACCT	Enables reporting
.OPTION ACCT = 1 (default)	Is the same as ACCT with no argument
.OPTION ACCT = 2	Enables reporting plus matrix statistic reporting

Example

The following output appears at the end of the output listing.

```
****** job statistics summary tnom = 25.000 temp = 25.000
# nodes = 15 # elements = 29 # real*8
mem avail/used = 333333/13454
# diodes =  0 # bjts = 0 # jfets = 0 # mosfets = 24
  analysis   time    # points   tot. iter  conv.iter
  op point   0.24    1          11
  transient  5.45    161        265        103 rev = 1
  pass1      0.08
  readin     0.12
  errchk     0.05
  setup      0.04
  output     0.00
```

The following time statistics are already included in the analysis time:

```
  load       5.22
  solver     0.16
# external nodes = 15 # internal nodes = 0
# branch currents = 5 total matrix size = 20
pivot based and non pivoting solution times
non pivoting: ---- decompose 0.08 solve 0.08
matrix size(109) = initial size(105) + fill(4)
words copied =  111124
total cpu time 6.02 seconds
job started at 11:54:11 21-sep92
job ended   at 11:54:36 21-sep92
```

The definitions for the items in the previous listing follow:

# BJTS	Number of bipolar transistors in the circuit
# ELEMENTS	Total number of elements
# JFETS	Number of JFETs in the circuit
# MOSFETS	Number of MOSFETs in the circuit
# NODES	Total number of nodes
# POINTS	Number of transient points specified by the user on the .TRAN statement. JTRFLG is usually at least 50 unless the option DELMAX is set.
CONV.ITER	Number of points that the simulator needed to take to preserve the accuracy specified by the tolerances
DC	DC operating point analysis time and number of iterations required. The option ITL1 sets the maximum number of iterations.
ERRCHK	Part of the input processing
MEM +	Amount of workspace available and used for the simulation
AVAILUSED	Measured in 64-bit (8-byte) words

OUTPUT	Time required to process all prints and plots
LOAD	Constructs the matrix equation
SOLVER	Solves equations
PASS1	Part of the input processing
READIN	Specifies the input reader that takes the user data file and any additional library files, and generates an internal representation of the information
REV	Number of times the simulator had to cut time (reversals). This is a measure of difficulty.
SETUP	Constructs a sparse matrix pointer system
TOTAL JOB TIME	Total amount of CPU time required to process the simulation. This is not the length of actual (clock) time that was taken, and may differ slightly from run to run, even if the runs are identical.

The ratio of TOT.ITER to CONV.ITER is the best measure of simulator efficiency. The theoretical ratio is 2:1. In this example the ratio was 2.57:1. SPICE generally has a ratio of 3:1 to 7:1.

In transient, the ratio of CONV.ITER to # POINTS is the measure of the number of points evaluated to the number of points printed. If this ratio is greater than about 4, the convergence and time step control tolerances might be too tight for the simulation.

Changing the File Descriptor Limit

A simulation that has a large number of .ALTER statements might fail due to the limit on the number of file descriptors. For example, for a Sun workstation, the default number of file descriptors is 64, and a design with more than 50 .ALTER statements is liable to fail 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 on a Sun workstation, enter the following operating system command before you start the simulation:

```
limit descriptors 128
```

For platforms other than Sun workstations, see your system administrator for help with increasing the number of files you can open concurrently.

Subcircuit Output Printing

The following examples demonstrate how to print or plot voltages of nodes in subcircuit definitions using .PRINT, .PLOT, .PROBE or .GRAPH.

Note: In the following example, you can substitute .PROBE, .PLOT, or .GRAPH for .PRINT.

Example 1

```
.GLOBAL vdd vss
X1 1 2 3 nor2
X2 3 4 5 nor2
.SUBCKT nor2 A B Y
  .PRINT v(B) v(N1) $ Print statement 1
  M1 N1 A vdd vdd pch w = 6u l = 0.8u
  M2 Y B N1 vdd pch w = 6u l = 0.8u
  M3 Y A vss vss vss nch w = 3u l = 0.8u
  M4 Y B vss vss nch w = 3u l = 0.8u
.ENDS
```

Print statement 1 invokes a printout of the voltage on input node B and internal node N1 for every instance of the nor2 subcircuit.

```
.PRINT v(1) v(X1.A) $ Print statement 2
```

The print statement above specifies two ways of printing the voltage on input A of instance X1.

```
.PRINT v(3) v(X1.Y) v(X2.A) $ Print statement 3
```

This print statement specifies three different ways of printing the voltage at output Y of instance X1. (input A of instance X2).

```
.PRINT v(X2.N1) $ Print statement 4
```

The print statement above prints out the voltage on the internal node N1 of instance X2.

```
.PRINT i(X1.M1) $ Print statement 5
```

The print statement above prints out the drain-to-source current through MOSFET M1 in instance X1.

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)
```

The .PRINT statement voltages are:

V(X1.25)	Local node to subcircuit definition YYY, called by subcircuit X1
V(X1.X2.56)	Local node to subcircuit definition ZZZ, called by subcircuit X2, which was called by X1
V(6)	Represents the voltage of node 16 in instance X1 of subcircuit YYY

This example prints analysis results for the voltage at node 56 within the subcircuits X2 and X1. The full path name X1.X2.56 specifies that node 56 is within subcircuit X2 that is within subcircuit X1.

Selecting Simulation Output Parameters

This section discusses how to define specific parameters so that the simulation provides the appropriate output. Define simulation parameters using the .OPTION and .MEASURE statements and specific variable element definitions.

DC and Transient Output Variables

Some types of output variables for DC and transient analyses are:

- Voltage differences between specified nodes (or one specified node and ground)
- Current output for an independent voltage source
- Current output for any element
- Element templates containing the values of user-input variables, state variables, element charges, capacitance currents, capacitances, and derivatives for the various types of devices

The codes that you can use to specify the element templates for output are summarized in “Print Control Options” on page Chapter 814.

Nodal Voltage

Syntax

```
V (n1<,n2>)
```

n1, *n2*	Defines the nodes between which the voltage difference (n1-n2) is to be printed or plotted. When n2 is omitted, the voltage difference between n1 and ground (node 0) is given.

Current: Voltage Sources

Syntax

```
I (Vxxx)
```

where:

Vxxx	Voltage source element name. If an independent power supply is within a subcircuit, its current output is accessed by appending a dot and the subcircuit name to the element name, for example, I(X1.Vxxx).

Example

```
.PLOT TRAN I(VIN)
.PRINT DC I(X1.VSRC)
.PLOT DC I(XSUB.XSUBSUB.VY)
```

Current: Element Branches

Syntax

```
In (Wwww)
```

where:

n	Node position number in the element statement. For example, if the element contains four nodes, I3 denotes the branch current output for the third node; if n is not specified, the first node is assumed.
Wwww	Element name. If the element is within a subcircuit, its current output is accessed by appending a dot and the subcircuit name to the element name, for example, I3(X1.Wwww).

Example

```
I1(R1)
```

This example specifies the current through the first node of resistor R1.

```
I4(X1.M1)
```

The above example specifies the current through the fourth node (the substrate node) of the MOSFET M1, which is defined in subcircuit X1.

```
I2(Q1)
```

The last example specifies the current through the second node (the base node) of the bipolar transistor Q1.

Define each branch circuit by a single element statement. Star-Hspice evaluates branch currents by inserting a zero-volt power supply in series with branch elements.

If Star-Hspice cannot interpret a .PRINT or .PLOT statement containing a branch current, a warning is generated.

Branch current direction for the elements in Figure 8-1 through Figure 8-6 is defined in terms of arrow notation (current direction) and node position number (terminal type).

Figure 8-1: Resistor (node1, node2)

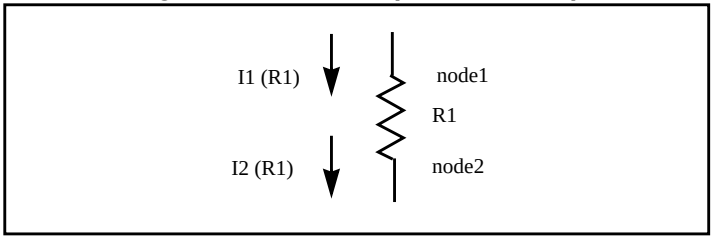


Figure 8-2: Capacitor (node1, node2); Inductor (node 1, node2)

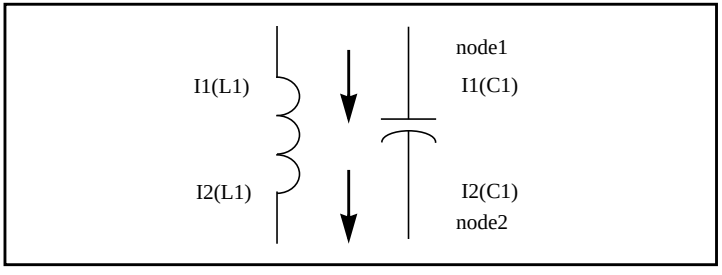


Figure 8-3: Diode (node1, node2)

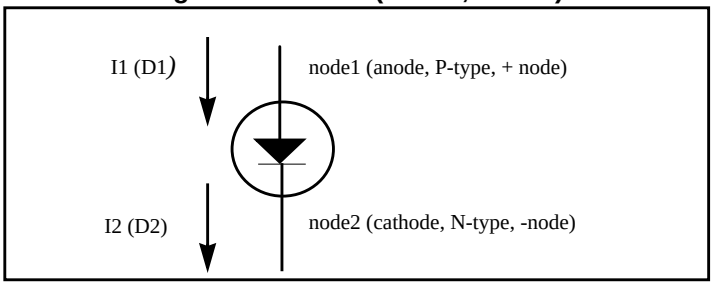


Figure 8-4: JFET (node1, node2, node3) - n-channel

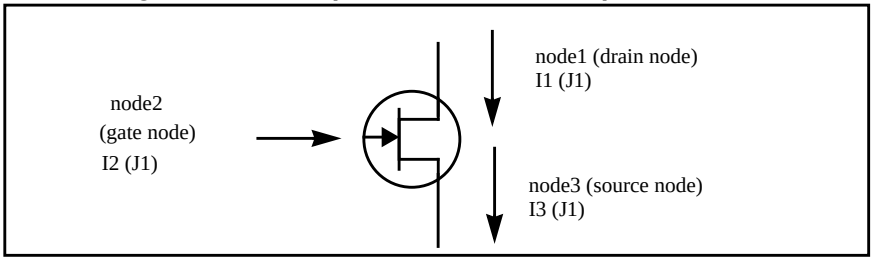


Figure 8-5: BJT (node1, node2, node3, node4) - npn

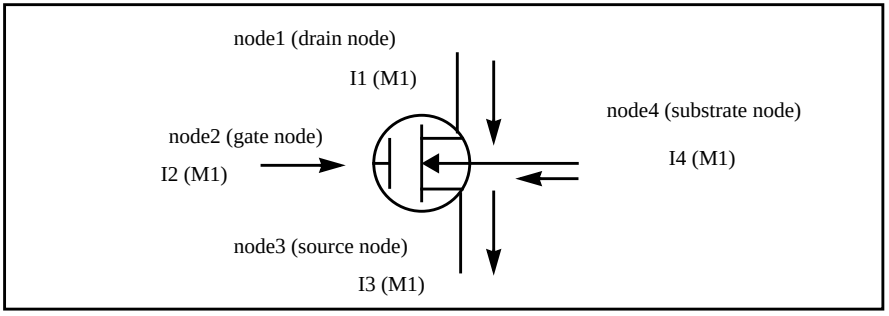
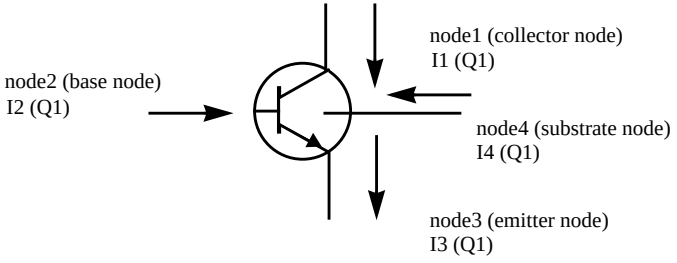


Figure 8-6: MOSFET (node1, node2, node3, node4) - n-channel



Power Output

For power calculations, Star-Hspice computes dissipated or stored power in each passive element (R, L, C), and source (V, I, G, E, F, and H) by multiplying the voltage across an element and its corresponding branch current. However, for semiconductor devices, Star-Hspice calculates only the dissipated power. The power stored in the device junction or parasitic capacitances is excluded from the device power computation. Equations for calculating the power dissipated in different types of devices are shown in the following sections.

Star-Hspice also computes the total power dissipated in the circuit, which is the sum of the power dissipated in the devices, resistors, independent current sources, and all the dependent sources. For hierarchical designs, Star-Hspice computes the power dissipation for each subcircuit as well.

Note: For the total power (dissipated power + stored power), it is possible to add up the power of each independent source (voltage and current sources).

Print or Plot Power

Output the instantaneous element power and the total power dissipation using a .PRINT or .PLOT statement.

Syntax

```
.PRINT <DC | TRAN> P(element_or_subcircuit_name)POWER
```

Power calculation is associated only with transient and DC sweep analyses. The .MEASURE statement can be used to compute the average, rms, minimum, maximum, and peak-to-peak value of the power. The POWER keyword invokes the total power dissipation output.

Example

```
.PRINT TRAN   P(M1)   P(VIN)   P(CLOAD)   POWER
.PRINT TRAN   P(Q1)   P(DIO)   P(J10)     POWER
.PRINT TRAN   POWER   $ Total transient analysis power
* dissipation
.PLOT DC POWER      P(IIN)    P(RLOAD)    P(R1)
.PLOT DC POWER      P(V1)     P(RLOAD)    P(VS)
.PRINT TRAN P(Xf1) P(Xf1.Xh1)
```

Diode Power Dissipation

$$Pd = Vpp' \cdot (Ido + Icap) + Vp'n \cdot Ido$$

Pd	Power dissipated in diode
Ido	DC component of the diode current
Icap	Capacitive component of the diode current
Vp'n	Voltage across the junction
Vpp'	Voltage across the series resistance RS

BJT Power Dissipation

- Vertical

$$Pd = Vc'e' \cdot Ico + Vb'e' \cdot Ibo + Vcc' \cdot Ictot + Vee' \cdot Ietot + Vsc' \cdot Iso - Vcc' \cdot Istot$$

- Lateral

$$Pd = Vc'e' \cdot Ico + Vb'e' \cdot Ibo + Vcc' \cdot Ictot + Vbb' \cdot Ibtot + Vee' \cdot Ietot + Vsb' \cdot Iso - Vbb' \cdot Istot$$

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 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

$$Pd = Vd's' \cdot Ido + Vgd' \cdot Igdo + Vgs' \cdot Igso + Vs's' \cdot (Ido + Igso + Icgs) + Vdd' \cdot (Ido - Igdo - Icgd)$$

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 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

$$Pd = Vd's' \cdot Ido + Vbd' \cdot Ibdo + Vbs' \cdot Ibso + Vs's \cdot (Ido + Ibso + Icbs + Icgs) + Vdd' \cdot (Ido - Ibdo - Icbd - Icgd)$$

Ibdo	DC component of the bulk-drain junction current
Ibso	DC component of the bulk-source junction current
Icbd	Capacitive component of the bulk-drain junction current
Icbs	Capacitive component of the bulk-source junction current
Icgd	Capacitive component of the gate-drain current
Icgs	Capacitive component of the gate-source current
Ido	DC component of the drain current
Pd	Power dissipated in the MOSFET
Vbd'	Voltage across the bulk-drain junction
Vbs'	Voltage across the bulk-source junction
Vd's'	Voltage across the internal drain-source terminals
Vdd'	Voltage across the series drain resistance RD
Vs's	Voltage across the series source resistance RS

AC Analysis Output Variables

Output variables for AC analysis include:

- Voltage differences between specified nodes (or one specified node and ground)
- Current output for an independent voltage source
- Element branch current
- Impedance (Z), admittance (Y), hybrid (H), and scattering (S) parameters
- Input and output impedance and admittance

AC output variable types are listed in Table 8-1. 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 8-1: AC Output Variable Types

Type Symbol	Variable Type
DB	decibel
I	imaginary part
M	magnitude
P	phase
R	real part
T	group delay

Specify real or imaginary parts, magnitude, phase, decibels, and group delay for voltages and currents.

Nodal Voltage

Syntax

```
Vx (n1,<,n2>)
```

where:

x	Specifies the voltage output type (see Table 8-1)
n1, *n2*	Specifies node names. If n2 is omitted, ground (node 0) is assumed.

Example

```
.PLOT AC VM(5) VDB(5) VP(5)
```

The above example plots the magnitude of the AC voltage of node 5 using the output variable VM. The voltage at node 5 is plotted with the VDB output variable. The phase of the nodal voltage at node 5 is plotted with the VP output variable.

Since an AC analysis produces complex results, the values of real or imaginary parts of complex voltages of AC analysis and their magnitude, phase, decibel, and group delay values are calculated using either the SPICE or Star-Hspice method and the control option ACOUT. The default for Star-Hspice is ACOUT = 1. To use the SPICE method, set ACOUT = 0.

The SPICE method is typically used to calculate the nodal vector difference in comparing adjacent nodes in a circuit. It is used to find phase or magnitude across a capacitor, inductor, or semiconductor device.

Use the Star-Hspice method to calculate an inter-stage gain in a circuit (such as an amplifier circuit) and to compare its gain, phase, and magnitude.

The following example defines the AC analysis output variables for the Star-Hspice method and then for the SPICE method.

Example: Star-Hspice Method (ACOUT = 1, Default)

- Real and imaginary:

```
VR(N1,N2) =  REAL [V(N1,0)] - REAL [V(N2,0)]
VI(N1,N2) =  IMAG [V(N1,0)] - IMAG [V(N2,0)]
```

- Magnitude:

$$VM(N1,0) = [VR(N1,0)^2 + VI(N1,0)^2]^{0.5}$$
$$VM(N2,0) = [VR(N2,0)^2 + VI(N2,0)^2]^{0.5}$$
```
VM(N1,N2) =  VM(N1,0) - VM(N2,0)
```

- Phase:

```
VP(N1,0) =  ARCTAN[VI(N1,0)/VR(N1,0)]
VP(N2,0) =  ARCTAN[VI(N2,0)/VR(N2,0)]
VP(N1,N2) =  VP(N1,0) - VP(N2,0)
```

- Decibel:

$$VDB(N1,N2) = 20 \cdot LOG10(VM(N1,0)/VM(N2,0))$$

Example: SPICE Method (ACOUT = 0)

- Real and imaginary:

```
VR(N1,N2) =  REAL [V(N1,0) - V(N2,0)]
VI(N1,N2) =  IMAG [V(N1,0) - V(N2,0)]
```

- Magnitude:

$$VM(N1,N2) = [VR(N1,N2)^2+VI(N1,N2)^2]^{0.5}$$

- Phase:

```
VP(N1,N2) =  ARCTAN[VI(N1,N2)/VR(N1,N2)]
```

- Decibel:

$$VDB(N1,N2) = 20 \cdot LOG10[VM(N1,N2)]$$

Current: Independent Voltage Sources

Syntax

```
Iz (Vxxx)
```

where:

z Current output type (see Table 8-1)

Vxxx Voltage source element name. If an independent power supply is within a subcircuit, to access its current output, append a dot and the subcircuit name to the element name, for example, IM(X1.Vxxx).

Example

```
.PLOT AC IR(V1) IM(VN2B) IP(X1.X2.VSRC)
```

Current: Element Branches

Syntax

```
Izn (Wwww)
```

where:

z Current output type (see Table 8-1)

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, to access its current output, append a dot and the subcircuit name to the element name, for example, IM3(X1.Wwww).

Example

```
.PRINT AC IP1(Q5) IM1(Q5) IDB4(X1.M1)
```

If you use the form In(Xxxx) for AC analysis output, the magnitude IMn(Xxxx) is the value printed.

Group Time Delay

The group time delay, TD, is associated with AC analysis and is defined as the negative derivative of phase, in radians, with respect to radian frequency. In Star-Hspice, the difference method is used to compute TD, as follows

$$TD = -\frac{1}{360} \cdot \frac{(phase2 - phase1)}{(f2 - f1)}$$

where phase1 and phase2 are the phases, in degrees, of the specified signal at the frequencies f1 and f2, in Hertz.

Syntax

```
.PRINT AC VT(10) VT(2,25) IT(RL)
.PLOT AC IT1(Q1) IT3(M15) IT(D1)
```

Note: Because there is discontinuity in phase each 360°, the same discontinuity is seen in TD, 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
.PLOT AC     VT(3)    (0,4U)    VP(3)
.END
```

Network

Syntax

```
Xij (z), ZIN(z), ZOUT(z), YIN(z), YOUT(z)
```

where:

X	Specifies Z for impedance, Y for admittance, H for hybrid, or S for scattering parameters
ij	i and j can be 1 or 2. They identify which matrix parameter is printed.
z	Output type (see Table 8-1). If z is omitted, the magnitude of the output variable is printed.
ZIN	Input impedance. For a one port network ZIN, Z11, and H11 are the same
ZOUT	Output impedance
YIN	Input admittance. For a one-port network, YIN and Y11 are the same.
YOUT	Output admittance

Example

```
.PRINT    AC   Z11(R)  Z12(R)  Y21(I)   Y22      S11   S11(DB)
.PRINT    AC   ZIN(R)  ZIN(I)  YOUT(M)  YOUT(P)  H11(M)
.PLOT     AC   S22(M)  S22(P)  S21(R)   H21(P)   H12(R)
```

Noise and Distortion

This section describes the variables used for noise and distortion analysis.

Syntax

```
ovar <(z)>
```

where:

ovar	Noise and distortion analysis parameter. It can be either ONOISE (output noise), or INOISE (equivalent input noise) or any of the distortion analysis parameters (HD2, HD3, SIM2, DIM2, DIM3).
z	Output type (only for distortion). If z is omitted, the magnitude of the output variable is output.

Example

```
.PRINT DISTO HD2(M) HD2(DB)
```

Prints the magnitude and decibel values of the second harmonic distortion component through the load resistor specified in the .DISTO statement (not shown).

```
.PLOT NOISE INOISE ONOISE
```

Note: The noise and distortion output variable may be specified along with other AC output variables in the .PRINT AC or .PLOT AC statements.

Element Template Output

Element templates are used in .PRINT, .PLOT, .PROBE, and .GRAPH statements for output of user-input parameters, state variables, stored charges, capacitor currents, capacitances, and derivatives of variables. The Star-Hspice element templates are listed at the end of this chapter.

Syntax

```
Elname:Property
```

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)
```

or

```
LXnn(Elname)
```

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, given in the tables in this section
Elname	Name of the element

Example

```
.PLOT TRAN V(1,12) I(X2.VSIN) I2(Q3) DI01:GD
.PRINT TRAN X2.M1:CGGBO M1:CGDBO X2.M1:CGSBO
```

Specifying User-Defined Analysis (.MEASURE)

Use the .MEASURE statement to modify information and define the results of successive simulations.

The .MEASURE statement prints user-defined electrical specifications of a circuit and is used extensively in optimization. The specifications include propagation, delay, rise time, fall time, peak-to-peak voltage, minimum and maximum voltage over a specified period, and a number of other user-defined variables. With either the error function or GOAL parameter, .MEASURE is also used extensively for optimization of circuit component values and curve fitting measured data to model parameters.

Measurement results are computed based on postprocessing output. Using the INTERP option to reduce the size of postprocessing output may lead to interpolation error in measurement results. See “Input and Output” on page -50 for more information on the INTERP option.

The .MEASURE statement has several different formats, depending on the application. You can use it for either DC sweep, AC, or transient analysis.

Fundamental measurement modes 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, Star-Hspice writes 0.0e0 in the *.mt#* file as the .MEASURE result, and writes FAILED in the output listing file. Use the MEASFAIL option to write results to the .mt#, .ms#, or .ma# files. See “Input and Output” on page -50 for information about the MEASFAIL option.

To control the output variables, listed in the .measure statement, use the .putmeas option. See “Input and Output Options” on page -60 for additional information.

Measure Parameter Types

Measurement parameter results produced by .PARAM statements in .SUBCKT blocks cannot be used outside the subcircuit. That means measurement parameters defined in .SUBCKT statements cannot be passed as bottom-up parameters in hierarchical designs.

Measurement parameter names cannot conflict with standard parameter names. Star-Hspice issues an error message if it encounters a measurement parameter with the same name as a standard parameter definition.

To prevent parameter values given in .MEASURE statements from overwriting parameter assignments in other statements, Star-Hspice keeps track of parameter types. If the same parameter name is used in both a .MEASURE statement and a .PARAM statement at the same hierarchical level, Star-Hspice terminates with an error. No error occurs if the parameter assignments are at different hierarchical levels. PRINT statements that occur at different levels do not print hierarchical information for the parameter name headings.

The following example illustrates how Star-Hspice handles .MEASURE statement parameters.

```
...
.MEASURE tran length TRIG v(clk) VAL = 1.4
+ TD = 11ns RISE = 1 TARGv(neq) VAL = 1.4 TD = 11ns
+ RISE = 1
.SUBCKT path out in width = 0.9u length = 600u
+ rm1 in m1 m2mg w = 'width' l = 'length/6'
...
.ENDS
```

In the above listing, the ‘length’ in the resistor statement:

```
rm1 in m1 m2mg w = 'width' l = 'length/6'
```

does not inherit its value from the 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
```

instead of a value derived from the measured value in transient analysis.

.MEASURE Statement: Rise, Fall, and Delay

This format is used to measure independent-variable (time, frequency, or any parameter or temperature) differential measurements such as rise time, fall time, slew rate, and any measurement that requires the determination of independent variable values. The format specifies substatements TRIG and TARG. These two statements specify the beginning and ending of a voltage or current amplitude measurement.

The rise, fall, and delay measurement mode computes the time, voltage, or frequency between a trigger value and a target value. Examples for transient analysis include rise/fall time, propagation delay, and slew rate measurement. Applications for AC analysis are the measurement of the bandwidth of an amplifier or the frequency at which a certain gain is achieved.

Syntax

```
.MEASURE <DC|AC|TRAN> result TRIG … TARG …
+ <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

where:

MEASURE Specifies measurements. You can abbreviate to MEAS.

result Name that is associated with the measured value in the Star-Hspice output. The item measured is the independent variable beginning at the trigger and ending at the target: for transient analysis it is time; for AC analysis it is frequency; for DC analysis it is the DC sweep variable. If the target is reached before the trigger is activated, the resulting value is negative.

Note: The terms “DC”, “TRAN”, and “AC” are illegal for *result* name.

TRIG…, TARG … Identifies the beginning of trigger and target specifications, respectively.

<DC|AC|TRAN> Specifies the analysis type of the measurement. If omitted, the last analysis mode requested is assumed.

GOAL Specifies the desired measure value in optimization. The error is calculated by $ERRfun = (GOAL - result)/GOAL$.

MINVAL If the absolute value of GOAL is less than MINVAL, the GOAL value is replaced by MINVAL in the denominator of the ERRfun expression. Default = 1.0e-12.

WEIGHT The calculated error is multiplied by the weight value. Used in optimization. Default = 1.0.

Trigger

```
TRIG trig_var VAL = trig_val <TD = time_delay>
+ <CROSS = c> <RISE = r> <FALL = f>
```

or

```
TRIG AT = val
```

Target

```
TARG targ_var VAL = targ_val <TD = time_delay>
+ <CROSS = c | LAST> <RISE = r | LAST> <FALL = f | LAST>
```

where:

TRIG	Indicates the beginning of the trigger specification
trig_val	Value of *trig_var* at which the counter for crossing, rises, or falls is incremented by one
trig_var	Specifies the name of the output variable, which determines the logical beginning of measurement. If the target is reached before the trigger is activated, .MEASURE reports a negative value.
TARG	Indicates the beginning of the target signal specification
targ_val	Specifies the value of the *targ_var* at which the counter for crossing, rises, or falls is incremented by one
targ_var	Name of the output variable whose propagation delay is determined with respect to the *trig_var*
time_delay	Amount of simulation time that must elapse before the measurement is enabled. The number of crossings, rises, or falls is counted only after *time_delay* value. The default trigger delay is zero.

CROSS = c
RISE = r
FALL = f

The numbers indicate which occurrence of a CROSS, FALL, or RISE event causes Star-Hspice to measure.

- RISE = r, the WHEN condition is met and measurement occurs after the designated signal rises *r* rise times.
- FALL = f, measurement occurs when the designated signal has fallen *f* fall times.
- A crossing is either a rise or a fall, so for CROSS = c, measurement occurs when the designated signal has achieved a total of *c* crossing times, as a result of either rising or falling.
- For TARG, the LAST keyword. specifies the last event.

LAST

Star-Hspice measures when the last CROSS, FALL, or RISE event occurs.

- CROSS = LAST, measurement occurs the last time the WHEN condition is true for a rising or falling signal.
- FALL = LAST, measurement occurs the last time the WHEN condition is true for a falling signal.
- RISE = LAST, measurement occurs the last time the WHEN condition is true for a rising signal.
- LAST is a reserved word; you cannot use it as a parameter name in the above .MEASURE statements.

AT = val

Special case for trigger specification. “val” is the time for TRAN analysis, the frequency for AC analysis, or the parameter for DC analysis, at which measurement starts.

Example

```
.MEASURE TRAN tdlay TRIG V(1) VAL = 2.5 TD = 10n
+ RISE = 2 TARG V(2) VAL = 2.5 FALL = 2
```

This example measures propagation delay between nodes 1 and 2 for a transient analysis. The delay is measured from the second rising edge of the voltage at node 1, to the second falling edge of node 2. The measurement begins when the second rising voltage at node 1 is 2.5 V, and ends when the second falling voltage at node 2 reaches 2.5 V. The TD = 10n parameter does not count the crossings until after 10 ns has elapsed. The results are printed as tdlay = <value>.

```
.MEASURE TRAN riset TRIG I(Q1) VAL = 0.5m RISE = 3
+ TARG I(Q1) VAL = 4.5m RISE = 3

.MEASURE pwidth TRIG AT = 10n TARG V(IN) VAL = 2.5
+ CROSS = 3
```

In the last example, TRIG. AT = 10n starts measuring time at t = 10 ns in the transient analysis. The TARG parameters end time measurement when V(IN) = 2.5 V on the third crossing. The *pwidth* variable is the printed output variable.

Note: If the .TRAN statement is used in conjunction with a .MEASURE statement, using a nonzero START time in the .TRAN statement can result in incorrect .MEASURE results. Do not use nonzero START times in .TRAN statements when .MEASURE is also being used.

FIND and WHEN Functions

The FIND and WHEN functions specify to measure any independent variables (time, frequency, parameter), any dependent variables (voltage or current, for example), or the derivative of any dependent variables, when some specific event occurs. You can use these measure statements in unity gain frequency or phase measurements, as well as to measure the time, frequency, or any parameter value when two signals cross each other, or 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 LAST for 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 8-5.

Syntax

```
.MEASURE <DC|TRAN| AC> result WHEN out_var = val <TD = val>
+ < RISE = r | LAST > < FALL = f | LAST > < CROSS = c | LAST >
+ <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

or

```
.MEASURE <DC|TRAN|AC> result WHEN out_var1 = out_var2
+ < TD = val > < RISE = r | LAST > < FALL = f | LAST >
+ < CROSS = c| LAST > <GOAL = val> <MINVAL = val>
+ <WEIGHT = val>
```

or

```
.MEASURE <DC|TRAN|AC> result FIND out_var1 WHEN out_var2 = val
+ < TD = val > < RISE = r | LAST > < FALL = f | LAST >
+ < CROSS = c| LAST > <GOAL = val> <MINVAL = val>
+ <WEIGHT = val>
```

or

```
.MEASURE <DC|TRAN|AC> result FIND out_var1
+ WHEN out_var2 = out_var3 <TD = val > < RISE = r | LAST >
+ < FALL = f | LAST > <CROSS = c | LAST> <GOAL = val>
+ <MINVAL = val> <WEIGHT = val>
```

or

```
.MEASURE <DC|TRAN|AC> result FIND out_var1 AT = val
+ <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

Parameter Definitions

CROSS = c *RISE = r* *FALL = f*	The numbers indicate which occurrence of a CROSS, FALL, or RISE event causes a measurement to be performed. For RISE = r, the WHEN condition is met and measurement is performed when the designated signal has risen *r* rise times. For FALL = f, measurement is performed when the designated signal has fallen *f* fall times. A crossing is either a rise or a fall, so for CROSS = c, measurement is performed when the designated signal has achieved a total of *c* crossing times, as a result of either rising or falling.
<DC|AC|TRAN>	Specifies the analysis type of the measurement. If omitted, the last analysis type requested is assumed.
FIND	Selects the FIND function
GOAL	Specifies the desired .MEASURE value. It is used in optimization. The error is calculated by $ERRfun = (GOAL - result)/GOAL$.

LAST	Measurement is performed when the last CROSS, FALL, or RISE event occurs. For CROSS = LAST, measurement is performed the last time the WHEN condition is true for either a rising or falling signal. For FALL = LAST, measurement is performed the last time the WHEN condition is true for a falling signal. For RISE = LAST, measurement is performed the last time the WHEN condition is true for a rising signal. LAST is a reserved word and cannot be chosen as a parameter name in the above .MEASURE statements.
MINVAL	If the absolute value of GOAL is less than MINVAL, the GOAL value is replaced by MINVAL in the denominator of the ERRfun expression. Default = 1.0e-12.
out_var(1,2,3)	Variables used to establish conditions at which measurement is to take place
result	Name associated with the measured value in the Star-Hspice output
TD	Identifies the time at which measurement is to start
WEIGHT	Calculated error is multiplied by the weight value. Default = 1.0.
WHEN	Selects the WHEN function

Equation Evaluation

Use this statement to evaluate an equation that is a function of the results of previous .MEASURE statements. The equation must not be a function of node voltages or branch currents.

Syntax

```
.MEASURE <DC|TRAN|AC> result PARAM = 'equation'
+ <GOAL = val> <MINVAL = val>
```

Average, RMS, MIN, MAX, INTEG, and Peak-To-Peak

The average (AVG), RMS, MIN, MAX, and peak-to-peak (PP) measurement modes report statistical functions of the output variable rather than the analysis value. Average calculates the area under the output variable divided by the periods of interest. RMS takes the square root of the area under the output variable square divided 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.

Syntax

```
.MEASURE <DC|AC|TRAN> result func out_var <FROM = val>
+ <TO = val> <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

where:

<DC|AC|TRAN>	Specifies the analysis type of the measurement. If omitted, the last analysis mode requested is assumed.
FROM	Specifies the initial value for the "func" calculation. For transient analysis, value is in units of time.
TO	Specifies the end of the "func" calculation.
GOAL	Specifies the desired .MEASURE value. It is used in optimization. The error is calculated by $ERRfun = (GOAL - result)/GOAL$

MINVAL	If the absolute value of GOAL is less than MINVAL, the GOAL value is replaced by MINVAL in the denominator of the ERRfun expression. Default = 1.0e-12.
func	Indicates the type of the measure statement, one of the following: ■ AVG (average): Calculates the area under the *out_var* divided by the periods of interest ■ MAX (maximum): Reports the maximum value of the *out_var* over the specified interval ■ MIN (minimum): Reports the minimum value of the *out_var* over the specified interval ■ PP (peak-to-peak): Reports the maximum value minus the minimum value of the *out_var* over the specified interval ■ RMS (root mean squared): Calculates the square root of the area under the out_var^2 curve divided by the period of interest
result	Name that is associated with the measured value in the Star-Hspice output. The value is a function of the variable specified (*out_var*) and func.
out_var	Name of any output variable whose function (“func”) is to be measured in the simulation.
WEIGHT	Multiplies he calculated error by the weight value. Default = 1.0.

Example

```
.MEAS TRAN avgval AVG V(10) FROM = 10ns TO = 55ns
```

The example above calculates the average nodal voltage value for node 10 during the transient sweep from the time 10 ns to 55 ns and prints out the result as “avgval”.

```
.MEAS  TRAN  MAXVAL  MAX  V(1,2)
FROM = 15ns  TO = 100ns
```

The example above finds the maximum voltage difference between nodes 1 and 2 for the time period from 15 ns to 100 ns.

```
.MEAS  TRAN  MINVAL  MIN  V(1,2)
FROM = 15ns  TO = 100ns
.MEAS  TRAN  P2PVAL  PP   I(M1)  FROM = 10ns  TO = 100ns
```

INTEGRAL Function

The INTEGRAL function provides the integral of an output variable over a specified period.

Syntax

```
.MEASURE <DC|AC|TRAN> result INTEGRAL out_var
+ <FROM = val> <TO = val> <GOAL = val> <MINVAL = val>
+ <WEIGHT = val>
```

The same syntax used for the average (AVG), RMS, MIN, MAX, and peak-to-peak (PP) measurement mode is used for the INTEGRAL function with *func* to be defined as INTEGRAL (INTEG).

Example

The following example calculates the integral of I(cload) from 10 ns to 100 ns.

```
.MEAS  TRAN charge INTEG I(cload)
FROM = 10ns  TO = 100ns
```

DERIVATIVE Function

The DERIVATIVE function provides the derivative of an output variable at a given time or frequency or for any sweep variable, depending on the type of analysis. It also provides the derivative of a specified output variable when some specific event occurs.

Syntax

```
.MEASURE <DC|AC|TRAN> result DERIVATIVE out_var
+ AT = val <GOAL = val> <MINVAL = val> <WEIGHT = val>
```

or

```
.MEASURE <DC|AC|TRAN> result DERIVATIVE out_var
+ WHEN var2 = val <RISE = r | LAST> <FALL = f | LAST>
+ <CROSS = c | LAST> <TD = tdval> <GOAL = goalval>
+ <MINVAL = minval> <WEIGHT = weightval>
```

or

```
.MEASURE <DC|AC|TRAN> result DERIVATIVE out_var
+ WHEN var2 = var3 <RISE = r | LAST> <FALL = f | LAST>
+ <CROSS = c | LAST> <TD = tdval> <GOAL = goalval>
+ <MINVAL = minval> <WEIGHT = weightval>
```

where:

AT = val	Value of *out_var* at which the derivative is to be found
CROSS = c *RISE = r* *FALL = f*	The numbers indicate which occurrence of a CROSS, FALL, or RISE event causes a measurement to be performed. For RISE = r, the WHEN condition is met and measurement is performed when the designated signal has risen *r* rise times. For FALL = f, measurement is performed when the designated signal has fallen *f* fall times. A crossing is either a rise or a fall, so for CROSS = c, measurement is performed when the designated signal has achieved a total of *c* crossing times, as a result of either rising or falling.

<DC|AC|TRAN>	Specifies the analysis type measured. If omitted, the last analysis mode requested is assumed.
DERIVATIVE	Selects the derivative function. You can abbreviate to DERIV.
GOAL	Specifies the desired .MEASURE value. It is used in optimization. The error is calculated by $ERRfun = (GOAL - result)/GOAL$.
LAST	Measures when the last CROSS, FALL, or RISE event occurs. ■ CROSS = LAST, measures the last time the WHEN condition is true for a rising or falling signal. ■ FALL = LAST, measures the last time the WHEN condition is true for a falling signal. ■ RISE = LAST, measures the last time the WHEN condition is true for a rising signal. ■ LAST is a reserved word; you cannot use it as a parameter name in the above .MEASURE statements.
MINVAL	If the absolute value of GOAL is less than MINVAL, the GOAL value is replaced by MINVAL in the denominator of the ERRfun expression. Default = 1.0e-12.
out_var	Variable for which the derivative is to be found
result	Name associated with the measured value in the Star-Hspice output
TD	Identifies the time at which measurement is to start
var(2,3)	Variables used to establish conditions at which measurement is to take place
WEIGHT	The calculated error between result and GOAL is multiplied by the weight value. Default = 1.0.
WHEN	Selects the WHEN function

Example

The following example calculates the derivative of V(out) at 25 ns:

```
.MEAS TRAN slew rate DERIV V(out) AT = 25ns
```

The following example calculates the derivative of v(1) when v(1) is equal to 0.9*vdd:

```
.MEAS TRAN slew DERIV v(1) WHEN v(1) = '0.90*vdd'
```

The following example calculates the derivative of VP(output)/360.0 when the frequency is 10 kHz:.

```
.MEAS AC delay DERIV 'VP(output)/360.0' AT = 10khz
```

ERROR Function

The relative error function reports the relative difference of two output variables. This format is often used in optimization and curve fitting of measured data. The relative error format specifies the variable to be measured and calculated from the .PARAM variables. The relative error between the two is calculated using the ERR, ERR1, ERR2, or ERR3 function. With this format, you can specify a group of parameters to vary to match the calculated value and the measured data.

Syntax

```
.MEASURE <DC|AC|TRAN> result ERRfun meas_var calc_var
+ <MINVAL = val> < IGNORE | YMIN = val> <YMAX = val>
+ <WEIGHT = val> <FROM = val> <TO = val>
```

where:

<DC|AC|TRAN>	Specifies the analysis type of the measurement. If omitted, the last analysis mode requested is assumed.
result	Name associated with the measured result in the output
ERRfun	ERRfun indicates which error function to use: ERR, ERR1, ERR2, or ERR3.
meas_var	Name of any output variable or parameter in the data statement. M denotes the *meas_var* in the error equation.

calc_var	Name of the simulated output variable or parameter in the .MEASURE statement to be compared with *meas_var.* C denotes the *calc_var* in the error equation.
IGNOR|YMIN	If the absolute value of *meas_var* is less than IGNOR value, then this point is not considered in the ERRfun calculation. Default = 1.0e-15.
FROM	Specifies the beginning of the ERRfun calculation. For transient analysis, the from value is in units of time. Defaults to the first value of the sweep variable.
WEIGHT	The calculated error is multiplied by the weight value. Default = 1.0.
YMAX	If the absolute value of *meas_var* is greater than the YMAX value, then this point is not considered in the ERRfun calculation. Default = 1.0e+15.
TO	Specifies the end of the ERRfun calculation. Defaults to the last value of the sweep variable.
MINVAL	If the absolute value of *meas_var* is less than MINVAL, the *meas_var* value is replaced by MINVAL in the denominator of the ERRfun expression. Default = 1.0e-12.

Error Equations

ERR

ERR sums the squares of (M-C)/max (M, MINVAL) for each point, divides by the number of points, and then takes the square root of the result. M (meas_var) and C (calc_var) are the measured and calculated values of the device or circuit response, respectively. NPTS is the number of data points.

$$ERR = \left[\frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} \left(\frac{\mathrm{M}_i - \mathrm{C}_i}{\max(MINVAL, M_i)}\right)^2\right]^{1/2}$$

ERR1

ERR1 computes the relative error at each point. For NPTS points, there are NPTS ERR1 error function calculations. For device characterization, the ERR1 approach has been found to be more efficient than the other error functions (ERR, ERR2, ERR3).

$$ERR1_i = \frac{M_i - C_i}{\max(MINVAL, M_i)} \mathrm{i} = 1,\mathrm{NPTS}$$

Star-Hspice does not print out each calculated ERR1 value. When the ERR1 option is set, it returns an ERR value calculated 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, there are NPTS error function calls.

$$ERR2_i = \left|\frac{M_i - C_i}{\max(MINVAL, M_i)}\right|, = 1,\mathrm{NPTS}$$

The returned value printed for ERR2 is:

$$ERR = \frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} ERR2_i$$

ERR3

$$ERR3_i = \frac{\pm \log\left|\frac{M_i}{C_i}\right|}{\left|\log[\max(MINVAL, |M_i|)]\right|} = 1,\text{NPTS}$$

The + and - signs correspond to a positive and negative M/C ratio, respectively.

Note: If the measured value M is less than MINVAL, the MINVAL is used instead. Also, if the absolute value of M is less than the IGNOR | YMIN value or greater than the YMAX value, then this point is not considered in the error calculation.

.DOUT Statement: Expected State of Digital Output Signal

The digital output (`.DOUT`) statement in Star-Hspice specifies the expected final state of an output signal.

During simulation, Star-Hspice compares the simulated results with the expected output vector. If the states are different, Star-Hspice reports an error.

Syntax

The `.DOUT` statement can use either of two syntaxes. In both syntaxes, the *time* and *state* parameters describe the expected output of the *nd* node.

- The first syntax specifies a single threshold voltage, *VTH*. 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 allows 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 Star-Hspice processes only VTH, and ignores VHI and VLO.

Element Template Listings

Table 8-2: Resistor

Name	Alias	Description
G	LV1	Conductance at analysis temperature
R	LV2	Resistance at reference temperature
TC1	LV3	First temperature coefficient
TC2	LV4	Second temperature coefficient

Table 8-3: Capacitor

Name	Alias	Description
CEFF	LV1	Computed effective capacitance
IC	LV2	Initial condition
Q	LX0	Charge stored in capacitor
CURR	LX1	Current flowing through capacitor
VOLT	LX2	Voltage across capacitor
–	LX3	Capacitance (not used after Star-Hspice release 95.3)

Table 8-4: Inductor

Name	Alias	Description
LEFF	LV1	Computed effective inductance
IC	LV2	Initial condition

Table 8-4: Inductor *(Continued)*

Name	Alias	Description
FLUX	LX0	Flux in the inductor
VOLT	LX1	Voltage across inductor
CURR	LX2	Current flowing through inductor
–	LX4	Inductance (not used after Star-Hspice release 95.3)

Table 8-5: Mutual Inductor

Name	Alias	Description
K	LV1	Mutual inductance

Table 8-6: Voltage-Controlled Current Source

Name	Alias	Description
CURR	LX0	Current through the source, if VCCS
R	LX0	Resistance value, if VCR
C	LX0	Capacitance value, if VCCAP
CV	LX1	Controlling voltage
CQ	LX1	Capacitance charge, if VCCAP
DI	LX2	Derivative of source current relative to control voltage
ICAP	LX2	Capacitance current, if VCCAP
VCAP	LX3	Voltage across capacitance, if VCCAP

Table 8-7: Voltage-Controlled Voltage Source

Name	Alias	Description
VOLT	LX0	Source voltage
CURR	LX1	Current through source
CV	LX2	Controlling voltage
DV	LX3	Derivative of source voltage relative to control current

Table 8-8: Current-Controlled Current Source

Name	Alias	Description
CURR	LX0	Current through source
CI	LX1	Controlling current
DI	LX2	Derivative of source current relative to control current

Table 8-9: Current-Controlled Voltage Source

Name	Alias	Description
VOLT	LX0	Source voltage
CURR	LX1	Source current
CI	LX2	Controlling current
DV	LX3	Derivative of source voltage relative to control current

Table 8-10: Independent Voltage Source

Name	Alias	Description
VOLT	LV1	DC/transient voltage

Table 8-10: Independent Voltage Source *(Continued)*

Name	Alias	Description
VOLTM	LV2	AC voltage magnitude
VOLTP	LV3	AC voltage phase

Table 8-11: Independent Current Source

Name	Alias	Description
CURR	LV1	DC/transient current
CURRM	LV2	AC current magnitude
CURRP	LV3	AC current phase

Table 8-12: Diode

Name	Alias	Description
AREA	LV1	Diode area factor
AREAX	LV23	Area after scaling
IC	LV2	Initial voltage across diode
VD	LX0	Voltage across diode (VD), excluding RS (series resistance)
IDC	LX1	DC current through diode (ID), excluding RS. Total diode current is the sum of IDC and ICAP
GD	LX2	Equivalent conductance (GD)
QD	LX3	Charge of diode capacitor (QD)
ICAP	LX4	Current through diode capacitor. Total diode current is the sum of IDC and ICAP.

Table 8-12: Diode *(Continued)*

Name	Alias	Description
C	LX5	Total diode capacitance
PID	LX7	Photo current in diode

Table 8-13: BJT *(Sheet 1 of 3)*

Name	Alias	Description
AREA	LV1	Area factor
ICVBE	LV2	Initial condition for base-emitter voltage (VBE)
ICVCE	LV3	Initial condition for collector-emitter voltage (VCE)
MULT	LV4	Number of multiple BJTs
FT	LV5	FT (Unity gain bandwidth)
ISUB	LV6	Substrate current
GSUB	LV7	Substrate conductance
LOGIC	LV8	LOG 10 (IC)
LOGIB	LV9	LOG 10 (IB)
BETA	LV10	BETA
LOGBETAI	LV11	LOG 10 (BETA) current
ICTOL	LV12	Collector current tolerance
IBTOL	LV13	Base current tolerance
RB	LV14	Base resistance
GRE	LV15	Emitter conductance, 1/RE
GRC	LV16	Collector conductance, 1/RC

Table 8-13: BJT *(Sheet 2 of 3)*

Name	Alias	Description
PIBC	LV18	Photo current, base-collector
PIBE	LV19	Photo current, base-emitter
VBE	LX0	VBE
VBC	LX1	Base-collector voltage (VBC)
CCO	LX2	Collector current (CCO)
CBO	LX3	Base current (CBO)
GPI	LX4	g_π = ib / vbe, constant vbc
GU	LX5	g_μ = ib / vbc, constant vbe
GM	LX6	g_m = ic / vbe+ ic / vbe, constant vce
G0	LX7	g_0 = ic / 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)

Table 8-13: BJT *(Sheet 3 of 3)*

Name	Alias	Description
–	LX18	Base-collector conductance (GEQCBO) (not used in Star-Hspice releases after 95.3)
CAP_BE	LX19	cbe capacitance (C_Π)
CAP_IBC	LX20	cbc internal base-collector capacitance (C_μ)
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	(TF*IBE) / vbc
VSUB	LX24	Substrate voltage

Table 8-14: JFET

Name	Alias	Description
AREA	LV1	JFET area factor
VDS	LV2	Initial condition for drain-source voltage
VGS	LV3	Initial condition for gate-source voltage
PIGD	LV16	Photo current, gate-drain in JFET
PIGS	LV17	Photo current, gate-source in JFET
VGS	LX0	VGS
VGD	LX1	Gate-drain voltage (VGD)
CGSO	LX2	Gate-to-source (CGSO)
CDO	LX3	Drain current (CDO)

Table 8-14: JFET *(Continued)*

Name	Alias	Description
CGDO	LX4	Gate-to-drain current (CGDO)
GMO	LX5	Transconductance (GMO)
GDSO	LX6	Drain-source transconductance (GDSO)
GGSO	LX7	Gate-source transconductance (GGSO)
GGDO	LX8	Gate-drain transconductance (GGDO)
QGS	LX9	Gate-source charge (QGS)
CQGS	LX10	Gate-source charge current (CQGS)
QGD	LX11	Gate-drain charge (QGD)
CQGD	LX12	Gate-drain charge current (CQGD)
CAP_GS	LX13	Gate-source capacitance
CAP_GD	LX14	Gate-drain capacitance
–	LX15	Body-source voltage (not used after Star-Hspice release 95.3)
QDS	LX16	Drain-source charge (QDS)
CQDS	LX17	Drain-source charge current (CQDS)
GMBS	LX18	Drain-body (backgate) transconductance (GMBS)

Table 8-15: MOSFET *(Sheet 1 of 5)*

Name	Alias	Description
L	LV1	Channel length (L)
W	LV2	Channel width (W)

Table 8-15: MOSFET *(Sheet 2 of 5)*

Name	Alias	Description
AD	LV3	Area of the drain diode (AD)
AS	LV4	Area of the source diode (AS)
ICVDS	LV5	Initial condition for drain-source voltage (VDS)
ICVGS	LV6	Initial condition for gate-source voltage (VGS)
ICVBS	LV7	Initial condition for bulk-source voltage (VBS)
–	LV8	Device polarity: 1 = forward, -1 = reverse (not used after Star-Hspice release 95.3)
VTH	LV9	Threshold voltage (bias dependent)
VDSAT	LV10	Saturation voltage (VDSAT)
PD	LV11	Drain diode periphery (PD)
PS	LV12	Source diode periphery (PS)
RDS	LV13	Drain resistance (squares) (RDS)
RSS	LV14	Source resistance (squares) (RSS)
XQC	LV15	Charge sharing coefficient (XQC)
GDEFF	LV16	Effective drain conductance (1/RDeff)
GSEFF	LV17	Effective source conductance (1/RSeff)
IDBS	LV18	Drain-bulk saturation current at -1 volt bias
ISBS	LV19	Source-bulk saturation current at -1 volt bias
VDBEFF	LV20	Effective drain bulk voltage
BETAEFF	LV21	BETA effective

Table 8-15: MOSFET *(Sheet 3 of 5)*

Name	Alias	Description
GAMMAEFF	LV22	GAMMA effective
DELTAL	LV23	ΔL (MOS6 amount of channel length modulation) (only valid for LEVELs 1, 2, 3 and 6)
UBEFF	LV24	UB effective (only valid for LEVELs 1, 2, 3 and 6)
VG	LV25	VG drive (only valid for LEVELs 1, 2, 3 and 6)
VFBEFF	LV26	VFB effective
–	LV31	Drain current tolerance (not used in Star-Hspice releases after 95.3)
IDSTOL	LV32	Source diode current tolerance
IDDTOL	LV33	Drain diode current tolerance
COVLGS	LV36	Gate-source overlap capacitance
COVLGD	LV37	Gate-drain overlap capacitance
COVLGB	LV38	Gate-bulk overlap capacitance
VBS	LX1	Bulk-source voltage (VBS)
VGS	LX2	Gate-source voltage (VGS)
VDS	LX3	Drain-source voltage (VDS)
CDO	LX4	DC drain current (CDO)
CBSO	LX5	DC source-bulk diode current (CBSO)
CBDO	LX6	DC drain-bulk diode current (CBDO)
GMO	LX7	DC gate transconductance (GMO)
GDSO	LX8	DC drain-source conductance (GDSO)

Table 8-15: MOSFET *(Sheet 4 of 5)*

Name	Alias	Description
GMBSO	LX9	DC substrate transconductance (GMBSO)
GBDO	LX10	Conductance of the drain diode (GBDO)
GBSO	LX11	Conductance of the source diode (GBSO)
Meyer and Charge Conservation Model Parameters		
QB	LX12	Bulk charge (QB)
CQB	LX13	Bulk charge current (CQB)
QG	LX14	Gate charge (QG)
CQG	LX15	Gate charge current (CQG)
QD	LX16	Channel charge (QD)
CQD	LX17	Channel charge current (CQD)
CGGBO	LX18	CGGBO = ∂Qg/∂Vgb = CGS + CGD + CGB
CGDBO	LX19	CGDBO = ∂Qg/∂Vdb, (for Meyer CGD = -CGDBO)
CGSBO	LX20	CGSBO = ∂Qg/∂Vsb, (for Meyer CGS = -CGSBO)
CBGBO	LX21	CBGBO = ∂Qb/∂Vgb, (for Meyer CGB = -CBGBO)
CBDBO	LX22	CBDBO = ∂Qb/∂Vdb
CBSBO	LX23	CBSBO = ∂Qb/∂Vsb
QBD	LX24	Drain-bulk charge (QBD)
–	LX25	Drain-bulk charge current (CQBD) (not used in Star-Hspice releases after 95.3)

Table 8-15: MOSFET *(Sheet 5 of 5)*

Name	Alias	Description
QBS	LX26	Source-bulk charge (QBS)
–	LX27	Source-bulk charge current (CQBS) (not used after Star-Hspice release 95.3)
CAP_BS	LX28	Bulk-source capacitance
CAP_BD	LX29	Bulk-drain capacitance
CQS	LX31	Channel charge current (CQS)
CDGBO	LX32	¦DGBO = ∂Qd/∂Vgb
CDDBO	LX33	¦DDBO = ∂Qd/∂Vdb
CDSBO	LX34	¦DSBO = ∂Qd/∂Vsb

Table 8-16: Saturable Core Element

Name	Alias	Description
MU	LX0	Dynamic permeability (mu) Weber/(amp-turn-meter)
H	LX1	Magnetizing force (H) Ampere-turns/meter
B	LX2	Magnetic flux density (B) Webers/meter^2

Table 8-17: 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)

Avant!

Chapter 9
Specifying Simulation Options

This chapter describes the options available for changing the Star-Hspice simulation. These options can modify various aspects of the simulation, including output types, accuracy, speed, and convergence. This chapter provides a complete reference of all options available in Star-Hspice from the .OPTION statement.

This chapter covers the following topics:

Setting Control Options

This section describes how to set control options.

.OPTIONS Statement

Control options are set in .OPTIONS statements. You can set any number of options in one .OPTIONS statement, and include any number of .OPTIONS statements in a Star-Hspice input netlist file. All the Star-Hspice control options are listed in Table 9-1. Descriptions of the options follow the table. Options that are relevant to a specific simulation type are also described in the appropriate DC, transient, and AC analysis chapters.

Generally, options default to 0 (OFF) when not assigned a value, either using .OPTIONS <*opt*> = <*val*> or by simply stating the option with no assignment: .OPTIONS <*opt*>. Option defaults are stated in the option descriptions in this section.

Syntax

```
.OPTIONS opt1 <opt2 opt3 ...>
```

opt1 ... Specifies any of the input control options. Many options are in the form <*opt*> = *x*, where <*opt*> is the option name and “x” is the value assigned to that option. All options are described in this section.

Example

You can reset options by setting them to zero (.OPTIONS <*opt*> = 0). You can redefine an option by entering a new .OPTIONS statement for it; the last definition will be used. For example, set the BRIEF option to 1 to suppress printout, and reset BRIEF to 0 later in the input file to resume printout.

```
.OPTIONS BRIEF $ Sets BRIEF to 1 (turns it on)
* Netlist, models,
...
.OPTIONS BRIEF = 0 $ Turns BRIEF off
```

Options Keyword Summary

Table 9-1 lists the keywords for the .OPTIONS statement, grouped by their typical application.

The sections that follow the table provide a description of the options listed under each type of analysis.

Table 9-1: .OPTION Keyword Application Table *(Sheet 1 of 3)*

GENERAL CONTROL OPTIONS		**MODEL ANALYSIS**	**DC OPERATING POINT, DC SWEEP, and POLE/ ZERO**		**TRANSIENT and AC SMALL SIGNAL ANALYSIS**	
Input, Output	*Interfaces*	*General*	*Accuracy*	*Convergence*	*Accuracy*	*Timestep*
ACCT	ARTIST	DCAP	ABSH	CONVERGE	ABSH	ABSVAR
ACOUT	CDS	SCALE	ABSI	CSHDC	ABSV, VNTOL	DELMAX
ALT999	CSDF	TNOM	ABSMOS	DCFOR	ACCURATE	DVDT
ALT9999	MEASOUT		ABSTOL	DCHOLD	ACOUT	FS
ALTER	DLENCSDF					
BINPRNT						
BRIEF	MENTOR	*MOSFETs*	ABSVDC	DCON	CHGTOL	FT
CO	POST	CVTOL	DI	DCSTEP	CSHUNT	IMIN, ITL3
INGOLD	PROBE	DEFAD	KCLTEST	DCTRAN	GSHUNT	IMAX, ITL4
LENNAM	PSF	DEFAS	MAXAMP	DV	DI	ITL5

Table 9-1: .OPTION Keyword Application Table *(Sheet 2 of 3)*

GENERAL CONTROL OPTIONS		MODEL ANALYSIS	DC OPERATING POINT, DC SWEEP, and POLE/ ZERO		TRANSIENT and AC SMALL SIGNAL ANALYSIS	
LIST	SDA	DEFL	RELH	GMAX	GMIN	RELVAR
MEASDGT	ZUKEN	DEFNRD	RELI	GMINDC	GSHUNT	RMAX
MEASFAIL						
MEASSORT						
NODE		DEFNRS	RELMOS	GRAMP	CSHUNT	RMIN
NOELCK	*Analysis*	DEFPD	RELV	*GSHUNT*	MAXAMP	SLOPETOL
NOMOD	ASPEC	DEFPS	RELVDC	ICSWEEP	RELH	TIMERES
NOPAGE	LIMPTS	DEFW		NEWTOL	RELI	
NOTOP	PARHIER	SCALM	*Matrix*	OFF	RELQ	*Algorithm*
NUMDGT	SPICE	WL	ITL1	RESMIN	RELTOL	DVTR
NXX	SEED		ITL2		RISETIME	IMAX
OPTLST		*Inductors*	NOPIV	*Pole/Zero*	TRTOL	IMIN
OPTS	*Error*	GENK	PIVOT, SPARSE	CSCAL	VNTOL, ABSV	LVLTIM
PATHNUM	BADCHR	KLIM		FMAX	*Speed*	MAXORD
PLIM	DIAGNOSTIC		PIVREF	FSCAL	AUTOSTOP	METHOD PURETP
POST_VERSION	NOWARN	*BJTs*	PIVREL	GSCAL	BKPSIZ	MU, XMU
PUTMEAS						
SEARCH	WARNLIMIT	EXPLI	PIVTOL	LSCAL	BYPASS	
VERIFY			SPARSE, PIVOT	PZABS	BYTOL	*Input, Output*
CPU	*Version*	*Diodes*		PZTOL	FAST	INTERP
CPTIME	H9007	EXPLI		RITOL	ITLPZ	ITRPRT
EPSMIN			*Input, Output*	X*n*R, X*n*I	MBYPASS	UNWRAP

Table 9-1: .OPTION Keyword Application Table *(Sheet 3 of 3)*

GENERAL CONTROL OPTIONS		**MODEL ANALYSIS**	**DC OPERATING POINT, DC SWEEP, and POLE/ ZERO**		**TRANSIENT and AC SMALL SIGNAL ANALYSIS**	
EXPMAX			CAPTAB	NEWTOL		
LIMTIM			DCCAP			
			VFLOOR			

General Control Options

Descriptions of the general control options follow. The descriptions are alphabetical by keyword under the sections presented in the table.

Input and Output Options

ACCT

Reports job accounting and runtime statistics at the end of the output listing. Simulation efficiency is determined by the ratio of output points to total iterations. Reporting is automatic unless you disable it.
Choices for ACCT are:

- 0 disables reporting
- 1 enables reporting
- 2 enables reporting of MATRIX statistics

ACOUT

AC output calculation method for the difference in values of magnitude, phase and decibels for prints and plots. The default value equals 1.

The default value, ACOUT = 1, selects the Star-Hspice method, which calculates the difference of the magnitudes of the values. The SPICE method, ACOUT = 0, calculates the magnitude of the differences.

ALT999, ALT9999

This option generates up to 1000 (ALT999) or 10,000 (ALT9999) unique output files from .ALTER runs. Star-Hspice appends a number from 0-999 (ALT999) or 0-9999 (ALT9999) to the extension of the output file. For example, for a .TRAN analysis with 50 .ALTER statements, the filenames would be filename.tr0, filename.tr1, ..., filename.tr50. Without this option, the files would be overwritten after the 36th .ALTER.

altchk

By default, Star-Hspice automatically reports topology errors, not only in the latest elements in your top-level netlist, but also in elements that you redefine using the `.ALTER` statement (altered netlist).

To disable topology checking in redefined elements (that is, to check topology *only* in the top-level netlist, but *not* in the altered netlist), set:

```
.option altchk=0
```

By default, `.option altchk` is set to `1`:

```
.option altchk=1
```

or

```
.option altchk
```

This enables topology checking, in elements that you redefine using the `.ALTER` statement.

BINPRINT

Outputs the binning parameters of the CMI MOSFET model. Currently available only for Level 57.

BRIEF, NXX

Stops printback of the data file until an .OPTIONS BRIEF = 0 or the .END statement is encountered. It also resets the options LIST, NODE and OPTS while setting NOMOD. BRIEF = 0 enables printback. NXX is the same as BRIEF.

CO = x

Sets the number of columns for printout: x can be either 80 (for narrow printout) or 132 (for wide carriage printouts). You also can set the output width by using the .WIDTH statement. The default value equals 80.

INGOLD = x

Specifies the printout data format. Use INGOLD = 2 for SPICE compatibility. The default value equals 0. Numeric output from Star-Hspice can be printed in one of three ways:

```
INGOLD = 0
```

Engineering format, exponents are expressed as a single character:

```
1G = 1e9  1X = 1e6  1K = 1e3   1M = 1e-3
1U = 1e-6 1N = 1e-9 1P = 1e-12
1F = 1e-15
```

```
INGOLD = 1
```

Combined fixed and exponential format (G Format). Fixed format for numbers between 0.1 and 999. Exponential format for numbers greater than 999 or less than 0.1.

```
INGOLD = 2
```

Exclusively exponential format (SPICE2G style). Exponential format generates constant number sizes suitable for post-analysis tools.

Use .OPTIONS MEASDGT in conjunction with INGOLD to control the output data format of .MEASURE results.

LENNAM = x

Specifies the maximum length of names in the operating point analysis results printout. The default value equals 8. The maximum value of *x* is 16.

LIST, VERIFY

Produces an element summary listing of the input data to be printed. Calculates effective sizes of elements and the key values. LIST is suppressed by BRIEF. VERIFY is an alias for LIST.

MEASDGT = x

Used for formatting of the .MEASURE statement output in both the listing file and the .MEASURE output files (*.ma0, .mt0, .ms0*, and so on).

The value of x is typically between 1 and 7, although it can be set as high as 10. The default value equals 4.0. For example, if MEASDGT = 5, numbers displayed by .MEASURE are displayed as:

- Five decimal digits for numbers in scientific notation
- Five digits to the right of the decimal for numbers between 0.1 and 999

In the listing (*.lis*), file, all .MEASURE output values are in scientific notation, so .OPTIONS MEASDGT = 5 results in five decimal digits.

Use MEASDGT in conjunction with .OPTIONS INGOLD = x to control the output data format.

NODE

Causes a node cross reference table to be printed. NODE is suppressed by BRIEF. The table lists each node and all the elements connected to it. The terminal of each element is indicated by a code, separated from the element name with a colon (:). The codes are as follows:

+	Diode anode
-	Diode cathode
B	BJT base
B	MOSFET or JFET bulk
C	BJT collector
D	MOSFET or JFET drain
E	BJT emitter
G	MOSFET or JFET gate
S	BJT substrate
S	MOSFET or JFET source

For example, part of a cross reference might look like:

```
1 M1:B D2:+ Q4:B
```

This line indicates that the bulk of M1, the anode of D2, and the base of Q4 are all connected to node 1.

NOELCK — No element check; bypasses element checking to reduce preprocessing time for very large files.

NOMOD — Suppresses the printout of model parameters

NOPAGE — Suppresses page ejects for title headings

NOTOP — Suppresses topology check resulting in increased speed for preprocessing very large files

NUMDGT = x — Sets the number of significant digits printed for output variable values. The value of x is typically between 1 and 7, although it can be set as high as 10. The default value equals 4.0. This option does not affect the accuracy of the simulation.

NXX — Stops printback of the data file until an .OPTIONS BRIEF = 0 or the .END statement is encountered. It also resets the options LIST, NODE and OPTS while setting NOMOD. BRIEF = 0 enables printback. NXX is the same as BRIEF.

OPTLST = x — Outputs additional optimization information:

- 0 No information (default)
- 1 Prints parameter, Broyden update, and bisection results information
- 2 Prints gradient, error, Hessian, and iteration information
- 3 Prints all of the above and Jacobian

OPTS	Prints the current settings of all control options. If any of the default values of the options have been changed, the OPTS option prints the values actually used for the simulation. Suppressed by the BRIEF option.
PATHNUM	Prints subcircuit path numbers instead of path names.
PLIM = x	Specifies plot size limits for of current and voltage plots: 0 Finds a common plot limit and plots all variables on one graph at the same scale 1 Enables SPICE-type plots, in which a separate scale and axis are created for each plot variable This option has no effect on graph data POST processing.
POST_VERSION = x	Sets the post-processing output version to x = 9601 or 9007. x = 9007 truncates the node name in the post-processor output file, to a maximum of 16 characters. x = 9601 sets the node name length for the output file, consistent with the input restrictions (1024 characters).

POST_VERSION =2001	Sets the post-processing output version to 2001. This option shows you the new output file header, which includes the right number of output variables, rather than **** when the number exceeds 9999. If you set .option post_version=2001 post=2 in the netlist, then you receive more accurate ASCII results.
	The syntax is:
	`.option post_version=2001`
	To use binary values with double precision in the output file, include the following in the input file:
	`**********`
	`.option post (or post=1) post_version=2001`
	`**********`
	For more accurate simulation results, comment this format.

SEARCH	Sets the search path for libraries and included files. Star-Hspice automatically looks in the directory specified with .OPTIONS SEARCH for libraries referenced in the simulation.
VERIFY	Produces an element summary listing of the input data to be printed. Calculates effective sizes of elements and the key values. LIST is suppressed by BRIEF. VERIFY is an alias for LIST.

CPU Options

CPTIME = x	Sets the maximum CPU time, in seconds, allotted for this job. When the time allowed for the job exceeds CPTIME, the results up to that point are printed or plotted and the job is concluded. Use this option when uncertain about how long the simulation will take, especially when debugging new data files. Also see LIMTIM. The default value equals 1e7 (400 days).
EPSMIN = x	Specifies the smallest number that can be added or subtracted on a computer, a constant value. The default value equals 1e-28.
EXPMAX = x	Specifies the largest exponent you can use for an exponential before overflow occurs. Typical value for an IBM platform is 350.
LIMTIM = x	Sets the amount of CPU time reserved for generating prints and plots in case a CPU time limit (CPTIME = x) causes termination. The default value equals 2 (seconds). This default is normally sufficient time for short printouts and plots.

Interface Options

ARTIST = x	ARTIST = 2 enables the Cadence Analog Artist interface. This option requires a specific license.
CDS, SDA	CDS = 2 produces a Cadence WSF ASCII format post-analysis file for Opus™. This option requires a specific license. SDA is the same as CDS.
CSDF	Selects Common Simulation Data Format (Viewlogic-compatible graph data file format)
DLENCSDF	If you use the Common Simulation Data Format (Viewlogic-compatible graph data file format) as your output format, this "digit length" option specifies the number of digits to include, either in scientific notation (exponents), or to the right of the decimal point. Valid values are any integer from 1 to 10. The default value is 5. If you assign a *floating* decimal point type, or if you specify a number of digits outside the 1 to 10 range, then Star-Hspice uses the default—for example, placing 5 digits to the right of a decimal point.
MEASOUT	*O*utputs .MEASURE statement values and sweep parameters into an ASCII file for post-analysis processing by AvanWaves or other analysis tools. The output file is named *<design>.mt#*, where # is incremented for each .TEMP or .ALTER block. For example, for a parameter sweep of an output load, measuring the delay, the *.mt#* file contains data for a delay versus fanout plot. The default value equals 1. You can set this option to 0 (off) in the *hspice.ini* file.
MENTOR = x	MENTOR = 2 enables the Mentor MSPICE-compatible ASCII interface. Requires a specific license.

MONTECON — Continues a Monte Carlo analysis, and retrieves the next random value, even if a non-convergence occurs. Although the random value might be too large or too small to cause a convergence failure, using this option allows other types of analysis to use this random value from the Monte Carlo analysis.

POST = x — Enables storing of simulation results for analysis using the AvanWaves graphical interface or other methods.POST = 1 saves the results in binary. POST = 2 saves the results in ASCII format. POST = 3 saves the results in New Wave binary format. Set the POST option, and use the .PROBE statement to specify which data you want saved. The default value equals 1.

To use binary values with double precision in the output file, include the following in the input file:

```
*****************************************
.option post (or post=1) post_version=2001
*****************************************
```

For more accurate simulation results, comment this format.

PROBE — Limits the post-analysis output to just the variables designated in .PROBE, .PRINT, .PLOT, and .GRAPH statements. By default, Star-Hspice outputs all voltages and power supply currents in addition to variables listed in .PROBE/.PRINT/.PLOT/.GRAPH statements. Use of PROBE significantly decreases the size of simulation output files.

PSF = x — Specifies whether Star-Hspice outputs binary or ASCII when Star-Hspice is run from Cadence Analog Artist. The value of *x* can be 1 or 2. If *x* is 2, Star-Hspice produces ASCII output. If .OPTIONS ARTIST PSF = 1, Star-Hspice produces binary output.

SDA CDS = 2 produces a Cadence WSF ASCII format post-analysis file for Opus. This option requires a specific license. SDA is the same as CDS.

ZUKEN = x If x is 2, enables the Zuken interactive interface. If x is 1, disables it. The default value equals 1.

Analysis Options

ASPEC Sets Star-Hspice into ASPEC compatibility mode. With this option set, Star-Hspice can read ASPEC models and netlists and the results are compatible. The default value equals 0 (Star-Hspice mode).

Note: When the ASPEC option is set, the following model parameters default to ASPEC values:

ACM = 1:

Default values for CJ, IS, NSUB, TOX, U0, UTRA are changed.

Diode Model:

TLEV = 1 affects temperature compensation of PB.

MOSFET Model:

TLEV = 1 affects PB, PHB, VTO, and PHI.

SCALM, SCALE:

Sets model scale factor to microns for length dimensions.

WL:

Reverses implicit order on MOSFET element of width and length.

FFTOUT — Prints out 30 harmonic fundamentals, sorted by size, THD, SNR, and SFDR, but only if you specify both a `.OPTION fftout` statement and a `.fft freq=`*xxx* statement.

LIMPTS = x — Sets the total number of points that you can print or plot in AC analysis. It is not necessary to set LIMPTS for DC or transient analysis, as Star-Hspice spools the output file to disk. The default value equals 2001.

PARHIER — Selects the parameter passing rules that control the evaluation order of subcircuit parameters. They only apply to parameters with the same name at different levels of subcircuit hierarchy.
The options are:

LOCAL — During analysis of a subcircuit, a parameter name specified in the subcircuit prevails over the same parameter name specified at a higher hierarchical level.

GLOBAL — A parameter name specified at a higher hierarchical level prevails over the same parameter name specified at a lower level.

SPICE Makes Star-Hspice compatible with Berkeley SPICE. When the option SPICE is set, the following options and model parameters are used:

Example of general parameters used with .OPTIONS SPICE:

```
          TNOM = 27 DEFNRD = 1 DEFNRS = 1
INGOLD = 2
          ACOUT = 0 DC
          PIVOT PIVTOL = IE-13
PIVREL = 1E-3 RELTOL = 1E-3
          ITL1 = 100
          ABSMOS = 1E-6 RELMOS = 1E-3
ABSTOL = 1E-12
          VNTOL = 1E-6
          ABSVDC = 1E-6 RELVDC = 1E-3
RELI = 1E-3
```

Example of transient parameters used with .OPTIONS SPICE:

```
DCAP = 1 RELQ = 1E-3 CHGTOL-1E-14 ITL3 = 4 ITL4 = 10
ITL5 = 5000

FS = 0.125 FT = 0.125
```

Example of model parameters used with .OPTIONS SPICE:

```
For BJT: MJS = 0
For MOSFET, CAPOP = 0
LD = 0 if not user-specified
UTRA = 0 not used by SPICE for LEVEL = 2
NSUB must be specified
NLEV = 0 for SPICE noise equation
```

SEED User-specified random number generator starting seed for Monte Carlo analysis. The minimum value is 1 and maximum value is 259200.

Error Options

BADCHR Generates a warning when a nonprintable character is found in an input file.

DIAGNOSTIC *L*ogs the occurrence of negative model conductances.

NOWARN Suppresses all warning messages except those generated from statements in .ALTER blocks.

WARNLIMIT = x Limits the number of times that certain warnings appear in the output listing, to reduce the output listing file size. x is the maximum number of warnings for each warning type. This limit applies to these warning messages:

- MOSFET has negative conductance
- Node conductance is zero
- Saturation current is too small
- Inductance or capacitance is too large

The default value equals 1.

Version Options

H9007 Sets general control option default values to correspond to the values for Star-Hspice Release H9007D. The EXPLI model parameter is not used when this option is set.

Model Analysis Options

General Options

DCAP

The DCAP option selects the equations used in calculating the depletion capacitance for Level 1 and 3 diodes and BJTs. See the individual device model chapters for information concerning the equations used.

MODSRH

If MODSRH=1, Hspice will not load the model and will consider it not referenced when the model is described by .model but does not appear in the netlist. This option will shorten the run time when many models are referenced but not called by an element in the netlist.

The default value is MODSRH=0. If MODSRH=1, then the read-in time will increase slightly.

```
example.sp:
* modsrh used incorrectly
.option post modsrh=1
xi1 net8 b c t6
xi0 a b net8 t6
v1 a 0 pulse 3.3 0.0 10E-6 1E-9 1E-9
+ 25E-6 50E-6
v2 b 0 2
v3 c 0 3
.model nch nmos level=49 version=3.2
.end
```

This input file searches for t6.inc automatically. If you use the *nch* model in t6.inc, and set MODSRH to 1, then Star-Hspice does not load *nch*. Do not set MODSRH=1 in this type of included file call. Use this option in front of the .model card definition.

SCALE

Element scaling factor. This option scales parameters in element cards, by their value. The default value is 1.

HIER_SCALE The HIER_SCALE option enables using the *S* parameter for scaling sub-circuits. `0` indicates to interpret *S* as a user-defined parameter. `1` indicates to interpret *S* as a Star-Hspice scale parameter. For more information about the *S* parameter, see "S (Scale) Parameter" on page 3-52.

TNOM The reference temperature for the simulation. This is the temperature at which component derating is zero. The default is 25 degrees Celsius, or if .OPTION SPICE is enabled the default is 27 degrees Celsius.

MOSFET Control Options

CVTOL Changes the number of numerical integration steps in the calculation of the gate capacitor charge for a MOSFET using CAPOP = 3. See the discussion of CAPOP = 3 in the chapter 18 for explicit equations and discussion.

DEFAD Default value for MOSFET drain diode area. The default value equals 0.

DEFAS Default value for MOSFET source diode area. The default value equals 0.

DEFL Default value for MOSFET channel length. The default value equals $1e^{-4}$m.

DEFNRD Default value for the number of squares for the drain resistor on a MOSFET. The default value equals 0.

DEFNRS Default value for the number of squares for the source resistor on a MOSFET. The default value equals 0.

DEFPD Default value for MOSFET drain diode perimeter. The default value equals 0.

DEFPS Default value for MOSFET source diode perimeter. The default value equals 0.

DEFW	Default value for MOSFET channel width. The default value equals $1e^{-4}$m.
SCALM	Model scaling factor. This option will scale parameters defined in device model cards by its value. The default value equals 1. See the individual device model chapters for information about which parameters are scaled.
WL	This option changes the order of specifying MOS element VSIZE from the default order length-width to width-length. The default value equals 0.

Inductors

GENK	Enables automatic computation of second-order mutual inductance, for several coupled inductors, where a value of 1 enables the calculation. The default value equals 1.
KLIM	Minimum mutual inductance, below which automatic second-order mutual inductance calculation no longer proceeds. KLIM is unitless (analogous to coupling strength specified in the K Element). Typical KLIM values are between .5 and 0.0. The default value is 0.01.

BJTs

EXPLI	Current explosion model parameter. The PN junction characteristics above the explosion current are linearized, with the slope determined at the explosion point. This speeds up simulation and improves convergence. The default value equals 0.0 amp/AREAeff.

Diodes

EXPLI	Current explosion model parameter. The PN junction characteristics above the explosion current are linearized, with the slope determined at the explosion point. This speeds up simulation and improves convergence. The default value equals 0.0 amp/AREAeff.

DC Operating Point, DC Sweep, and Pole/Zero

Accuracy

ABSH = x

Sets the absolute current change through voltage-defined branches (voltage sources and inductors). In conjunction with DI and RELH, ABSH is used to check for current convergence. The default value equals 0.0.

ABSI = x

Sets the absolute branch current error tolerance in diodes, BJTs, and JFETs, during DC and transient analysis. Decrease ABSI if accuracy is more important than convergence time.

If you want an analysis with currents less than 1 nanoamp, change ABSI to a value at least two orders of magnitude smaller than the minimum expected current.

The default value is 1e-9 for KCLTEST = 0, or 1e-6 for KCLTEST = 1.

ABSMOS = x

Current error tolerance for MOSFET device, in DC or transient analysis. The ABSMOS setting determines if the drain-to-source current solution has converged. If the difference between the last and the present iteration's drain-to-source current is less than ABSMOS, or if it is greater than ABSMOS, but the percent change is less than RELMOS, the drain-to-source current converged.

Star-Hspice then checks the other accuracy tolerances. If they indicate convergence, the circuit solution at that timepoint is solved, and the next timepoint solution is calculated. For low power circuits, optimization, and single transistor simulations, set ABSMOS = 1e-12. The default value equals 1e-6 (amperes).

ABSTOL = x Sets the absolute error tolerance for branch currents. Decrease ABSTOL if accuracy is more important than convergence time.

ABSVDC = x Sets the absolute minimum voltage for DC and transient analysis. Decrease ABSVDC if accuracy is of more concern than convergence. If voltages less than 50 microvolts are required, ABSVDC can be reduced to two orders of magnitude less than the smallest desired voltage. This ensures at least two digits of significance. Typically ABSVDC need not be changed unless the circuit is a high voltage circuit. For 1000-volt circuits, a reasonable value can be 5 to 50 millivolts. The default value equals VNTOL (VNTOL default = 50 μV).

DI = x Sets the maximum iteration-to-iteration current change through voltage defined branches (voltage sources and inductors). This option is only applicable when the value of the ABSH control option is greater than 0. The default value equals 0.0.

KCLTEST Activates the KCL test (Kirchhoff's Current Law) function. This test results in a longer simulation time, especially for large circuits, but provides a very accurate check of the solution. The default value equals 0.

When set to 1, Star-Hspice sets the following options:

- RELMOS and ABSMOS options are set to 0 (off).
- ABSI is set to 1e-16 A
- RELI is set to 1e-6

To satisfy the KCL test, the following condition must be satisfied for each node:

$$\left|\Sigma i_b\right| < RELI \cdot \Sigma\left|i_b\right| + ABSI$$

where the i_bs are the node currents.

MAXAMP = x Sets the maximum current through voltage-defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, an error is issued. The default value equals 0.0.

RELH = x Sets relative current tolerance through voltage-defined branches (voltage sources and inductors), and checks current convergence. Use this option only if the ABSH control value is greater than zero. The default is 0.05.

RELI = x Sets the relative error/tolerance change from iteration to iteration to determine convergence for all currents in diode, BJT, and JFET devices. (RELMOS sets the tolerance for MOSFETs). This is the change in current from the value calculated at the previous timepoint. The default value equals 0.01 for KCLTEST = 0, 1e-6 for KCLTEST = 1.

RELMOS = x Sets the relative drain-to-source current error tolerance percent, from iteration to iteration, to determine convergence for currents in MOSFET devices. (RELI sets the tolerance for other active devices.) This is the change in current since the previous timepoint. Star-Hspice uses RELMOS only when the current is greater than the floor value, ABSMOS. The default value equals 0.05.

RELV = x Sets the relative error tolerance for voltages. When voltages or currents exceed their absolute tolerances, the RELV test is used to determine convergence. Increasing RELV increases the relative error. In general, RELV should be left at its default value. RELV controls simulator charge conservation. For voltages, RELV is the same as RELTOL. The default value equals 1e-3.

RELVDC = x Sets the relative error tolerance for voltages. When voltages or currents exceed their absolute tolerances, the RELVDC test is used to determine convergence. Increasing RELVDC increases the relative error. In general, RELVDC should be left at its default value. RELVDC controls simulator charge conservation. The default value equals RELTOL (RELTOL default = 1e-3).

Matrix-Related

ITL1 = x Sets the maximum DC iteration limit. Increasing this value is unlikely to improve convergence for small circuits. Values as high as 400 have resulted in convergence for certain large circuits with feedback, such as operational amplifiers and sense amplifiers. Something is usually wrong with a model if more than 100 iterations are required for convergence. Set .OPTION ACCT to obtain a listing of how many iterations are required for an operating point. The default value equals 200.

ITL2 = x Sets the DC transfer curve iteration limit. Increasing the iteration limit can be effective in improving convergence only on very large circuits. The default value equals 50.

NOPIV Prevents Star-Hspice from switching automatically to pivoting matrix factorization, when a node conductance is less than PIVTOL. NOPIV inhibits pivoting (see PIVOT).

PIVOT = x

Provides different pivoting algorithm selections. These can be used effectively to reduce simulation time and achieve convergence in circuits that produce hard-to-solve matrix equations. The pivot algorithm is selected by setting PIVOT to one of the following values:

- 0: Original nonpivoting algorithm
- 1: Original pivoting algorithm
- 2: Pick largest pivot in row algorithm
- 3: Pick best in row algorithm
- 10: Fast nonpivoting algorithm, more memory required
- 11: Fast pivoting algorithm, more memory required than PIVOT values less than 11
- 12: Pick largest pivot in row algorithm, more memory required than for PIVOT values less than 12
- 13: Fast best pivot: faster, more memory required than for PIVOT values less than 13

The default value equals 10.

The fastest algorithm is PIVOT = 13, which can improve simulation time by up to ten times on very large circuits. However, the PIVOT = 13 option requires substantially more memory for the simulation. Some circuits with large conductance ratios, such as switching regulator circuits, might need pivoting. If PIVTOL = 0, Star-Hspice automatically changes from nonpivoting to a row pivot strategy upon detection of any diagonal matrix entry less than PIVTOL. This strategy provides the time and memory advantages of nonpivoting inversion, while avoiding unstable simulations and incorrect results. Use .OPTION NOPIV to prevent pivoting from being used under any circumstances.

For very large circuits, PIVOT = 10, 11, 12, or 13 can require excessive memory.

If Star-Hspice switches to pivoting during a simulation, the message “pivot change on the fly” is printed, followed by the node numbers causing the problem. Use .OPTION NODE to obtain a node-to-element cross reference.

SPARSE is the same as PIVOT.

PIVREF Pivot reference. Used in PIVOT = 11, 12, 13 to limit the size of the matrix. The default value equals 1e+8.

PIVREL = x Sets the maximum/minimum row/matrix ratio. Use only for PIVOT = 1. Large values for PIVREL can result in very long matrix pivot times. If the value is too small, however, no pivoting occurs. It is best to start with small values of PIVREL, using an adequate but not excessive value for convergence and accuracy. The default value equals 1E-20 (max = 1e-20, min = 1).

PIVTOL = x Sets the absolute minimum value for which a matrix entry is accepted as a pivot. PIVTOL is used as the minimum conductance in the matrix when PIVOT = 0. The default value equals 1.0e-15.
Note: PIVTOL should always be less than GMIN or GMINDC. Values approaching 1 yield increased pivot.

SPARSE = x Provides different pivoting algorithm selections. These can be used effectively to reduce simulation time and achieve convergence in circuits that produce hard-to-solve matrix equations. The pivot algorithm is selected by setting PIVOT to one of the following values:

- 0: Original nonpivoting algorithm
- 1: Original pivoting algorithm
- 2: Pick largest pivot in row algorithm
- 3: Pick best in row algorithm
- 10: Fast nonpivoting algorithm, more memory required
- 11: Fast pivoting algorithm, more memory required than PIVOT values less than 11

- 12: Pick largest pivot in row algorithm, more memory required than for PIVOT values less than 12
- 13: Fast best pivot: faster, more memory required than for PIVOT values less than 13
- The default value equals 10.

The fastest algorithm is PIVOT = 13, which can improve simulation time by up to ten times on very large circuits. However, the PIVOT = 13 option requires substantially more memory for the simulation. Some circuits with large conductance ratios, such as switching regulator circuits, might need pivoting. If PIVTOL = 0, Star-Hspice automatically changes from nonpivoting to a row pivot strategy upon detection of any diagonal matrix entry less than PIVTOL. This strategy provides the time and memory advantages of nonpivoting inversion, while avoiding unstable simulations and incorrect results. Use .OPTION NOPIV to prevent pivoting from being used under any circumstances.

For very large circuits, PIVOT = 10, 11, 12, or 13 can require excessive memory.

If Star-Hspice switches to pivoting during a simulation, the message “pivot change on the fly” is printed, followed by the node numbers causing the problem. Use .OPTION NODE to obtain a node-to-element cross reference.

SPARSE is the same as PIVOT.

Input and Output

CAPTAB Prints table of single-plate node capacitance for diodes, BJTs, MOSFETs, JFETs, and passive capacitors, at each operating point.

DCCAP Generates C-V plots, and prints out the capacitance values of a circuit (both model and element), during a DC analysis. You can use a DC sweep of the capacitor to generate C-V plots. The default value equals 0 (off).

VFLOOR = x Sets a lower limit for the voltages that print in the output listing. All voltages lower than VFLOOR are printed as 0. This affects only the output listing: VNTOL (ABSV) sets the minimum voltage used in a simulation.

Convergence

CONVERGE Invokes different methods to solve non-convergence:

CONVERGE = -1
together with DCON = -1, disables autoconvergence

CONVERGE = 1
uses the Damped Pseudo Transient Algorithm. If simulation fails to converge within the amount of CPU time set in the CPTIME control option, simulation halts.

CONVERGE = 2
uses a combination of DCSTEP and GMINDC ramping

CONVERGE = 3
invokes the source stepping method

Even if it is not set in an .OPTIONS statement, the CONVERGE option is activated if a matrix floating point overflows, or a timestep is too small. The default is 0.

In the event of a matrix floating point overflow, Star-Hspice sets CONVERGE = 1.

CSHDC The same option as CSHUNT, but only with the CONVERGE option.

DCFOR = x

Used in conjunction with the DCHOLD option and the .NODESET statement to enhance the DC convergence properties of a simulation. DCFOR sets the number of iterations that are to be calculated after a circuit converges in the steady state. Since the number of iterations after convergence is usually zero, DCFOR adds iterations (and computational time) to the calculation of the DC circuit solution. DCFOR helps ensure that a circuit has actually, not falsely, converged. The default value equals 0.

DCHOLD = x

DCFOR and DCHOLD are used together for the initialization process of a DC analysis. They enhance the convergence properties of a DC simulation. DCFOR and DCHOLD work together with the .NODESET statement.

DCHOLD specifies how many iterations to hold a node at the voltage values specified in a .NODESET statement. The effects of DCHOLD on convergence differ according to the DCHOLD value, and the number of iterations needed to obtain DC convergence.

If a circuit converges in the steady state in fewer than DCHOLD iterations, the DC solution includes the values set by the .NODESET statement. However, if the circuit requires more than DCHOLD iterations to converge, the values set in the .NODESET statement are ignored and the DC solution is calculated with the .NODESET fixed source voltages open circuited. The default value equals 1.

DCON = X

In the case of convergence problems, Star-Hspice automatically sets DCON = 1 and the following calculations are made:

$$DV = max\left(0.1, \frac{V_{max}}{50}\right), \text{ if } DV = 1000$$

$$GRAMP = max\left(6, \log_{10}\left(\frac{I_{max}}{GMINDC}\right)\right)$$

$$ITL1 = ITL1 + 20 \cdot GRAMP$$

where V_{max} is the maximum voltage and $I_{max}$is the maximum current.

If convergence problems persist, Star-Hspice sets DCON = 2 (the same as the above, except DV = 1e6). The above calculations are used if DCON = 1 or 2. DCON = 1 is automatically invoked if the circuit fails to converge. DCON = 2 is invoked if DCON = 1 fails.

If the circuit contains uninitialized flip-flops or discontinuous models, the simulation might be unable to converge. Setting DCON = -1 and CONVERGE = -1 disables the autoconvergence algorithm and provides a list of nonconvergent nodes and devices.

DCSTEP = x

Used to convert DC model and element capacitors to a conductance to enhance DC convergence properties. The value of the element capacitors are all divided by DCSTEP to obtain a DC conductance model. The default value equals 0 (seconds).

DCTRAN

Invokes different methods for solving nonconvergence problems:

CONVERGE = -1
together with DCON = -1, disables autoconvergence

CONVERGE = 1
uses the Damped Pseudo Transient Algorithm. If the simulation fails to converge within the amount of CPU time set in the CPTIME control option, simulation halts.

CONVERGE = 2
uses a combination of DCSTEP and GMINDC ramping

CONVERGE = 3
invokes the source stepping method

Even if it is not set in an .OPTIONS statement, the CONVERGE option activates if a matrix floating point overflows, or a timestep is too small. The default is 0.

In the event of a matrix floating point overflow, Star-Hspice sets CONVERGE = 1.

DCTRAN is an alias for CONVERGE.

DV = x The maximum iteration-to-iteration voltage change for all circuit nodes, in both DC and transient analysis. require Some high-gain bipolar amplifiers values of 0.5 to 5.0 to achieve a stable DC operating point. CMOS circuits frequently require a value of about 1 volt for large digital circuits. The default value is 1000 (or 1e6 if DCON = 2).

GMAX = x The conductance in parallel with the current source used for .IC and .NODESET initialization conditions circuitry. Some large bipolar circuits can require GMAX set to 1 for convergence. The default value equals 100 (mho).

GMINDC = x This conductance is placed parallel to all pn junctions and MOSFET nodes, for DC analysis. GMINDC helps overcome DC convergence problems caused by low values of off-conductance for pn junctions and MOSFETs. You can use GRAMP to reduce GMINDC by one order of magnitude for each step. Set GMINDC between 1e-4 and PIVTOL. The default is 1e-12.

Large values of GMINDC can cause unreasonable circuit response. If large values are required for convergence, a bad model or circuit is suspect. In the event of a matrix floating point overflow, if GMINDC is 1.0e-12 or less, Star-Hspice sets it to 1.0e-11.

Star-Hspice manipulates `GMINDC` in autoconverge mode.

GRAMP = x

Value is set by Star-Hspice during the autoconvergence procedure. GRAMP is used in conjunction with the GMINDC convergence control option to find the smallest value of GMINDC that results in DC convergence.

GRAMP specifies the conductance range over which GMINDC is to be swept during a DC operating point analysis. Star-Hspice substitutes values of GMINDC over this range and simulates at each value. It then picks the lowest value of GMINDC that resulted in the circuit converging in the steady state.

If you sweep GMINDC between 1e-12 mhos (the default) and 1e-6 mhos, GRAMP is set to 6 (the value of the exponent difference between the default and the maximum conductance limit). In this case, GMINDC is first set to 1e-6 mhos, and the circuit is simulated.

If convergence occurs, Star-Hspice sets GMINDC to 1e-7 mhos, and simulates the circuit again. The sweep continues until all values on the GRAMP ramp have been simulated. If the combined conductance of GMINDC and GRAMP is greater than 1e-3 mho, a false convergence can occur. The default value equals 0.

GSHUNT

Conductance added from each node to ground. The default value is zero. Add a small GSHUNT to each node to possibly solve “Timestep too small” problems caused by high frequency oscillations or by numerical noise.

ICSWEEP

For a parameter or temperature sweep, saves the results of the current analysis for use as the starting point in the next analysis in the sweep. When ICSWEEP = 1, the current results are used in the next analysis. When ICSWEEP = 0, the results of the current analysis are not used in the next analysis. The default value equals 1.

NEWTOL Calculates one more iterations past convergence for every DC solution and timepoint circuit solution calculated. When NEWTOL is not set, once convergence is determined, the convergence routine is ended and the next program step begun. The default value equals 0.

OFF Initializes the terminal voltages of all active devices to zero, if they are not initialized to other values. For example, if the drain and source nodes of a transistor are not both initialized using .NODESET or .IC statements, or by connecting them to sources, then the OFF option initializes all nodes of the transistor to zero.

The OFF option is checked before element IC parameters,. If you assigned an element IC parameter to a node, the node is initialized to the element IC parameter value, even if the OFF option previously set tit o zero. (You can use the OFF element parameter to initialize the terminal voltages to zero for particular active devices).

The OFF option is used to help find exact DC operating point solutions for large circuits.

RESMIN = x Specifies the minimum resistance value for all resistors, including parasitic and inductive resistances. The default value equals 1e-5 (ohm). Range: 1e-15 to 10 ohm.

Pole/Zero Control Options

CSCAL Sets the capacitance scale. Capacitances are multiplied by CSCAL. The default value equals 1e+12 (thus, by default, all capacitances are entered in units of pF).

FMAX Sets the limit for maximum pole and zero angular frequency value. The default value is 1.0e+12 rad/sec.

FSCAL Sets the frequency scale. Frequency is multiplied by FSCAL. The default value equals 1e-9 (thus, by default, all frequencies are entered in units of GHz).

GSCAL Sets the conductance scale. Conductances are multiplied by GSCAL. Resistances are divided by GSCAL. The default value equals 1e+3 (thus, by default, all resistances are entered in units of kΩ).

LSCAL Sets inductance scale. Inductances are multiplied by LSCAL. The default value equals 1e+6 (thus, by default, all inductances are entered in units of μH).

The scale factors must satisfy the following relations:

$$GSCA = CSCAL \cdot FSCAL$$

$$GSCAL = \frac{1}{LSCAL} \cdot FSCAL$$

If scale factors are changed, it might be necessary to modify the initial Muller points (X0R, X0I), (X1R, X1I) and (X2R, X2I), even though Star-Hspice multiplies initial values by (1e-9/GSCAL).

PZABS Sets absolute tolerances for poles and zeros. This option affects the low frequency poles or zeros. It is used as follows:

$$\text{If } (|X_{real}| + |X_{imag}| < PZABS),$$

$$\text{then } X_{real} = 0 \text{ and } X_{imag} = 0 .$$

It is also used for convergence tests. The default value equals 1e-2.

PZTOL Sets the relative error tolerance for poles or zeros. The default value equals 1.0e-6.

RITOL

Sets the minimum ratio value for (real/imaginary) or (imaginary/real) parts of the poles or zeros. RITOL is used as follows:

$$\text{If } |X_{imag}| \leq RITOL \cdot |X_{real}| \text{ , then } X_{imag} = 0$$

$$\text{If } |X_{real}| \leq RITOL \cdot |X_{imag}| \text{ , then } X_{real} = 0$$

The default value equals 1.0e-6.

(X0R,X0I),
(X1R,X1I),
(X2R,X2I)

The three complex starting points in the Muller pole/zero analysis algorithm are:

```
X0R = -1.23456e6  X0I = 0.0
X1R = -1.23456e5  X1I = 0.0
X2R = +.23456e6  X2I = 0.0
```

These initial points are multiplied by FSCAL.

Transient and AC Small Signal Analysis

Accuracy

ABSH = x

Sets the absolute current change through voltage defined branches (voltage sources and inductors). In conjunction with DI and RELH, ABSH is used to check for current convergence. The default value equals 0.0.

ABSV = x

Sets the absolute minimum voltage for DC and transient analysis. Decrease VNTOL if accuracy is of more concern than convergence. If voltages less than 50 microvolts are required, VNTOL can be reduced to two orders of magnitude less than the smallest desired voltage, ensuring at least two digits of significance. Typically, VNTOL need not be changed unless the circuit is a high voltage circuit. For 1000 volt circuits, a reasonable value can be 5 to 50 millivolts. ABSV is the same as VNTOL. The default value equals 50 (microvolts).

ACCURATE

Selects a time algorithm that uses LVLTIM = 3 and DVDT = 2, for circuits such as high-gain comparators. Use this option with circuits that combine high gain and large dynamic range, to guarantee accurate solutions. When set to 1, ACCURATE sets these control options:

- LVLTIM = 3
- DVDT = 2
- RELVAR = 0.2
- ABSVAR = 0.2
- FT = 0.2
- RELMOS = 0.01

The default value equals 0.

ACOUT	AC output calculation method for the difference in values of magnitude, phase and decibels for prints and plots. The default value equals 1. The default value, ACOUT = 1, selects the Star-Hspice method, which calculates the difference of the magnitudes of the values. The SPICE method, ACOUT = 0, calculates the magnitude of the differences.
CHGTOL = x	Sets the charge error tolerance when LVLTIM = 2 is set. CHGTOL, along with RELQ, sets the absolute and relative charge tolerance for all Star-Hspice capacitances. The default value equals 1e-15 (coulomb).
CSHUNT	Capacitance added from each node to ground. Adding a small CSHUNT to each node can solve some “internal timestep too small” problems caused by high-frequency oscillations or numerical noise. The default value is 0.
DI = x	Sets the maximum iteration-to-iteration current change through voltage defined branches (voltage sources and inductors). This option is only applicable when the value of the DI control option is greater than 0. The default value equals 0.0.
GMIN = x	Sets the minimum conductance allowed for in a transient analysis time sweep. The default value equals 1e-12.
GSHUNT	Conductance added from each node to ground. The default is zero. Adding a small GSHUNT to each node can solve some “internal timestep too small” problems, caused by high frequency oscillations or numerical noise.
MAXAMP = x	Sets the maximum current through voltage-defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, an error is issued. The default value equals 0.0.

RELH = x	Sets relative current tolerance through voltage defined branches (voltage sources and inductors). RELH is used to check current convergence. This option is applicable only if the value of the ABSH control option is greater than zero. The default value equals 0.05.
RELI = x	Sets the relative error/tolerance change from iteration to iteration, to determine convergence for all currents in diode, BJT, and JFET devices. (RELMOS sets tolerance for MOSFETs). This is the change in current, from the value calculated at the previous timepoint. The default is 0.01 for KCLTEST = 0, or 1e-4 for KCLTEST = 1.
RELQ = x	Used in the local truncation error timestep algorithm (LVLTIM = 2). RELQ changes the size of the timestep. If the capacitor charge calculation of the present iteration exceeds that of the past iteration by a percentage greater than the value of RELQ, the internal timestep (Delta) is reduced. The default value equals 0.01.
RELTOL, RELV	Sets the relative error tolerance for voltages. RELV is used in conjunction with the ABSV control option, to determine voltage convergence. Increasing RELV increases the relative error. RELV is the same as the RELTOL. Options RELI and RELVDC default to the RELTOL value. The default value equals 1e-3.

RISETIME

Specifies the smallest risetime of the signal, .OPTION RISETIME = x. Currently, used only in transmission line models. In the U Element, the number of lumps is determined by:

$$MIN\left[20, 1 + \left(\frac{TDeff}{RISETIME}\right) \cdot 20\right]$$

where *TDeff* is the total end-to-end delay in a transmission line. The W Element uses, RISETIME only if R_s or G_d is nonzero. In such cases, RISETIME determines the maximum frequency content of signals.

TRTOL = x

Used in the local truncation error timestep algorithm (LVLTIM = 2). TRTOL is multiplied by the internal timestep, which the local truncation error timestep algorithm generates. TRTOL reduces simulation time, while maintaining accuracy. It estimates the amount of error introduced when the algorithm truncates the Taylor series expansion. This error reflects the minimum time-step, to reduce simulation time and maintain accuracy. The range of TRTOL is 0.01 to 100; typical values are 1 to 10. If you set TRTOL to 1, the minimum value, a very small timestep is used. As you increase the TRTOL setting, the timestep size increases. The default is 7.0.

VNTOL = x, ABSV

Sets the absolute minimum voltage for DC and transient analysis. Decrease VNTOL if accuracy is of more concern than convergence. If voltages less than 50 microvolts are required, VNTOL can be reduced to two orders of magnitude less than the smallest desired voltage, ensuring at least two digits of significance. Typically, VNTOL need not be changed unless the circuit is a high voltage circuit. For 1000 volt circuits, a reasonable value can be 5 to 50 millivolts. ABSV is the same as VNTOL. The default value equals 50 (microvolts).

Speed

AUTOSTOP

Stops the transient analysis when all TRIG-TARG and FIND-WHEN measure functions are calculated. This option can result in a substantial CPU time reduction. If the data file contains measure functions such as AVG, RMS, MIN, MAX, PP, ERR, ERR1,2,3, and PARAM, then AUTOSTOP is disabled.

If you set autostop, do not use the preceding measure result. as the measured variable. Otherwise, Star-Hspice reports a warning message in the `.lis` file. It also reports *output failed* or *0* as this measure result, in the `m##` file.

BKPSIZ = x

Sets the size of the breakpoint table. The default value equals 5000.

BYPASS

Speeds up simulation by not updating the status of latent devices. Setting .OPTION BYPASS = 1 enables bypassing. BYPASS applies to MOSFETs, MESFETs, JFETs, BJTs, and diodes. The default value equals 1.

Note: Use the BYPASS algorithm cautiously. Some circuit types might not converge, and might lose accuracy in transient analysis and operating point calculations.

BYTOL = x

Specifies the tolerance for the voltage at which a MOSFET, MESFET, JFET, BJT, or diode is considered latent. Star-Hspice does not update the status of latent devices. The default value equals MBYPASSxVNTOL.

FAST Speeds up simulation by not updating the status of latent devices. This option is applicable for MOSFETs, MESFETs, JFETs, BJTs, and diodes. The default is 0.

A device is considered to be latent when its node voltage variation from one iteration to the next is less than the value of either the BYTOL control option or the BYPASSTOL element parameter. (When FAST is on, Star-Hspice sets BYTOL to different values for different types of device models.)

In addition to the FAST option, the input preprocessing time can be reduced by the options NOTOP and NOELCK. Increasing the value of the MBYPASS option or the BYTOL option setting also helps simulations run faster, but can reduce accuracy.

ITLPZ Sets the pole/zero analysis iteration limit. The default value equals 100.

MBYPASS = x Computes the default value of the BYTOL control option:

```
BYTOL = MBYPASSxVNTOL
```

Also multiplies voltage tolerance RELV. MBYPASS should be set to about 0.1 for precision analog circuits. The default value equals 1 for DVDT = 0, 1, 2, or 3. The default value equals 2 for DVDT = 4.

Timestep

ABSVAR = x Sets the limit on the maximum voltage change from one time point to the next. Used with the DVDT algorithm. If the simulator produces a convergent solution that is greater than ABSVAR, the solution is discarded, the timestep is set to a smaller value, and the solution is recalculated. This is called a timestep reversal. The default value equals 0.5 (volts).

DELMAX = x

Sets the maximum value for the internal timestep Delta. Star-Hspice automatically sets the DELMAX value based on various factors, which are listed in "Timestep Control for Accuracy" on page Chapter 1127. This means that the initial DELMAX value shown in the Star-Hspice output listing is generally not the value used for simulation.

DVDT

Allows the timestep to be adjusted based on node voltage rates of change. Choices are:

- 0 - original algorithm
- 1 - fast
- 2 - accurate
- 3,4 - balance speed and accuracy

The default value equals 4.

The ACCURATE option also increases the accuracy of the results.

FS = x

Sets the fraction of a timestep (TSTEP) that Delta (the internal timestep) is decreased for the first time point of a transient. Decreasing the FS value helps circuits that have timestep convergence difficulties. It also is used in the DVDT = 3 method to control the timestep.

$$Delta = FS \times [MIN(TSTEP, DELMAX, BKPT)]$$

where DELMAX is specified and BKPT is related to the breakpoint of the source. TSTEP is set in the .TRAN statement. The default value equals 0.25.

FT = x

Sets the fraction of a timestep (TSTEP) by which Delta (the internal timestep) is decreased for an iteration set that does not converge. It is also used in DVDT = 2 and DVDT = 4 to control the timestep. The default is 0.25.

IMIN = x, *ITL3 = x*	Determines the timestep in the algorithms used for transient analysis siimulations. IMIN sets a lower limit on the number of iterations required to obtain convergence. If the number of iterations is less than IMIN, then the internal timestep (Delta) doubles. Use this option to decrease simulation times in circuits where the nodes are stable most of the time, such as digital circuits. If the number of iterations is greater than IMIN, the timestep stays the same, unless the timestep exceeds the IMAX option. ITL3 is the same as IMIN. The default is 3.0.
IMAX = x, *ITL4 = x*	Determines the maximum timestep in the timestep algorithms used for transient analysis simulations. IMAX sets an upper limit on the number of iterations allowed, to obtain a convergent solution at a timepoint. If the number of iterations needed is greater than IMAX, the internal timestep (Delta) decreases by a factor equal to the transient control option FT, and a new solution is calculated using the new timestep. IMAX also works in conjunction with the transient control option IMIN. ITL4 is the same as IMAX. The default value equals 8.0.
ITL3 = x	Determines the timestep in the algorithms used for transient analysis simulations. IMIN sets a lower limit on the number of iterations required to obtain convergence. If the number of iterations is less than IMIN, the internal timestep (Delta) doubles. Use this option to decrease simulation times in circuits where the nodes are stable most of the time, such as digital circuits. If the number of iterations is greater than IMIN, the timestep stays the same, unless the timestep exceeds the IMAX option. ITL3 is the same as IMIN. The default value equals 3.0.

ITL4 = x

Determines the maximum timestep in the timestep algorithms used for transient analysis simulations. IMAX sets an upper limit on the number of iterations allowed to obtain a convergent solution at a timepoint. If the number of iterations needed is greater than IMAX, the internal timestep Delta is decreased by a factor equal to the transient control option FT, and a new solution is calculated using the new timestep. IMAX also works in conjunction with the transient control option IMIN. ITL4 is the same as IMAX. The default value equals 8.0.

ITL5 = x

Sets the transient analysis total iteration limit. If a circuit uses more than ITL5 iterations, the program prints all results to that point. The default allows an infinite number of iterations. The default value equals 0.0.

RELVAR = x

Used with ABSVAR and the DVDT timestep algorithm option. RELVAR sets the relative voltage change for LVLTIM = 1 or 3. If the node voltage at the current time point exceeds the node voltage at the previous time point by RELVAR, Star-Hspice reduces the timestep, and calculates a new solution at a new time point. The default value equals 0.30 (30%).

RMAX = x

Sets the TSTEP multiplier, which determines the maximum value, DELMAX, that can be used for the internal timestep Delta:

```
DELMAX = TSTEPxRMAX
```

The default value equals 5 when dvdt = 4 and lvltim = 1; otherwise, the default = 2.

RMIN = x	Sets the minimum value of Delta (internal timestep). An internal timestep smaller than RMINxTSTEP terminates the transient analysis, with the error message “internal timestep too small”. Delta decreases by the amount set in the FT option, if the circuit has not converged in IMAX iterations. The default is 1.0e-9.
SLOPETOL = x	Sets a lower limit for breakpoint table entries in a piecewise linear (PWL) analysis. If the difference in the slopes of two consecutive PWL segment is less than the SLOPETOL value, the breakpoint table entry for the point between the segments is ignored. The default is 0.5
TIMERES = x	Sets a minimum separation between breakpoint values for the breakpoint table. If two breakpoints are closer together in time than the TIMERES value, only one of them is entered in the breakpoint table. The default value equals 1 ps.

Algorithm

DVTR	Allows the use of voltage limiting in transient analysis. The default value equals 1000.
IMAX = x, ITL4 = x	Determines the maximum timestep in the timestep algorithms used for transient analysis simulations. IMAX sets an upper limit on the number of iterations allowed to obtain a convergent solution at a timepoint. If the number of iterations needed is greater than IMAX, the internal timestep Delta is decreased by a factor equal to the transient control option FT, and a new solution is calculated using the new timestep. IMAX also works in conjunction with the transient control option IMIN. ITL4 is the same as IMAX. The default value equals 8.0.

IMIN = x,
ITL3 = x

Determines the timestep in the algorithms used for transient analysis simulations. IMIN sets a lower limit on the number of iterations required to obtain convergence. If the number of iterations is less than IMIN, the internal timestep, Delta, is doubled. This option is useful for decreasing simulation times in circuits where the nodes are stable most of the time, such as digital circuits. If the number of iterations is greater than IMIN, the timestep is kept the same unless the option IMAX is exceeded (see IMAX). ITL3 is the same as IMIN. The default is 3.0.

LVLTIM = x

Selects the timestep algorithm used for transient analysis. If LVLTIM = 1, the DVDT timestep algorithm is used. If LVLTIM = 2, the local truncation error timestep algorithm is used. If LVLTIM = 3, the DVDT timestep algorithm with timestep reversal is used.

To use the GEAR method of numerical integration and linearization, select LVLTIM = 2. To use the TRAP linearization algorithm, select LVLTIM 1 or 3. Using LVLTIM = 1 (the DVDT option) helps avoid “internal timestep too small” convergence. Using LVLTIM = 1 (the DVDT option) helps avoid “internal timestep too small” nonconvergence. The local truncation algorithm (LVLTIM = 2), however, provides a higher degree of accuracy, and prevents errors propagating from time point to time point, which can sometimes result in an unstable solution. The default value equals 1.

MAXORD = x

Sets the maximum order of integration when the GEAR method is used (see METHOD). The value of x can be either 1 or 2. If MAXORD = 1, the backward Euler method of integration is used. MAXORD = 2, however, is more stable, accurate, and practical. The default is 2.0.

METHOD = name

Sets the numerical integration method for a transient analysis, to either GEAR or TRAP. To use GEAR, set METHOD = GEAR, which sets LVLTIM = 2.

Note: To change LVLTIM from 2 to 1 or 3, set LVLTIM = 1 or 3 after the METHOD = GEAR option. This overrides LVLTIM = 2, which METHOD = GEAR.sets.

TRAP (trapezoidal) integration usually results in reduced program execution time, and more accurate results. However, trapezoidal integration can introduce an apparent oscillation on printed or plotted nodes, that might not result from circuit behavior. To test this, run a transient analysis with a small timestep. If oscillation disappears, the cause was the trapezoidal method.

The GEAR method acts as a filter, removing oscillations found in the trapezoidal method. Highly nonlinear circuits, such as operational amplifiers, can require long execution times with the GEAR method. Circuits that do not converge in trapezoidal integration, often converge in GEAR. The default value is TRAP (trapezoidal).

PURETP

Sets the integration method to use for the reversal time point. The default value is 0. If you set `puretp=1`, when Star-Hspice encounters non-convergence, it uses TRAP (instead of B.E) for the reversed time point.

Use this option to help an oscillating circuit to oscillate, if the default simulation process cannot satisfy the result.

Use this option with the `method=TRAP` statement.

MU = x,
XMU = x

The coefficient for trapezoidal integration. The range for MU is 0.0 to 0.5. XMU is the same as MU. The default value equals 0.5.

XMU = x — The coefficient for trapezoidal integration. The range for MU is 0.0 to 0.5. XMU is the same as MU. The default value equals 0.5.

Input and Output

INTERP — Limits output to post-analysis tools, such as Cadence or Zuken, to only the .TRAN timestep intervals. By default, Star-Hspice outputs all convergent iterations. INTERP typically produces a much smaller *design.tr#* file.

Use INTERP = 1 with caution, when the .MEASURE statement is present. Star-Hspice computes measure statements using the postprocessing output. Reducing postprocessing output may lead to interpolation errors in measure results.

When you run a data-driven transient analysis (.TRAN DATA statement) within optimization routines, INTERP is forced to 1. Thus, the results of all measurements are at the time points of the data, in the data-driven sweep. If the measurement needs to use converged internal timesteps, such as AVG or RMS calculations, set ITRPRT = 1.

ITRPRT — Prints output variables at their internal timepoint values. Using this option can generate a long output list.

MEASFAIL

- Produces “0” into .mt#, .ms# or .ma# and prints “failed” to the listing file when measfail=0.
- Prints “failed” into .mt#, .ms# or .ma# file and the listing file when measfail=1.

Note: The default value is 1.

Use the following syntax:

```
.option measfail=1 | 0
```

MEASSORT

To help you automatically sort large numbers of `.measure` statements, use the `.option meassort` statement.

- `.option meassort=0` (the default; does not sort `.measure` statements)
- `.option meassort=1` (internally sorts `.measure` statements).

Set this option to `1` only if you use a large number of `.measure` statements, where it is important to list similar variables together, to reduce simulation run time. For a small number of `.measure` statements, turning on internal sorting might slow-down the simulation while sorting, compared to **not** sorting first.

PUTMEAS

Allows the user to control the output variables listed in the .measure statement.

The syntax is:

```
.option putmeas=0 or (1)
```

The default value is 1.

0 does not save the values of the variables, which are listed in the .measure statement, into the corresponding output file (such as .tr#, .ac# or .sw#). This option decreases the size of the output file.

1 saves the values of the variables, which are listed in the .measure statement, into the corresponding output file (such as .tr#, .ac# or sw#). This option is similar to the output of Hspice 2000.4.

UNWRAP

Displays phase results from AC analysis, in unwrapped form (with a continuous phase plot). This allows accurate calculation of group delay. Star-Hspice always computes the group delay, based on unwrapped phase results, even if you do not set the UNWRAP option.

Avant!

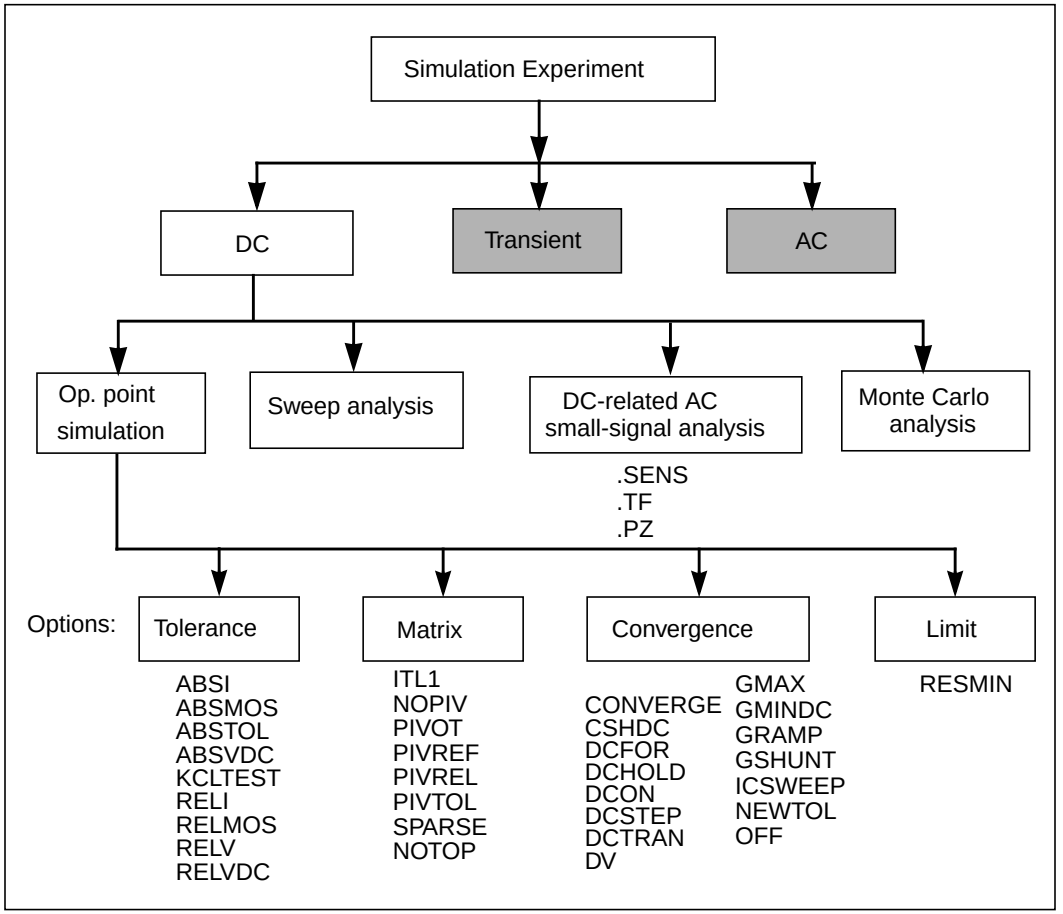
Chapter 10
Initializing DC/Operating Point Analysis

This chapter describes DC initialization and operating point analysis. It covers the following topics:

Understanding the Simulation Flow

Figure 10-1 illustrates the simulation flow for Star-Hspice.

Figure 10-1: DC Initialization and Operating Point Analysis Simulation Flow



Performing Initialization and Analysis

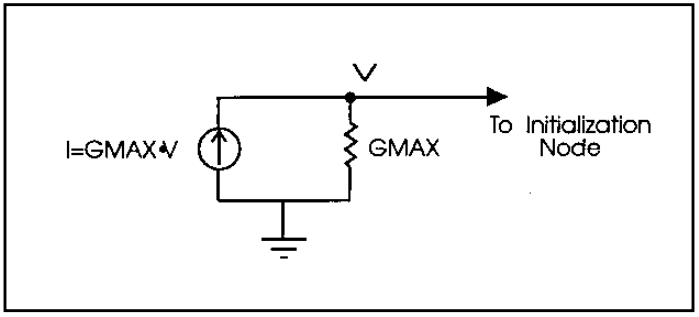
The first task Star-Hspice performs for .OP, .DC sweep, .AC, and .TRAN analyses is to set the DC operating point values for all nodes and sources. It does this either by calculating all of the values or by applying values specified in .NODESET and .IC statements or stored in an initial conditions file. The .OPTIONS OFF statement and the element parameters OFF and IC = val also control initialization.

Initialization is fundamental to the operation of simulation. Star-Hspice starts any analysis with known nodal voltages or initial estimates for unknown voltages and some branch currents, and then iteratively finds the exact solution. Initial estimates 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 the UIC parameter is specified in the .TRAN statement). The resulting DC operating point is then used as an initial estimate to solve the next timepoint in the transient analysis.

If you do not provide an initial guess, or provide only partial information, Star-Hspice provides a default estimate of each of the nodes in the circuit and then uses this estimate to iteratively find the exact solution. The .NODESET and .IC statements are two methods that supply an initial guess for the exact DC solution of nodes within a circuit. Set any circuit node to any value by using the .NODESET statement. Star-Hspice then connects a voltage source equivalent to each initialized node (a current source with a parallel conductance GMAX set with a .OPTION statement). Next, a DC operating point is calculated with the .NODESET voltage source equivalent connected. Then Star-Hspice disconnects the equivalent voltage sources set with the .NODESET statement and recalculates the DC operating point. This is considered the DC operating point solution.

Figure 10-2: Equivalent Voltage Source: NODESET and .IC



Use the .IC statement to provide both an initial guess and final solution to selected nodes within the circuit. Nodes initialized with the .IC statement become part of the solution of the DC operating point.

You can also use the OFF option to initialize active devices. The OFF option works in conjunction with .IC and .NODESET voltages as follows:

1. If any .IC or .NODESET statements exist, node voltages are set according to those statements.
2. If the OFF option is set, the terminal voltages of all active devices (BJTs, diodes, MOSFETs, JFETs, MESFETs) that are not set by .IC or .NODESET statements or by sources are set to zero.
3. If any IC parameters are specified in element statements, those initial conditions are set.
4. The resulting voltage settings are used as the initial guess at the operating point.

Use OFF to find an exact solution when performing an operating point analysis in a large circuit, where the majority of device terminals are at zero volts for the operating point solution. You can initialize the terminal voltages for selected active devices to zero by setting the OFF parameter in the element statements for those devices.

After a DC operating point has been found, use the .SAVE statement to store the operating point node voltages in a <*design*>.*ic* file. Then use the .LOAD statement to restore the operating point values from the *ic* file for subsequent analyses.

Setting Initial Conditions for Transient Analysis

If UIC is included in the .TRAN statement, a transient analysis is started 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 failure can be due to stated initial conditions that make it impossible to calculate the actual DC operating point.

Using DC Initialization and Operating Point Statements

.OP Statement — Operating Point

When an .OP statement is included in an input file, the DC operating point of the circuit is calculated. You can also use the .OP statement to produce an operating point during a transient analysis. Only one .OP statement can appear in a Star-Hspice simulation.

If an analysis is being used which requires an operating point to be calculated, the .OP statement is not required; an operating point calculation will be performed. If a .OP statement is specified and the keyword UIC exists in a .TRAN analysis statement, the time = 0 operating point analysis will be omitted and a warning issued in the output listing.

Syntax

```
.OP <format> <time> <format> <time>
```

format — Any of the following keywords. (Only the first letter is required. Default = ALL.)

- ALL
  Full operating point, including voltage, currents, conductances, and capacitances. This parameter causes voltage/current output for time specified.
- BRIEF
  Produces a one line summary of each element's voltage, current, and power. Current is stated in milliamperes and power in milliwatts.
- CURRENT
  Voltage table with element currents and power, a brief summary

- DEBUG
  Usually only invoked by the program in the event of a nonconvergent simulation. Debug prints back the nonconvergent nodes with the new voltage, old voltage, and the tolerance (degree of nonconvergence). It also prints back the nonconvergent elements with their tolerance values.
- NONE
  Inhibits node and element printouts but allows additional analysis specified to be performed
- VOLTAGE
  Voltage table only

Note: The preceding keywords are mutually exclusive; use only one at a time.

---

time — Parameter that is placed directly following ALL, VOLTAGE, CURRENT, or DEBUG and specifies the time at which the report is printed.

Example

The following example calculates operating point voltages and currents for the DC solution, as well as currents at 10 ns, and voltages at 17.5 ns, 20 ns and 25 ns for the transient analysis.

```
.OP .5NS CUR 10NS VOL 17.5NS 20NS 25NS
```

The following example calculates the complete DC operating point solution. A printout of the solution is shown below.

```
.OP
```

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
 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. The value set by IC = <*val*> is used as the DC operating point value, as in the DC solution.

Example

```
HXCC 13 20 VIN1 VIN2 IC = 0.5, 1.3
```

The example above describes an H element dependent voltage source with the current through VIN1 initialized to 0.5 mA and the current through VIN2 initialized to 1.3 mA.

.IC and .DCVOLT Initial Condition Statements

The .IC statement or the .DCVOLT statement is used to set transient initial conditions. How it initializes depends upon whether the UIC parameter is included in the .TRAN analysis statement.

When the UIC parameter is specified in the .TRAN statement, Star-Hspice does not calculate the initial DC operating point. In this case, the transient analysis is entered directly. The transient analysis uses the .IC initialization values as part of the solution for timepoint zero (a fixed equivalent voltage source is applied during the calculation of the timepoint zero). The .IC statement is equivalent to specifying the IC parameter on each element statement, but is more convenient. You can still specify the IC parameter, but it does not take precedence over values set in the .IC statement.

When the UIC parameter is *not* specified in the .TRAN statement, the DC operating point solution is computed before the transient analysis. In this case, the node voltages specified in the .IC statement are fixed for the determination of the DC operating point. For the transient analysis, the initialized nodes are released for the calculation of the second timepoint and later.

Syntax

```
.IC V(node1) = val1 V(node2) = val2 ...
```

or

```
.DCVOLT V(node1) = val1 V(node2) = val2 ...
```

where:

val1 ...	Specifies voltages. The significance of these specified voltages depends on whether the UIC parameter is specified in the .TRAN statement.
node1 ...	Node numbers or node names can include full path names or circuit numbers.

Example

```
.IC V(11) = 5 V(4) = -5 V(2) = 2.2
.DCVOLT 11 5 4 -5 2 2.2
```

.NODESET Statement

.NODESET initializes specified nodal voltages for a DC operating point analysis. The .NODESET statement often is used to correct convergence problems in DC analysis. Setting the nodes in the circuit to values that are close to the actual DC operating point solution enhances the convergence of the simulation. The simulator uses the NODESET voltages for the first iteration only.

Syntax

```
.NODESET V(node1) = val1 <V(node2) = val2 ...>
```

or

```
.NODESET node1 val1 <node2 val2>
```

node1 ...	Node numbers or node names can include full path names or circuit numbers.

Example

```
.NODESET V(5:SETX) = 3.5V V(X1.X2.VINT) = 1V
.NODESET V(12) = 4.5 V(4) = 2.23
.NODESET 12 4.5 4 2.23 1 1
```

Using .SAVE and .LOAD Statements

Star-Hspice always saves the operating point unless the .SAVE LEVEL = NONE statement is used. The saved operating-point file is restored only if the Star-Hspice input file contains a .LOAD statement.

Any node initialization commands, such as .NODESET and .IC, overwrite the initialization done through a .LOAD command if they appear in the netlist after the .LOAD command. This feature helps you to set particular states for multistate circuits such as flip-flops and still take advantage of the .SAVE command to speed up the DC convergence.

.SAVE and .LOAD continues to work even on changed circuit topologies. Adding or deleting nodes results in a new circuit topology. The new nodes are initialized as if no operating point were saved. References to deleted nodes are ignored. The coincidental nodes are initialized to the values saved from the previous run.

When nodes are initialized to voltages, Star-Hspice inserts Norton equivalent circuits at each initialized node. The conductance value of a Norton equivalent circuit is GMAX = 100. This conductance value might be too large for some circuits.

If using .SAVE and .LOAD does not speed up the simulation or causes problems with the simulation, you can use .OPTION GMAX = 1e-12 to minimize the effect of the Norton equivalent circuits on matrix conductances. Star-Hspice still uses the initialized node voltages for device initialization.

.SAVE Statement

The .SAVE statement stores the operating point of a circuit in a user-specified file. Then you can use the .LOAD statement to input the contents of this file for subsequent simulations to obtain quick DC convergence. The operating point is always saved by default, even if the Star-Hspice input file does not contain a .SAVE statement. To not save the operating point, specify .SAVE LEVEL = NONE.

You can specify that the operating point data be saved as an .IC statement or a .NODESET statement.

Syntax:

```
.SAVE <TYPE = type_keyword> <FILE = save_file>
+ <LEVEL = level_keyword> <TIME = save_time>
```

where:

type_keyword — Type of operating point storage desired. The type can be one of the following. Default: NODESET.

- .NODESET
  Stores the operating point as a .NODESET statement. In subsequent simulations, all node voltages are initialized to these values if the .LOAD statement is used. Assuming incremental changes in circuit conditions, DC convergence should be achieved in a few iterations.
- .IC
  Causes the operating point to be stored as a .IC statement. In subsequent simulations, node voltages are initialized to these values if .LOAD is included in the netlist file.

save_file — Name of the file in which the DC operating point data is stored. The default is <*design*>.*ic*.

level_keyword — Circuit level at which the operating point is saved. The level can be one of the following.
Default = ALL.

- ALL
  All nodes from the top to the lowest circuit level are saved. This option provides the greatest improvement in simulation time.
- TOP
  Only nodes in the top-level design are saved. No subcircuit nodes are saved.
- NONE
  The operating point is not saved.

save_time	Time during transient analysis at which the operating point is saved. A valid transient analysis statement is required to successfully save a DC operating point. Default = 0.

For a parameter or temperature sweep, only the first operating point is saved. For example, if the Star-Hspice input netlist file contains the statement

```
.TEMP -25 0 25
```

the operating point corresponding to .TEMP -25 is saved.

.LOAD Statement

Use the .LOAD statement to input the contents of a file stored with the .SAVE statement. Files stored with the .SAVE statement contain operating point information for the point in the analysis at which the .SAVE was executed.

Do not use the .LOAD command for concatenated netlist files.

Syntax

```
.LOAD <FILE = load_file>
```

load_file	Name of the file in which an operating point for the circuit under simulation was saved using .SAVE. The default is *<design>.ic*, where *design* is the root name of the design.

.DC Statement—DC Sweeps

The .DC statement is used in DC analysis to:

- Sweep any parameter value
- Sweep any source value
- Sweep temperature range
- Perform a DC Monte Carlo analysis (random sweep)
- Perform a DC circuit optimization
- Perform a DC model characterization

The format for the .DC statement depends on the application in which it is used, as shown in the following examples:

Syntax

Sweep or parameterized sweep:

```
.DC var1 START = start1 STOP = stop1 STEP = incr1
```

or

```
.DC var1 START = <param_expr1> STOP = <param_expr2>
+   STEP = <param_expr3>
```

or

```
.DC var1 start1 stop1 incr1 <SWEEP var2 type np start2 stop2>
```

or

```
.DC var1 start1 stop1 incr1 <var2 start2 stop2 incr2 >
```

Data driven sweep:

```
.DC var1 type np start1 stop1 <SWEEP DATA = datanm>
```

or

```
.DC DATA = datanm<SWEEP var2 start2 stop2 incr2>
```

or

```
.DC DATA = datanm
```

Monte Carlo:

```
.DC var1 type np start1 stop1 <SWEEP MONTE = val>
```

or

```
.DC MONTE = val
```

Optimization:

```
.DC DATA = datanm OPTIMIZE = opt_par_fun
+   RESULTS = measnames MODEL = optmod
```

or

```
.DC var1 start1 stop1 SWEEP OPTIMIZE = OPTxxx
+   RESULTS = measname MODEL = optmod
```

The .DC statement keywords and parameters have the following descriptions:

DATA = datanm	Datanm is the reference name of a .DATA statement.
incr1 …	Voltage, current, element, model parameters, or temperature increment values
MODEL	Specifies the optimization reference name used in the .MODEL OPT statement used in an optimization analysis
MONTE = val	Produces a number *val* of randomly generated values, which are used to select parameters from a distribution. The distribution can be *Gaussian*, *Uniform*, or *Random Limit.*
np	Number of points per decade or per octave, or just number of points depending on the preceding keyword.
OPTIMIZE	Specifies the parameter reference name used for optimization in the .PARAM statement
RESULTS	Specifies the measure name used for optimization in the .MEASURE statement

start1 …	Starting voltage, current, element, model parameters, or temperature values If type variation “POI” (list of points) is used, a list of parameter values, instead of “start stop” is specified.
stop1 …	Final voltage, current, any element, model parameter, or temperature values
SWEEP	Keyword to indicate a second sweep has different type of variation (DEC, OCT, LIN, POI, DATA statement, or MONTE = val)
TEMP	Keyword to indicate a temperature sweep
type	Can be any of the following keywords: DEC — decade variation OCT — octave variation LIN — linear variation POI — list of points
var1 …	Name of an independent voltage or current source, any element or model parameter, or the keyword TEMP (indicating a temperature sweep). Star-Hspice supports source value sweep, referring to the source name (SPICE style). However, if parameter sweep, a .DATA statement, and temperature sweep are selected, a parameter name must be chosen for the source value and subsequently referred to in the .DC statement. The parameter name can not start with V or I.

Example

The following example causes the value of the voltage source VIN to be swept from 0.25 volts to 5.0 volts in increments of 0.25 volts.

```
.DC VIN 0.25 5.0 0.25
```

The following example invokes a sweep of the drain-to-source voltage from 0 to 10 V in 0.5 V increments at VGS values of 0, 1, 2, 3, 4, and 5 V.

```
.DC VDS 0 10 0.5 VGS 0 5 1
```

The following example asks for a DC analysis of the circuit from -55°C to 125°C in 10°C increments.

```
.DC TEMP -55 125 10
```

As a result of the following script, a DC analysis is conducted at five temperatures: 0, 30, 50, 100°and 125°C.

```
.DC TEMP POI 5 0 30 50 100 125
```

In the following example, a DC analysis is performed on the circuit at each temperature value, which results from a linear temperature sweep from 25°C to 125°C (five points), sweeping a resistor value called xval from 1 k to 10 k in 0.5 k increments.

```
.DC xval 1k 10k .5k SWEEP TEMP LIN 5 25 125
```

The example below specifies a sweep of the value *par1* from 1 k to 100 k by 10 points per decade.

```
.DC DATA = datanm SWEEP par1 DEC 10 1k 100k
```

The next example also requests a DC analysis at specified parameters in the .DATA statement referenced by the .DATA statement reference name *datanm*. Parameter par1 also is swept from 1k to 100k by 10 points per decade.

```
.DC par1 DEC 10 1k 100k SWEEP DATA = datanm
```

The final example invokes a DC sweep of the parameter *par1* from 1k to 100k by 10 points per decade, using 30 randomly generated (Monte Carlo) values.

```
.DC par1 DEC 10 1k 100k SWEEP MONTE = 30
```

Schmitt Trigger Example

```
*file: bjtschmt.sp      bipolar schmitt trigger
.options post = 2
vcc 6 0 dc 12
vin 1 0 dc 0 pwl(0,0 2.5u,12 5u,0)
cb1 2 4 .1pf
rc1 6 2 1k
rc2 6 5 1k
rb1 2 4 5.6k
rb2 4 0 4.7k
re 3 0 .47k
*
diode 0 1 dmod
q1 2 1 3 bmod 1 ic = 0,8
q2 5 4 3 bmod 1 ic = .5,0.2
*
.dc vin 0,12,.1
*
.model dmod d is = 1e-15 rs = 10
.model bmod npn is = 1e-15 bf = 80 tf = 1n
+ cjc = 2pf cje = 1pf rc = 50 rb = 100 vaf = 200
.plot v(1) v(5)
.graph dc model = schmittplot input = v(1)
+ output = v(5) 4.0 5.0
.model schmittplot plot xscal = 1 yscal = 1 xmin = .5u
+ xmax = 1.2u
.end
```

Using Other DC Analysis Statements

Star-Hspice provides the following additional DC analysis statements. Each of these statements uses the DC equivalent model of the circuit for its analysis functions. For .PZ, capacitors and inductors are included in the equivalent circuit.

.PZ	Performs pole/zero analysis (.OP specification is not required)
.SENS	Obtains the DC small-signal sensitivities of specified output variables with respect to circuit parameters (.OP specification is not required)
.TF	Calculates the DC small-signal value of a transfer function (the ratio of an output variable to an input source). An .OP specification is not required.

Star-Hspice also provides a set of DC control options and DC initialization statements that allow for the modeling of resistive parasitics and the initialization of nodes. These enhance the convergence properties and the accuracy of the simulation. This section describes how to perform DC-related small signal analysis.

.SENS Statement — DC Sensitivity Analysis

If a .SENS statement is included in the input file, Star-Hspice determines the DC small-signal sensitivities of each specified output variable relative to every circuit parameter. The sensitivity measurement is the partial derivative of each output variable with respect to the value of a given circuit element, taken at the operating point and normalized to the total change in output magnitude. Therefore, the sum of the sensitivities of all elements is 100%. Sensitivities are calculated for resistors, voltage sources, current sources, diodes, BJTs, and MOSFETs (Level49 and Level53, Version=3.22).

You can perform only one .SENS analysis per simulation. If more than one .SENS statement is present, only the last one is run.

Syntax

```
.SENS ov1 <ov2 ...>
```

ov1 ov2 …	Branch currents or nodal voltage for DC component sensitivity analysis

Example

```
.SENS V(9) V(4,3) V(17) I(VCC)
```

Note: The .SENS statement can generate very large amounts of output for large circuits.

.TF Statement — DC Small-Signal Transfer Function Analysis

The transfer function statement (.TF) defines the small-signal output and input for DC small-signal analysis. When the .TF statement is included, Star-Hspice computes the DC small-signal value of the transfer function (output/input), input resistance, and output resistance.

Syntax

```
.TF ov srcnam
```

where:

ov	Small-signal output variable
srcnam	Small-signal input source

Example

```
.TF V(5,3) VIN
.TF I(VLOAD) VIN
```

For the first example, Star-Hspice computes the ratio of V(5,3) to VIN, the small-signal input resistance at VIN, to the small-signal output resistance measured across nodes 5 and 3. Only one .TF statement can be used per simulation. If more than one .TF statement is present, only the last is performed.

.PZ Statement— Pole/Zero Analysis

Syntax

```
.PZ ov srcnam
```

where:

ov	Output variable: a node voltage V(n), or branch current I(element)
srcnam	Input source: an independent voltage or current source name

Example

```
.PZ   V(10)   VIN
.PZ   I(RL)   ISORC
```

See Performing Pole/Zero Analysis on page 18-1 for complete information about pole/zero analysis.

Setting DC Initialization Control Options

The DC operating point analysis control options control the DC convergence properties, as well as simulation algorithms. Many of these options also affect transient analysis because DC convergence is an integral part of transient convergence. The absolute and relative voltages, the current tolerances, and the matrix options should be considered for both DC and transient convergence.

Options are specified in .OPTIONS statements. The .OPTIONS statement is discussed in Specifying Simulation Options on page 9-1.

The following options are associated with controlling DC analysis. They are described in this section.

ABSTOL	GRAMP	NOPIV
CAPTAB	GSHUNT	OFF
CSHDC	ICSWEEP	PIVOT
DCCAP	ITL1	PIVREF
DCFOR	ITL2	PIVREL
DCHOLD	KCLTEST	PIVTOL
DCSTEP	MAXAMP	RESMIN
DV	NEWTOL	SPARSE

Some of these options also are used in DC and AC analysis. Many of these options also affect the transient analysis, because DC convergence is an integral part of transient convergence. Transient analysis is discussed in “Performing Transient Analysis” on page Chapter 111.

Option Descriptions

ABSTOL = x — Sets the absolute node voltage error tolerance for DC and transient analysis. Decrease ABSTOL if accuracy is more important than convergence time.

CAPTAB — Prints table of single plate nodal capacitance for diodes, BJTs, MOSFETs, JFETs and passive capacitors at each operating point.

CSHDC	The same option as CSHUNT, but is used only with option CONVERGE.
DCCAP	Generates C-V plots and prints out the capacitance values of a circuit (both model and element) during a DC analysis. C-V plots are often generated using a DC sweep of the capacitor. Default = 0 (off).
DCFOR = x	Used in conjunction with the DCHOLD option and the .NODESET statement to enhance the DC convergence properties of a simulation. DCFOR sets the number of iterations that are to be calculated after a circuit converges in the steady state. Since the number of iterations after convergence is usually zero, DCFOR adds iterations (and computational time) to the calculation of the DC circuit solution. DCFOR helps ensure that a circuit has actually, not falsely, converged. Default = 0.
DCHOLD = x	DCFOR and DCHOLD are used together for initialization in a DC analysis. They enhance the convergence properties of a DC simulation. DCFOR and DCHOLD work together with the .NODESET statement. The DCHOLD option specifies how many iterations to hold a node at the voltage values specified in the .NODESET statement. The effects of DCHOLD on convergence differ according to the DCHOLD value, and the number of iterations needed to obtain DC convergence.

If a circuit converges in the steady state in fewer than DCHOLD iterations, the DC solution includes the values set in the .NODESET statement. However, if the circuit requires more than DCHOLD iterations to converge, Star-Hspice ignores the values set in the .NODESET statement, and calculates the DC solution with the .NODESET fixed source voltages open circuited. Default = 1.

DCSTEP = x

Converts DC model and element capacitors to a conductance, to enhance DC convergence properties. The value of the element capacitors are all divided by DCSTEP, to obtain a DC conductance model. Default = 0 (seconds).

DV = x

The maximum iteration-to-iteration voltage change for all circuit nodes in both DC and transient analysis. Values of 0.5 to 5.0 can be necessary for some high-gain bipolar amplifiers to achieve a stable DC operating point. CMOS circuits frequently require a value of about 1 volt for large digital circuits. Default = 1000 (or 1e6 if DCON = 2).

GRAMP = x

Star-Hspice sets the value during auto-convergence. GRAMP is used in conjunction with the GMINDC convergence control option to find the smallest value of GMINDC that results in DC convergence. GMINDC is described in Convergence Control Option Descriptions on page 10-36.

GRAMP specifies the conductance range over which GMINDC is to be swept during a DC operating point analysis. Star-Hspice substitutes values of GMINDC over this range and simulates at each value. It then picks the lowest value of GMINDC that resulted in the circuit converging in the steady state.

If GMINDC is swept between 1e-12 mhos (the default) and 1e-6 mhos, GRAMP is set to 6 (the value of the exponent difference between the default and the maximum conductance limit). In this case, GMINDC is first set to 1e-6 mhos, and the circuit is simulated. If convergence is achieved, GMINDC is next set to 1e-7 mhos, and the circuit is simulated again. The sweep continues until a simulation has been performed at all values on the GRAMP ramp. If the combined conductance of GMINDC and GRAMP is greater than 1e-3 mho, a false convergence can occur. Default = 0.

GSHUNT Conductance added from each node to ground. The default is zero. Add a small GSHUNT to each node to help solve "Timestep too small" problems caused by high frequency oscillations or numerical noise.

ICSWEEP For a parameter or temperature sweep, saves the results of the current analysis for use as the starting point in the next analysis in the sweep. When ICSWEEP = 1, the current results are used in the next analysis. When ICSWEEP = 0, the results of the current analysis are not used in the next analysis. Default = 1.

ITL1 = x Sets the maximum DC iteration limit. Increasing this value is unlikely to improve convergence for small circuits. Values as high as 400 have resulted in convergence for certain large circuits with feedback, such as operational amplifiers and sense amplifiers. Something is usually wrong with a model if more than 100 iterations are required for convergence. Set .OPTION ACCT to obtain a listing of how many iterations are required for an operating point. Default = 200.

ITL2 = val

Sets the DC transfer curve iteration limit. Increasing the iteration limit can be effective in improving convergence only on very large circuits.
Default = 50.

KCLTEST

Activates the KCL test (Kirchhoff's Current Law) function. This test results in a longer simulation time, especially for large circuits, but provides a very accurate check of the solution. Default = 0.

When set to 1, Star-Hspice sets these options:

- RELMOS and ABSMOS options are set to 0 (off).
- ABSI is set to 1e-16 A.
- RELI is set to 1e-6.

To satisfy the KCL test, the following condition must be satisfied for each node:

$$\left|\Sigma i_b\right| < RELI \cdot \Sigma\left|i_b\right| + ABSI$$

where the i_bs are the node currents.

MAXAMP = x

Sets the maximum current through voltage defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, an error is issued. Default = 0.0.

NEWTOL

Calculates one more iterations past convergence for every DC solution and timepoint circuit solution calculated. When NEWTOL is not set, once convergence is determined, the convergence routine is ended and the next program step begun.
Default = 0.

NOPIV

Prevents Star-Hspice from switching automatically to pivoting matrix factorization when a nodal conductance is less than PIVTOL. NOPIV inhibits pivoting. Also see PIVOT.

OFF

Initializes the terminal voltages of all active devices to zero if they are not initialized to other values. For example, if the drain and source nodes of a transistor are not both initialized using .NODESET or .IC statements or by connecting them to sources, then the OFF option initializes all of the nodes of the transistor to zero. The OFF option is checked before element IC parameters, so if an element IC parameter assignment exists for a particular node, the node is initialized to the element IC parameter value even if it was previously set to zero by the OFF option. (The element parameter OFF can be used to initialize the terminal voltages to zero for particular active devices).

The OFF option is used to help find exact DC operating point solutions for large circuits.

PIVOT = x

Provides different pivoting algorithm selections. These can be used effectively to reduce simulation time and achieve convergence in circuits that produce hard-to-solve matrix equations. The pivot algorithm is selected by setting PIVOT to one of the following values:

- 0 Original nonpivoting algorithm
- 1 Original pivoting algorithm
- 2 Pick largest pivot in row algorithm
- 3 Pick best in row algorithm
- 10 Fast nonpivoting algorithm, more memory required
- 11 Fast pivoting algorithm, more memory required than for PIVOT values less than 11

- 12 Pick largest pivot in row algorithm, more memory required than for PIVOT values less than 12
- 13 Fast best pivot: faster, more memory required than for PIVOT values less than 13

Default = 10.

The fastest algorithm is PIVOT = 13, which can improve simulation time by up to ten times on very large circuits. However, the PIVOT = 13 option requires substantially more memory for the simulation. Some circuits with large conductance ratios, such as switching regulator circuits, might need pivoting. If PIVTOT = 0, Star-Hspice automatically changes from nonpivoting to a row pivot strategy upon detection of any diagonal matrix entry less than PIVTOL. This strategy provides the time and memory advantages of nonpivoting inversion, while avoiding unstable simulations and incorrect results. Use .OPTION NOPIV to prevent pivoting from being used under any circumstances.

For very large circuits, PIVOT = 10, 11, 12, or 13 can require excessive memory.

If Star-Hspice switches to pivoting during a simulation, the message “pivot change on the fly” is printed, followed by the node numbers causing the problem. Use .OPTION NODE to obtain a node-to-element cross reference.

SPARSE is the same as PIVOT.

PIVREF Pivot reference. Used in PIVOT = 11, 12, 13 to limit the size of the matrix. Default = 1e+8.

PIVREL = x

Sets the maximum/minimum row/matrix ratio. Use only for PIVOT = 1. Large values for PIVREL can result in very long matrix pivot times. If the value is too small, however, no pivoting occurs. It is best to start with small values of PIVREL, using an adequate but not excessive value for convergence and accuracy. Default = 1E-20 (max = 1e-20, min = 1).

PIVTOL = x

Sets the absolute minimum value for which a matrix entry is accepted as a pivot. PIVTOL is used as the minimum conductance in the matrix when PIVOT = 0. Default = 1.0e-15.

Note: PIVTOL must be less than GMIN or GMINDC. Values approaching 1 yield increased pivot.

RESMIN = x

Specifies the minimum resistance for all resistors, including parasitic and inductive resistances. Default = 1e-5 (ohm). Range: 1e-15 to 10 ohm.

SPARSE = x

Provides different pivoting algorithm selections. You can use these to reduce simulation time, and achieve convergence in circuits that produce hard-to-solve matrix equations. To select the pivot algorithm, set PIVOT to one of the following values:

- 0 Original nonpivoting algorithm
- 1 Original pivoting algorithm
- 2 Pick largest pivot in row algorithm
- 3 Pick best in row algorithm
- 10 Fast nonpivoting algorithm, more memory required
- 11 Fast pivoting algorithm, more memory required than for PIVOT values less than 11

- 12 Pick largest pivot in row algorithm, more memory required than for PIVOT values less than 12
- 13 Fast best pivot: faster, more memory required than for PIVOT values less than 13

Default = 10.

The fastest algorithm is PIVOT = 13, which can improve simulation time by up to ten times on very large circuits. However, the PIVOT = 13 option requires substantially more memory for the simulation. Some circuits with large conductance ratios, such as switching regulator circuits, might need pivoting. If PIVTOT = 0, Star-Hspice automatically changes from nonpivoting to a row pivot strategy upon detection of any diagonal matrix entry less than PIVTOL. This strategy provides the time and memory advantages of nonpivoting inversion, while avoiding unstable simulations and incorrect results. Use .OPTION NOPIV to prevent pivoting from being used under any circumstances.

For very large circuits, PIVOT = 10, 11, 12, or 13 can require excessive memory.

If Star-Hspice switches to pivoting during a simulation, the message “pivot change on the fly” is printed, followed by the node numbers causing the problem. Use .OPTION NODE to obtain a node-to-element cross reference.

SPARSE is the same as PIVOT.

Pole/Zero Analysis Options

Control options are set using the .OPTIONS statement. The following are the control options used in pole/zero analysis.

CSCAL Sets the capacitance scale. Capacitances are multiplied by CSCAL. Default = 1e+12.

FMAX Sets the limit for maximum pole and zero frequency value. Default = 1.0e+12 · FSCAL.

FSCAL Sets the frequency scale. Frequency is multiplied by FSCAL. Default = 1e-9.

GSCAL Sets the conductance scale. Conductances are multiplied by GSCAL. Resistances are divided by GSCAL. Default = 1e+3.

ITLPZ Sets the pole/zero analysis iteration limit. Default = 100.

LSCAL Sets inductance scale. Inductances are multiplied by LSCAL. Default = 1e+6.

The scale factors must satisfy the following relations:

$$GSCA = CSCAL \cdot FSCAL$$

$$GSCAL = \frac{1}{LSCAL} \cdot FSCAL$$

If you change scale factors, it might be necessary to modify the initial Muller points (X0R, X0I), (X1R, X1I) and (X2R, X2I), even though Star-Hspice multiplies initial values by (1e-9/GSCAL).

PZABS Sets absolute tolerances for poles and zeros. This option affects the low frequency poles or zeros. It is used as follows:

$$\text{If } (|X_{real}| + |X_{imag}| < PZABS),$$

$$\text{then } X_{real} = 0 \text{ and } X_{imag} = 0 .$$

It is also used for convergence tests. Default = 1e-2.

PZTOL Sets the relative error tolerance for poles or zeros. Default = 1.0e-6.

RITOL Sets the minimum ratio value for (real/imaginary) or (imaginary/real) parts of the poles or zeros. RITOL is used as follows:

$$\text{If } |X_{imag}| \le RITOL \cdot |X_{real}| \text{ , then } X_{imag} = 0$$

$$\text{If } |X_{real}| \le RITOL \cdot |X_{imag}| \text{, then } X_{real} = 0$$

Default = 1.0e-6.

(X0R,X0I),
(X1R,X1I),
(X2R,X2I)

The three complex starting points in the Muller pole/zero analysis algorithm are:

```
X0R = -1.23456e6  X0I = 0.0
X1R = -1.23456e5  X1I = 0.0
X2R = +.23456e6  X2I = 0.0
```

These initial points are multiplied by FSCAL.

Specifying Accuracy and Convergence

Convergence is defined as the ability to obtain a solution to a set of circuit equations within a given tolerance criteria. In numerical circuit simulation, the designer specifies a relative and absolute accuracy for the circuit solution and the simulator iteration algorithm attempts to converge onto a solution that is within these set tolerances.

Accuracy Tolerances

Star-Hspice uses accuracy tolerance specifications to help assure convergence by determining whether or not to exit the convergence loop. For each iteration of the convergence loop, Star-Hspice takes the value of the previously calculated solution and subtracts it from the present solution, then compares this result with the accuracy tolerances.

$$| Vn^{k} - Vn^{k-1} | < = \text{accuracy tolerance}$$

where:

- Vn^{k} is the solution at timepoint n and iteration k
- Vn^{k-1} is the solution at timepoint n and iteration k - 1

Absolute and Relative Accuracy Tolerances

As shown in Table 10-1, Star-Hspice defaults to specific absolute and relative values. You can change these tolerance levels so that simulation time is not excessive and accuracy is not compromised. The options in the table are described in the following section.

Table 10-1: Absolute and Relative Accuracy Tolerances

Type	Option	Default
Nodal Voltage Tolerances	ABSVDC	50 µv
	RELVDC	.001
Current Element Tolerances	ABSI	1 nA
	RELI	.01
	ABSMOS	1 uA
	RELMOS	.05

Nodal voltages and element currents are compared to the values from the previous iteration. If the absolute value of the difference is less than ABSVDC or ABSI, the node or element is considered to be convergent. ABSV and ABSI set the floor value below which values are ignored. Values above the floor use the relative tolerances of RELVDC and RELI. If the iteration-to-iteration absolute difference is less than these tolerances, then it is considered to be convergent. ABSMOS and RELMOS are the tolerances for MOSFET drain currents.

The number of iterations required is directly affected by the value of the accuracy settings. If the accuracy tolerances are tight, a longer time is required to converge. If the accuracy setting is too loose, the resulting solution can be inaccurate and unstable.

Table 10-2 shows an example of the relationship between the RELVDC value and the number of iterations.

Table 10-2: RELV vs. Accuracy and Simulation Time for 2 Bit Adder

RELVDC	Iteration	Delay (ns)	Period (ns)	Fall time (ns)
.001	540	31.746	14.336	1.2797
.005	434	31.202	14.366	1.2743
.01	426	31.202	14.366	1.2724
.02	413	31.202	14.365	1.3433
.05	386	31.203	14.365	1.3315
.1	365	31.203	14.363	1.3805
.2	354	31.203	14.363	1.3908
.3	354	31.203	14.363	1.3909
.4	341	31.202	14.363	1.3916
.4	344	31.202	14.362	1.3904

Accuracy Control Options

Star-Hspice is shipped with control option settings designed to maximize accuracy without significantly degrading performance. The options and their settings are discussed in “Controlling Simulation Speed and Accuracy” on page Chapter 1126.

Convergence Control Option Descriptions

The options listed below are described in this section.

ABSH	DCON	RELH
ABSI	DCTRAN	RELI
ABSMOS	DI	RELMOS
ABSVDC	GMAX	RELV
CONVERGE	GMINDC	RELVDC

ABSH = x

Sets the absolute current change through voltage defined branches (voltage sources and inductors). In conjunction with DI and RELH, ABSH is used to check for current convergence. Default = 0.0.

ABSI = x

Sets the absolute branch current error tolerance in diodes, BJTs, and JFETs during DC and transient analysis. Decrease ABSI if accuracy is more important than convergence time.

If you want an analysis with currents less than 1 nanoamp, change ABSI to a value at least two orders of magnitude smaller than the minimum expected current.

Default: 1e-9 for KCLTEST = 0, 1e-16 for KCLTEST = 1

ABSMOS = x

Current error tolerance, for MOSFET devices in DC and transient analysis. Star-Hspice uses the ABSMOS setting to determine if the drain-to-source current solution has converged. If the difference between the last and the present iteration's drain-to-source current is less than ABSMOS (or if it is greater than ABSMOS, but the percent change is less than RELMOS), the drain-to-source current has converged. Star-Hspice then checks the other accuracy tolerances and, if all indicate convergence, the circuit solution at that timepoint is solved, and the next timepoint solution is calculated. For low power circuits, optimization, and single transistor simulations, set ABSMOS = 1e-12. Default = 1e-6 (amperes).

ABSVDC = x

Sets the absolute minimum voltage for DC and transient analysis. Decrease ABSVDC if accuracy is of more concern than convergence. If voltages less than 50 microvolts are required, ABSVDC can be reduced to two orders of magnitude less than the smallest desired voltage. This ensures at least two digits of significance. Typically ABSVDC need not be changed unless the circuit is a high voltage circuit. For 1000-volt circuits, a reasonable value can be 5 to 50 millivolts. Default = VNTOL (VNTOL default = 50 μV).

CONVERGE Invokes different methods for solving nonconvergence problems

CONVERGE = -1
together with DCON = -1, disables autoconvergence

CONVERGE = 1
uses the Damped Pseudo Transient Algorithm. If the simulation fails to converge within the amount of CPU time set by the CPTIME control option, the simulation halts.

CONVERGE = 2
uses a combination of DCSTEP and GMINDC ramping

CONVERGE = 3
invokes the source stepping method

Even if it is not set in an .OPTIONS statement, the CONVERGE option is activated in the event of a matrix floating point overflow, or a timestep too small error. Default = 0.

In the event of a matrix floating point overflow, Star-Hspice sets CONVERGE = 1.

DCON = x

In the case of convergence problems, Star-Hspice automatically sets DCON = 1 and the following calculations are made:

$$DV = max\left(0.1, \frac{V_{max}}{50}\right), \text{ if } DV = 1000$$

$$GRAMP = max\left(6, \log_{10}\left(\frac{I_{max}}{GMINDC}\right)\right)$$

$$ITL1 = ITL1 + 20 \cdot GRAMP$$

where V_{max} is the maximum voltage and I_{max} is the maximum current.

If convergence problems still exist, Star-Hspice sets DCON = 2, which is the same as the above except DV = 1e6. The above calculations are used for DCON = 1 or 2. DCON = 1 is automatically invoked if the circuit fails to converge. DCON = 2 is invoked if DCON = 1 fails.

If the circuit contains uninitialized flip-flops or discontinuous models, the simulation might be unable to converge. Setting DCON = -1 and CONVERGE = -1 disables the autoconvergence algorithm and provides a list of nonconvergent nodes and devices.

DCTRAN

DCTRAN is an alias for CONVERGE. See CONVERGE.

DI = x

Sets the maximum iteration to iteration current change through voltage defined branches (voltage sources and inductors). This option is only applicable when the value of the ABSH control option is greater than 0. Default = 0.0.

GMAX = x

The conductance in parallel with the current source used for .IC and .NODESET initialization conditions circuitry. Some large bipolar circuits can require GMAX set to 1 for convergence. Default = 100 (mho).

GMINDC = x

A conductance that is placed in parallel with all pn junctions and all MOSFET nodes except gate (see Figure 6-4 for details) for DC analysis. GMINDC helps overcome DC convergence problems caused by low values of off conductance for pn junctions and MOSFET devices. GRAMP can be used to reduce GMINDC by one order of magnitude for each step. GMINDC can be set between 1e-4 and PIVTOL. Default = 1e-12.

Large values of GMINDC can cause unreasonable circuit response. If large values are required for convergence, a bad model or circuit is suspect. In the event of a matrix floating point overflow, if GMINDC is 1.0e-12 or less, Star-Hspice sets it to 1.0e-11.

GMINDC is manipulated by Star-Hspice in autoconverge mode, as described in the Autoconverge Process on page 10-42.

RELH = x

Sets relative current tolerance through voltage defined branches (voltage sources and inductors). It is used to check current convergence. This option is applicable only if the value of the ABSH control option is greater than zero. Default = 0.05.

RELI = x Sets the relative error tolerance for current from iteration-to-iteration to determine convergence for all currents in diode, BJT, and JFET devices. (RELMOS sets the tolerance for MOSFETs). This is the change in current from the value calculated at the previous timepoint. Default = 0.01 for KCLTEST = 0, 1e-6 for KCLTEST = 1.

RELMOS = x Sets the relative drain-to-source current error tolerance from iteration-to-iteration to determine convergence for currents in MOSFET devices. (RELI sets the tolerance for other active devices.) This is the change in current from the value calculated at the previous timepoint. RELMOS is only considered when the current is greater than the floor value, ABSMOS. Default = 0.05.

RELV = x Sets the relative error tolerance for voltages. When voltages or currents exceed their absolute tolerances, the RELV test is used to determine convergence. Increasing RELV increases the relative error. In general, RELV should be left at its default value. RELV controls simulator charge conservation. For voltages, RELV is the same as RELTOL. Default = 1e-3.

RELVDC = x Sets the relative error tolerance for voltages. When voltages or currents exceed their absolute tolerances, the RELVDC test is used to determine convergence. Increasing RELVDC increases the relative error. In general, RELVDC should be left at its default value. RELVDC controls simulator charge conservation. Default = RELTOL (RELTOL default = 1e-3).

Autoconverge Process

If convergence is not achieved in the number of iterations set by ITL1, Star-Hspice initiates an autoconvergence process, in which it manipulates DCON, GRAMP, and GMINDC, as well as CONVERGE in some cases. The autoconverge process is illustrated in Figure 10-3.

Notes:

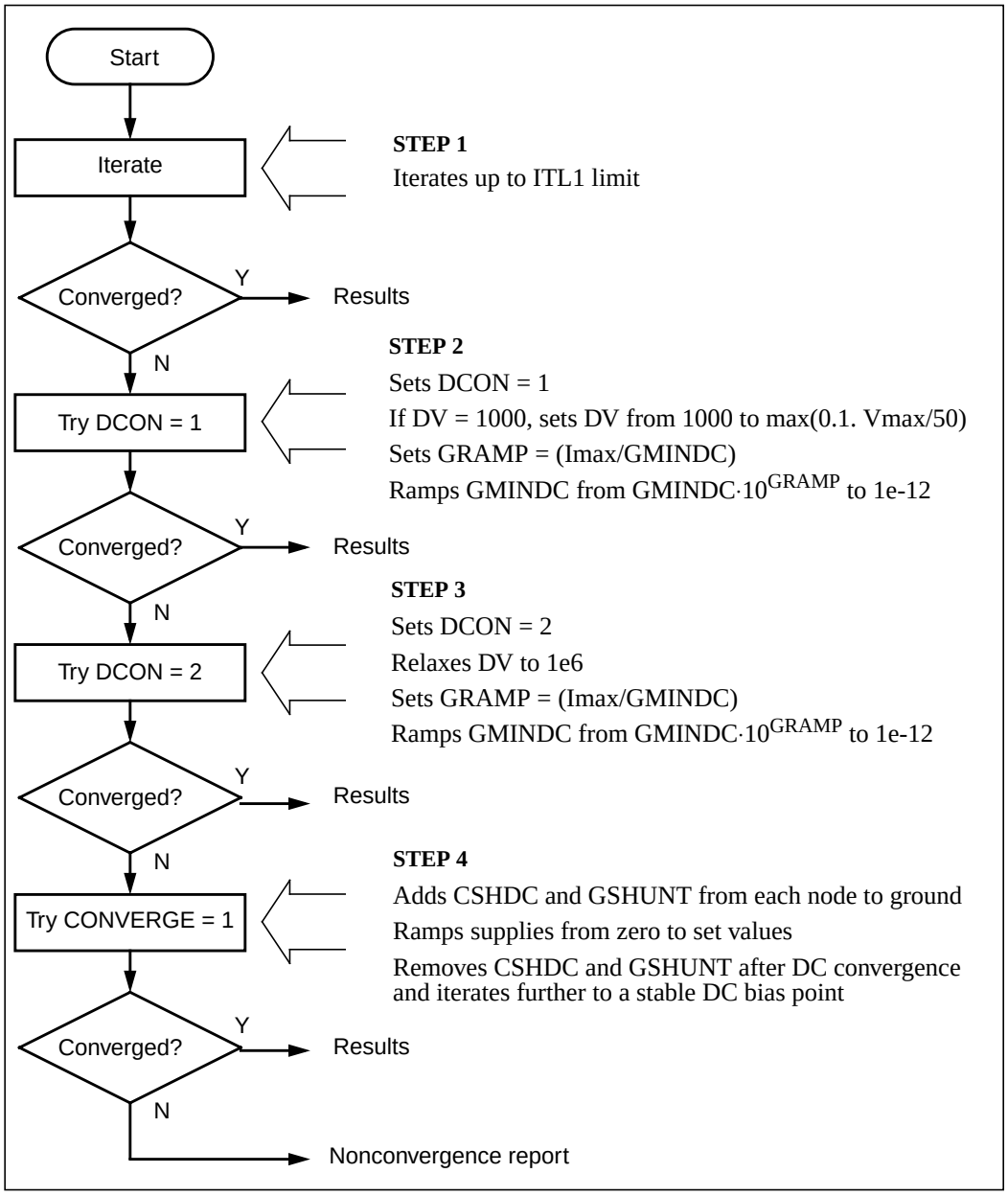
1. Setting .OPTIONS DCON = -1 disables steps 2 and 3.
2. Setting .OPTIONS CONVERGE = -1 disables step 4.
3. Setting .OPTIONS DCON = -1 CONVERGE = -1 disables steps 2, 3, and 4.
4. If you set the DV option to a value different from the default value, the value
   you set for DV is used in step 2, but DV is changed to 1e6 in step 3.
5. Setting GRAMP in an .OPTIONS statement has no effect on the autoconverge process. The autoconverge process sets GRAMP independently.
6. If you specify a value for GMINDC in an .OPTIONS statement, GMINDC is ramped to the value you set instead of to 1e-12 in steps 2 and 3.

DCON and GMINDC

GMINDC is important in stabilizing the circuit during DC operating point analysis. For MOSFETs, GMINDC helps stabilize the device in the vicinity of the threshold region. GMINDC is inserted between drain and bulk, source and bulk, and drain and source. The drain to source GMINDC helps linearize the transition from cutoff to weakly on, helps smooth out model discontinuities, and compensates for the effects of negative conductances.

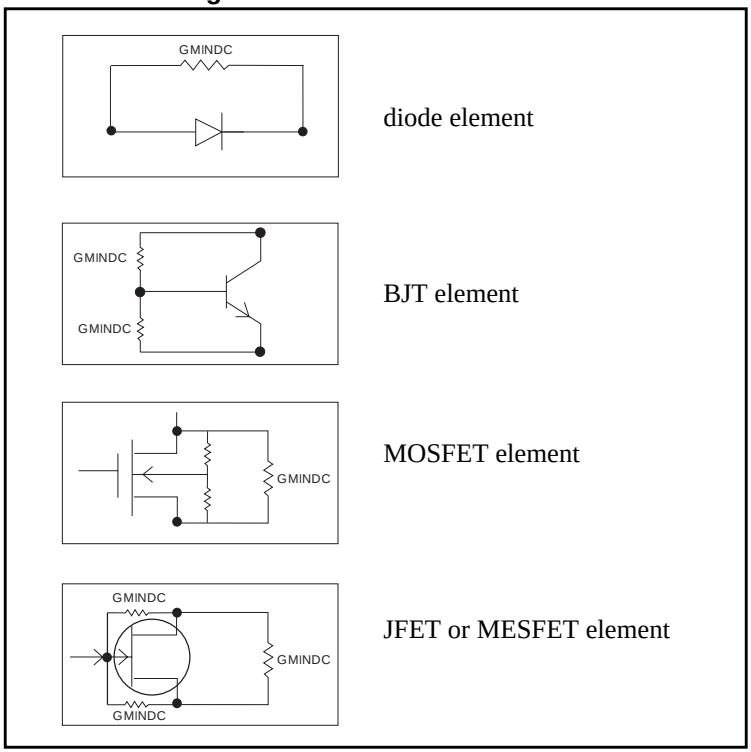
The pn junction insertion of GMINDC in junction diodes linearizes the low conductance region so that the device behaves like a resistor in the low conductance region. This prevents the occurrence of zero conductance and improves the convergence of the circuit.

Figure 10-3: Autoconvergence Process Flow Diagram



DCON is an option that Star-Hspice sets automatically in case of nonconvergence. It invokes the GMINDC ramping process in steps 2 and 3 in Figure 10-3. GMINDC is shown for various elements in Figure 10-4.

Figure 10-4: GMINDC Insertion



Reducing DC Errors

You can reduce DC errors by performing the following steps.

1. Check topology, set .OPTION NODE to get a nodal cross reference listing if you are in doubt.
   - Are all MOS p-channel substrates connected to VCC or positive supplies?
   - Are all MOS n-channel substrates connected to GND or negative supplies?
   - Are all vertical NPN substrates connected to GND or negative supplies?
   - Are all lateral PNP substrates connected to negative supplies?
   - Do all latches have either an OFF transistor or a .NODESET or an .IC on one side?
   - Do all series capacitors have a parallel resistance, or is .OPTION DCSTEP set?
2. Check your .MODEL statements.
   - Be sure to check your model parameter units. Use model printouts to verify actual values and units, since some model parameters are multiplied by scaling options.
   - Do MOS models have subthreshold parameters set with reasonable value (such as NFS = 1e11 for SPICE models 1, 2, and 3 and N0 = 1.0 for Star-Hspice models BSIM1, BSIM2, and Level 28)?
   - Avoid setting UTRA in MOS Level 2 models.
   - Are JS and JSW set in MOS model for DC portion of diode model? A typical JS value is 1e-4A/M^2.
   - Are CJ and CJSW set in MOS diode model?
   - Do JFET and MESFET models have weak inversion NG and ND set?
   - If MOS Level 6 LGAMMA equation is used, is UPDATE = 1?
   - DIODE models should have nonzero 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 because they do not ramp up with the supply voltages and can force reverse bias conditions, leading to excessive nodal voltages.
   - Schmitt triggers are unpredictable for DC sweep and sometimes for operating points for the same reasons oscillators and flip-flops are. 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 and 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 with a voltage-variable resistor or a resistor with a large AC value for AC analysis.
4. Check your options:
   - Remove all convergence-related options and try first with no special options settings.
   - Check nonconvergence diagnostic tables for nonconvergent nodes. Look up nonconvergent nodes in the circuit schematic. They are generally latches, Schmitt triggers, or oscillating nodes.
   - For stubborn convergence failures, bypass DC altogether with .TRAN with UIC set. Continue transient analysis until transients settle out, then specify .OP time to obtain an operating point during the transient analysis. An AC analysis also can be specified during the transient analysis by adding an .AC statement to the .OP time statement.
   - SCALE and SCALM scaling options have a significant effect on the element and model parameter values. Be careful with units.

Shorted Element Nodes

Star-Hspice disregards any capacitor, resistor, inductor, diode, BJT, or MOSFET that has all its leads connected together. The component is not counted in the component tally Star-Hspice produces. Star-Hspice issues the following warning:

```
** warning ** all nodes of element x:<name> are
connected together
```

Conductance Insertion Using DCSTEP

In a DC operating point analysis, failure to include conductances in a capacitor model results in broken circuit loops (since a DC analysis opens all capacitors), which might not be solvable. By including a small conductance in the capacitor model, the circuit loops are complete and can be solved.

Modeling capacitors as complete opens often results in the following error message:

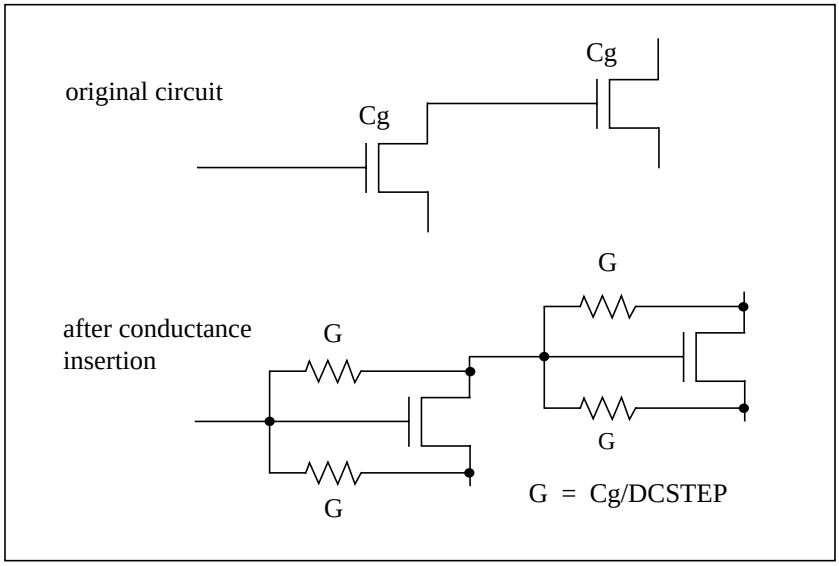
"No DC Path to Ground"

For a DC analysis, .OPTION DCSTEP is used to give a conductance value to all capacitors in the circuit. DCSTEP calculates the value as follows:

conductance = capacitance/DCSTEP

Figure 10-5 illustrates how Star-Hspice inserts conductance G in parallel with capacitance Cg to provide current paths around capacitances in DC analysis.

Figure 10-5: Conductance Insertion



Floating Point Overflow

Negative or zero MOS conductance sometimes results in Star-Hspice having difficulty converging. An indication of this type of problem is a floating point overflow during matrix solutions. Star-Hspice detects floating point overflow and invokes the Damped Pseudo Transient algorithm (CONVERGE = 1) to try to achieve DC convergence without requiring user intervention. If GMINDC is 1.0e-12 or less when a floating point overflow occurs, Star-Hspice sets it to 1.0e-11.

Diagnosing Convergence Problems

Before simulation, Star-Hspice diagnoses potential convergence problems in the input circuit, and provides an early warning to help debugging. When a circuit condition that indicates possible convergence problems is detected, Star-Hspice 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."
```

Nonconvergence Diagnostic Table

Two automatic printouts are generated when nonconvergence is encountered: the nodal voltage printout and the element printout (the diagnostic tables). The nodal voltage printout prints all nonconvergent node voltage names and the associated voltage error tolerances (tol). The element printout lists all nonconvergent elements, along with their associated element currents, element voltages, model parameters, and current error tolerances (tol).

To locate the branch current or nodal voltage resulting in nonconvergence, analyze the diagnostic tables for unusually large values of branch currents, nodal voltages or tolerances. Once located, initialize the node or branch using the .NODESET or .IC statements. The nonconvergence diagnostic table is automatically generated when a circuit simulation has not converged, indicating the quantity of recorded voltage failures and the quantity of recorded branch element failures. A voltage failure can be generated by any node in the circuit, including "hidden" nodes, such as the extra nodes created by parasitic resistors.

The element printout lists the subcircuit, model name, and element name of all parts of the circuit having nonconvergent nodal voltages or currents. Table 10-3 identifies the inverters, xinv21, xinv22, xinv23, and xinv24 as problem sub-circuits of a ring oscillator. It also indicates that the p-channel transistor of sub-circuits xinv21, xinv22, xinv24 are nonconvergent elements. The n-channel transistor of xinv23 is also a nonconvergent element.

The table gives the voltages and currents of the transistors, so the designer can quickly check to see if they are of a reasonable value. The tolds, tolbd, and tolbs error tolerances indicate how close the element currents (drain to source, bulk to drain, and bulk to source) were to a convergent solution. For tol variables, a value close to or below 1.0 indicates a convergent solution. As shown in Table 10-3, the tol values in the order of 100 indicate the currents were far from convergence. The element current and voltage values are also given (id, ibs, ibd, vgs, vds, and vbs). These values can be examined for realistic values and determination of the transistor regions of operation.

Table 10-3: Voltages, Currents, and Tolerances for Subcircuits

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 Nonconvergence Source

To locate a nonconvergence source, trace the circuit path for error tolerance. In an inverter chain, for example, the last inverter can have a very high error tolerance. If this is the case, the error tolerance of the elements driving the inverter should be examined. If the driving tolerance is high, the driving element could be the source of nonconvergence. However, if the tolerance is low, the driven element should be checked as the source of nonconvergence.

By examining the voltages and current levels of a nonconvergent MOSFET, you can 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 the current or nodal voltage values are extremely low, nonconvergence errors can be induced because a relatively large number is being divided by a very small number. This results in nonconvergence because the calculation produces a large result. A solution is to increase the value of the absolute accuracy options.

Use the diagnostic table in conjunction with the DC iteration limit (ITL1 statement) to find the sources of nonconvergence. By increasing or decreasing ITL1, output for the problem nodes and elements for a new iteration is printed—that is, the last iteration of the analysis set by ITL1.

Solutions for Nonconvergent Circuits

Nonconvergent circuits generally result from:

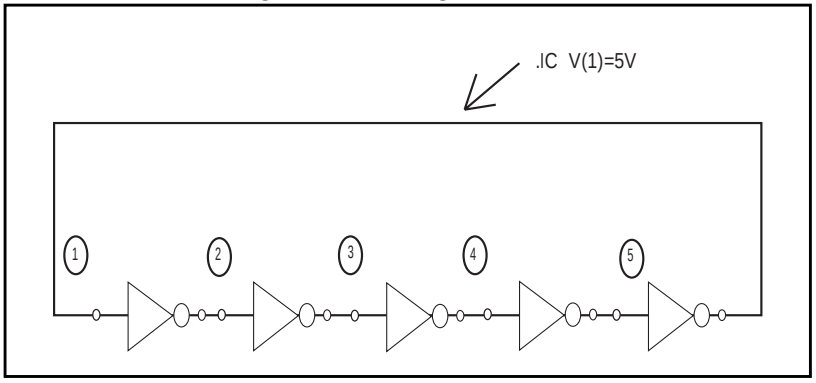
- Poor Initial Conditions
- Inappropriate Model Parameters
- PN Junctions (Diodes, MOSFETs, BJTs)

These conditions are discussed in the following sections.

Poor Initial Conditions

Multistable circuits need state information to guide the DC solution. You must initialize ring oscillators and flip-flops. These multistable circuits either give the intermediate forbidden state or cause a DC convergence problem. Initialize a circuit using the .IC statement to force a node to the requested voltage. Ring oscillators usually need only one stage set.

Figure 10-6: Ring Oscillator



It is best to set up the flip-flop with an .IC statement inside the subcircuit definition. In the following example, a local parameter “Qset” is set to 0. It is used as the value for the .IC statement to initialize the latch output node “Q”. This results in all latches having a default state of “Q” low. This state is overridden by the call to a latch by setting “Qset” to vdd.

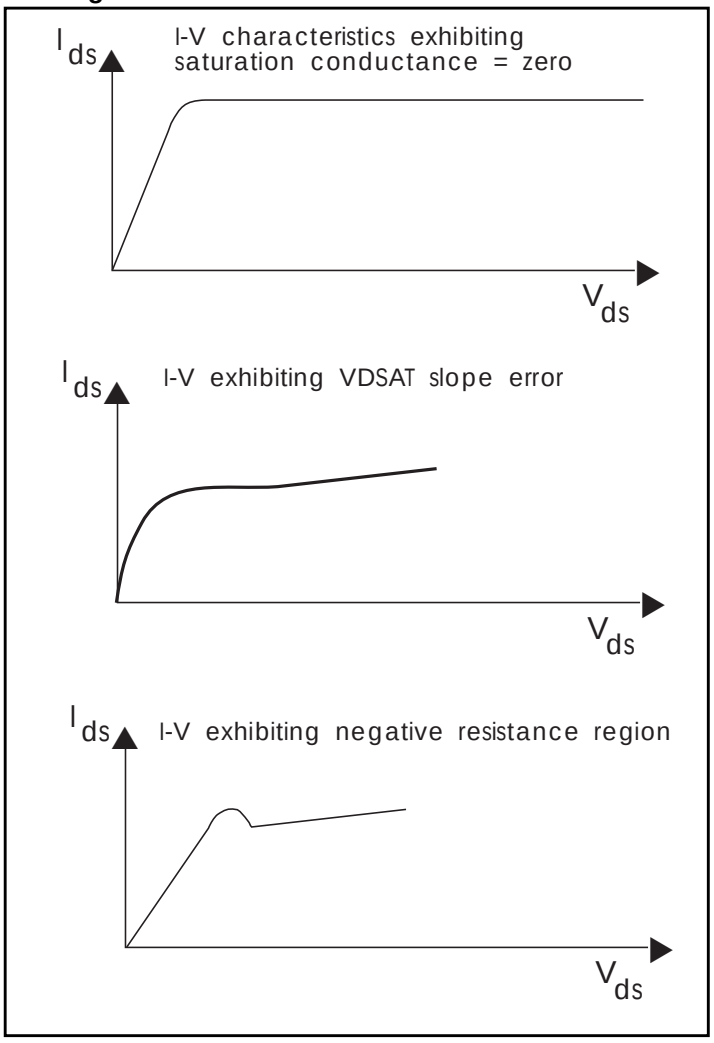
Example

```
.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

You can create a discontinuous IDS or capacitance model by imposing nonphysical model parameters. This can cause an “internal timestep too small” error during the transient simulation. The demonstration file *mosivcv.sp* 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 10-7: Discontinuous I-V Characteristics



If the simulation no longer converges when a component is added or a component value is changed, the model parameters are inappropriate or do not correspond to the physical values they represent. Check the Star-Hspice input netlist file for nonconvergent elements. Devices with a “TOL” greater than 1 are nonconvergent. Find the devices at the beginning of the combined logic string of gates that seem to start the nonconvergent string. Check the operating point of these devices very closely to see what region they operate in. The model parameters associated with this region are most likely inappropriate.

Circuit simulation is based on using single-transistor characterization to simulate a large collection of devices. If a circuit fails to converge, it can be caused by a single transistor somewhere in the circuit.

PN Junctions (Diodes, MOSFETs, BJTs)

PN junctions found in diode, BJT, and MOSFET models can exhibit nonconvergent behavior in both DC and transient analysis. For example, PN junctions often have a high off resistance, resulting in an ill-conditioned matrix. To overcome this, the options GMINDC and GMIN automatically parallel every PN junction in a design with a conductance. Nonconvergence can occur by overdriving the PN junction. This happens when a current-limiting resistor is omitted or has a very small value. In transient analysis, protection diodes often are temporarily forward biased (due to the inductive switching effect), overdriving the diode and resulting in nonconvergence if a current-limiting resistor is omitted.

Avant!

Chapter 11
Performing Transient Analysis

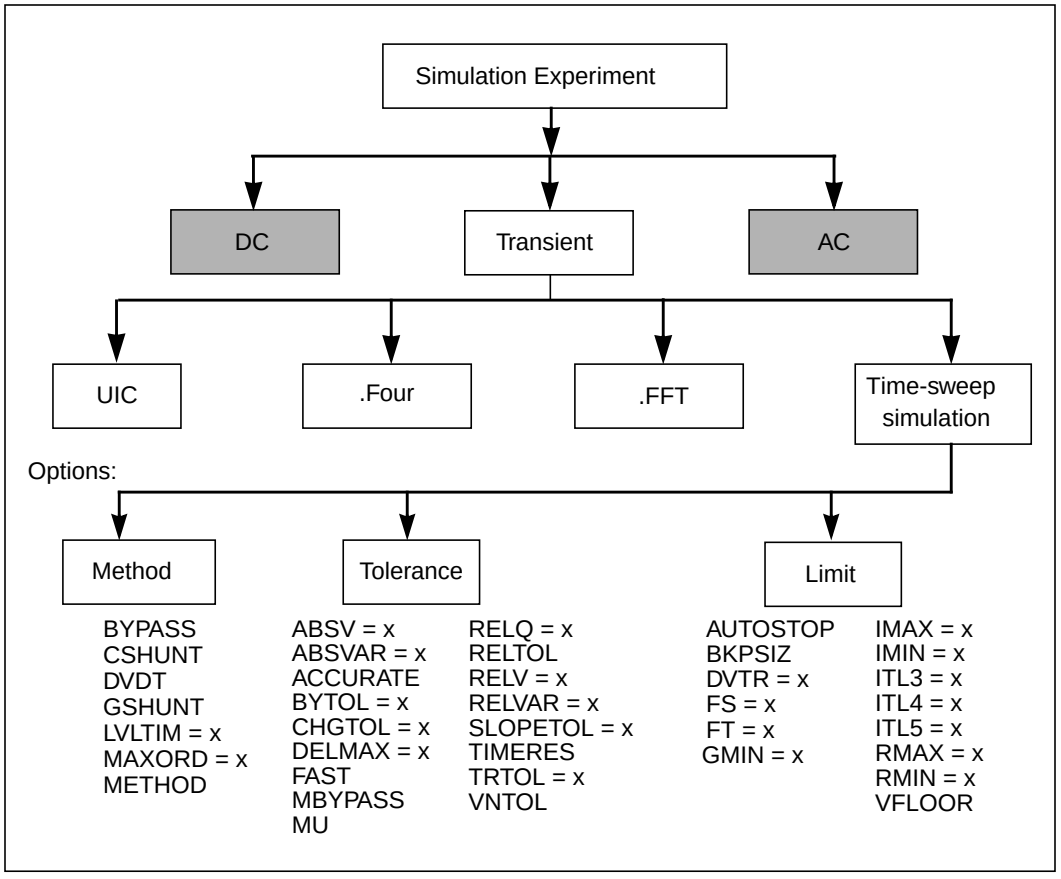
Star-Hspice transient analysis computes the circuit solution as a function of time over a time range specified in the .TRAN statement.

This chapter covers the following topics:

Understanding the Simulation Flow

Figure 11-1 illustrates the transient analysis simulation flow for Star-Hspice.

Figure 11-1: Transient Analysis Simulation Flow



Understanding Transient Analysis

Since transient analysis is dependent on time, it uses different analysis algorithms, control options with different convergence-related issues and different initialization parameters than DC analysis. However, since a transient analysis first performs a DC operating point analysis (unless the UIC option is specified in the .TRAN statement), most of the DC analysis algorithms, control options, and initialization and convergence issues apply to transient analysis.

Initial Conditions for Transient Analysis

Some circuits, such as oscillators or circuits with feedback, do not have stable operating point solutions. For these circuits, either the feedback loop must be broken so that a DC operating point can be calculated or the initial conditions must be provided in the simulation input. The DC operating point analysis is bypassed if the UIC parameter is included in the .TRAN statement. If UIC is included in the .TRAN statement, a transient analysis is started using node voltages specified in an .IC statement. If a node is set to 5 V in a .IC statement, the value at that node for the first time point (time 0) is 5 V.

You can use the .OP statement to store an estimate of the DC operating point during a transient analysis.

Example

```
.TRAN 1ns 100ns UIC
.OP 20ns
```

The .TRAN statement UIC parameter in the above example bypasses the initial DC operating point analysis. The .OP statement calculates transient operating point at t = 20 ns during the transient analysis.

Although a transient analysis might provide a convergent DC solution, the transient analysis itself can still fail to converge. In a transient analysis, the error message “internal timestep too small” indicates that the circuit failed to converge. The convergence failure might be due to stated initial conditions that are not close enough to the actual DC operating point values. See the later part of this chapter for a discussion of transient analysis convergence aids.

Using the .TRAN Statement

Syntax

Single-point analysis:

```
.TRAN var1 START = start1 STOP = stop1 STEP = incr1
```

or

```
.TRAN var1 START = <param_expr1> STOP = <param_expr2>
+ STEP = <param_expr3>
```

Double-point analysis:

```
.TRAN var1 START = start1 STOP = stop1 STEP = incr1
+ <SWEEP var2 type np start2 stop2>
```

or

```
.TRAN tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>
+ <START = val> <UIC> +    <SWEEP var pstart
+ pstop pincr>
```

Parameterized sweep:

```
.TRAN tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>
+ <START = val> <UIC>
```

Data driven sweep:

```
.TRAN DATA = datanm
```

or

```
.TRAN var1 START = start1 STOP = stop1 STEP = incr1
+ <SWEEP DATA = datanm>
```

or

```
.TRAN DATA = datanm<SWEEP var pstart pstop pincr>
```

Monte Carlo:

```
.TRAN tincr1 tstop1 <tincr2 tstop2 ...tincrN tstopN>
+ <START = val> <UIC><SWEEP MONTE = val>
```

Optimization:

```
.TRAN DATA = datanm OPTIMIZE = opt_par_fun
+ RESULTS = measnames MODEL = optmod
```

Transient sweep specifications can include the following keywords and parameters:

DATA = datanm	Data name referred to in the .TRAN statement
MONTE = val	Produces a number *val* of randomly generated values that are used to select parameters from a distribution. The distribution can be *Gaussian*, *Uniform*, or *Random Limit.*
np	Number of points or number of points per decade or octave, depending on the preceding keyword
param_expr...	User-specified expressions—for example, param_expr1...param_exprN
pincr	Voltage, current, element or model parameter, or temperature increment value **Note:** If you set the “type” variation, use “np” (number of points) instead of “pincr”.

pstart	Starting voltage, current, temperature, any element or model parameter value **Note:** If you set the type variation to POI (list of points), specify a list of parameter values, instead of "pstart pstop".
pstop	Final voltage, current, temperature, any element or model parameter value
START	Time at which printing or plotting is to begin. The START keyword is optional: you can specify start time without preceding it with "START = " **Note:** If you use the .TRAN statement with a .MEASURE statement, a nonzero START time can result in incorrect .MEASURE results. Do not use nonzero START times in .TRAN statements, when you also use .MEASURE.
SWEEP	Keyword to indicate a second sweep is specified on the .TRAN statement
tincr1…	Printing or plotting increment for printer output, and the suggested computing increment for the postprocessor.
tstop1…	Time at which the transient analysis stops incrementing by tincr1. If another tincr-tstop pair follows, the analysis continues with the new increment.

type	Specifies any of the following keywords: ■ DEC – decade variation ■ OCT – octave variation (the value of the designated variable is eight times its previous value) ■ LIN – linear variation ■ POI – list of points
UIC	Causes Star-Hspice to use the nodal voltages specified in the .IC statement (or by the "IC = " parameters in the various element statements) to calculate the initial transient conditions, rather than solving for the quiescent operating point
var	Name of an independent voltage or current source, any element or model parameter, or the keyword TEMP (indicating a temperature sweep). Star-Hspice supports source value sweep, referring to the source name (SPICE style). However, if a parameter sweep, a .DATA statement, and a temperature sweep are specified, a parameter name must be chosen for the source value and subsequently referred to in the .TRAN statement. The parameter name must not start with V or I.

Example

1. The following example performs and prints the transient analysis every 1 ns for 100 ns.

   ```
   .TRAN 1NS 100NS
   ```

2. The following example performs the calculation every 0.1 ns for the first 25 ns, and then every 1 ns until 40 ns; the printing and plotting begin at 10 ns.

   ```
   .TRAN .1NS 25NS 1NS 40NS START = 10NS
   ```

3. The following example performs the calculation every 10 ns for 1 µs; the initial DC operating point calculation is bypassed, and the nodal voltages specified in the .IC statement (or by IC parameters in element statements) are used to calculate initial conditions.

   ```
   .TRAN 10NS 1US UIC
   ```

4. The following example increases the temperature by 10 °C through the range -55 °C to 75 °C and performs transient analysis for each temperature.

   ```
   .TRAN 10NS 1US UIC SWEEP TEMP -55 75 10
   ```

5. The following performs an analysis for each load parameter value at 1 pF, 5 pF, and 10 pF.

   ```
   .TRAN 10NS 1US SWEEP load POI 3 1pf 5pf 10pf
   ```

6. The following example is a data driven time sweep and allows a data file to be used as sweep input. If the parameters in the data statement are controlling sources, they must be referenced by a piecewise linear specification.

   ```
   .TRAN data = dataname
   ```

Using the .BIASCHK Statement

Breakdown can occur if the voltage bias between some terminals of an element is too large. The .BIASCHK statement monitors the voltage bias, using the limits and noise that you define. Bias monitoring can check the bias that you want to monitor during transient analysis, and report the following:

- element name
- time
- terminals
- bias exceeding the limit
- number of times the bias exceeds the limit for an appointed element

The information is saved as a warning and a BIASCHK summary, in the **.lis* file.

This command is for MOS and capacitors only in Star-Hspice release 2001.4. For example, the .BIASCHK statement checks for voltages that exceed a user-specified limit for MOS dielectric breakdown. BIASCK can check voltages from the gate to either the source, drain, or bulk.

BIASCHK cannot detect the bias that exceeds the limit, if the bias is always the same value during transient analysis.

If a model name, referenced in an active element statement, contains a period (.), then .BIASCHK reports an error. This occurs because it is unclear whether a reference such as `x.123` is a model name or a sub-circuit name (123 model in the x sub-circuit).

Instance (element) and model names *can* contain wildcards, either ? (stands for one character) or * (stands for 0 or more characters).

Syntax

```
.biaschk type terminal1=t1 terminal2=t2 limit=lim
+ <noise=ns><name=devname1><name=devname2>...
+ <mname=modelname1><mname=modelname2> ...
```

where:

type	Element type that you want to check MOS (R, C ...) Type is NMOS, PMOS, or C in 2001.4
terminal 1, 2	Terminals that you want to check between: ■ For MOS level 57: *nd, ng, ns, ne, np, n6* ■ For MOS level 58: *nd, ngf, ns, ngb* ■ For MOS level 59: *nd, ng, ns, ne, np* ■ For other MOS level: *nd, ng, ns, nb* For Capacitor: *n1, n2*
limit	Biaschk limit that you define. Reports an error if the bias voltage, between appointed terminals of appointed elements and models, are larger than the limit.
noise	Biaschk noise that you define Default = 0.1v Noise filer off some of the results (the local maximum bias voltage that is larger than the limit). The local max is replaced by the next local max, if the following conditions are satisfied: 1. local_max-local_min<noise 2. next local_max-local_min<noise 3. this local max is smaller than the next local max.
name	Element name that you want to check
mname	Model name. Elements of this model are checked for bias

If you do not set `name` and `mname` in the statement, all elements of this type (type is a required keyword in the .biaschk card) are checked for bias voltage.

You can use a wild card to describe name and mname in biaschk card.

- ? stands for one character.
- * stands for 0 or more characters.

Example

```
.biaschk NMOS terminal1=ng terminal2=nb limit=2v
+ noise=0.01v name=x1.x3.m1 mname=nch.1 name=m3
```

Options for the .biaschk command

biasfile

- If you use this option, the results of all .biaschk commands in this netlist are output to a file that you specify.
- If you do not set this option, the results are output to the `*.lis` file.

Example

```
.option biasfile='biaschk/mos.bias'
```

biawarn

- If you set this option to 1, a warning message is sent out immediately, when any local max bias voltage exceeds the limit *during* transient analysis. The results summary, which was filtered by noise, is sent out *after* this transient analysis.
- If you set this option to 0 (the default), no warning message is sent out *during* transient analysis. The results is sent out *after* this transient analysis.

Example

```
.option biawarn=1
```

Understanding the Control Options

The options in this section modify the behavior of the transient analysis integration routines. Delta refers to the internal timestep. TSTEP and TSTOP refer to the step and stop values entered with the .TRAN statement. The options are grouped into three categories: method, tolerance, and limit:

Method	Tolerance		Limit	
BYPASS	ABSH	RELH	AUTOSTO	ITL3
CSHUNT	ABSV	RELI	P	ITL4
DVDT	ABSVAR	RELQ	BKPSIZ	ITL5
GSHUNT	ACCURAT	RELTOL	DELMAX	RMAX
INTERP	E BYTOL	RELV	DVTR	RMIN
ITRPRT	CHGTOL	RELVAR	FS	VFLOOR
LVLTIM	DI	SLOPETOL	FT	
MAXORD	FAST	TIMERES	GMIN	
METHOD	MBYPASS	TRTOL		
PURETP	MAXAMP	VNTOL		
	MU	XMU		

Method Options

BYPASS	Speeds up simulation by not updating the status of latent devices. Setting .OPTION BYPASS = 1 enables bypassing. BYPASS applies to MOSFETs, MESFETs, JFETs, BJTs, and diodes. Default = 1. **Note:** For some circuit types, BYPASS can result in non-convergence, and loss of accuracy in transient analysis and operating point calculations.

CSHUNT	Capacitance added from each node to ground. Adding a small CSHUNT to each node can solve some “internal timestep too small” problems caused by high-frequency oscillations or numerical noise. Default = 0.
DVDT	Allows the timestep to be adjusted based on node voltage rates of change. Choices are: ■ 0 - original algorithm ■ 1 - fast ■ 2 - accurate ■ 3,4 - balance speed and accuracy Default = 4. The ACCURATE option also increases the accuracy of the results.
GSHUNT	Conductance added from each node to ground. The default value is zero. Adding a small GSHUNT to each node can solve some “internal timestep too small” problems caused by high frequency oscillations or by numerical noise.
INTERP	Limits output to post-analysis tools, such as Cadence or Zuken, to only the .TRAN timestep intervals. By default, Star-Hspice outputs all convergent iterations in *design.tr#* file. INTERP typically produces a much smaller *design.tr#* file. Use INTERP = 1 with caution, when you also use a .MEASURE statement. To compute measure statements, Star-Hspice uses the postprocessing output. Reducing postprocessing output can lead to interpolation errors in measure results.

	Note: When you rung a data-driven transient analysis (.TRAN DATA statement) within optimization routines, INTERP is forced to 1. As a result, all measurement results are made at the time points of the data in the data-driven sweep. If the measurement needs to use all converged internal timesteps, such as AVG or RMS calculations, set ITRPRT = 1.
ITRPRT	Prints output variables at their internal timepoint values. Using this option can generate a long output list.
LVLTIM = x	Selects the timestep algorithm used for transient analysis. If LVLTIM = 1, the DVDT timestep algorithm is used. If LVLTIM = 2, the local truncation error timestep algorithm is used. If LVLTIM = 3, the DVDT timestep algorithm with timestep reversal is used. If the GEAR method of numerical integration and linearization is used, LVLTIM = 2 is selected. If the TRAP linearization algorithm is used, LVLTIM 1 or 3 can be selected. Using LVLTIM = 1 (the DVDT option) helps avoid the “internal timestep too small” nonconvergence problem. The local truncation algorithm (LVLTIM = 2), however, provides a higher degree of accuracy and prevents errors propagating from time point to time point, which can sometimes result in an unstable solution. Default = 1.

MAXORD = x	Sets the maximum order of integration when the GEAR method is used (see METHOD). The value of x can be either 1 or 2. If MAXORD = 1, the backward Euler method of integration is used. MAXORD = 2, however, is more stable, accurate, and practical. Default = 2.0.
METHOD = name	Sets the numerical integration method used for a transient analysis to either GEAR or TRAP. To use GEAR, set METHOD = GEAR. This automatically sets LVLTIM = 2. (You can change LVLTIM from 2 to 1 or 3 by setting LVLTIM = 1 or 3 after the METHOD = GEAR option. This overrides the LVLTIM = 2 setting made by METHOD = GEAR.) TRAP (trapezoidal) integration generally results in reduced program execution time, with more accurate results. However, trapezoidal integration can introduce an apparent oscillation on printed or plotted nodes that might not be caused by circuit behavior. To test if this is the case, run a transient analysis with a small timestep. If the oscillation disappears, it was due to the trapezoidal method. The GEAR method acts as a filter, removing the oscillations found in the trapezoidal method. Highly nonlinear circuits such as operational amplifiers can require very long execution times with the GEAR method. Circuits that are not convergent with trapezoidal integration often converge with GEAR. Default = TRAP (trapezoidal).

PURETP	Sets the integration method to use for the reversal time point. The default value is 0. If you set `puretp=1`, when Star-Hspice encounters non-convergence, it uses TRAP (instead of B.E) for the reversed time point. You can use this option to help some oscillating circuits to oscillate, if the default simulation process cannot satisfy the result. Use this option with the `method=TRAP` statement.

Tolerance Options

ABSH = x	Sets the absolute current change through voltage defined branches (voltage sources and inductors). In conjunction with DI and RELH, ABSH is used to check for current convergence. Default = 0.0.
ABSV = x	Sets the absolute minimum voltage for DC and transient analysis. Decrease VNTOL if accuracy is of more concern than convergence. If voltages less than 50 microvolts are required, VNTOL can be reduced to two orders of magnitude less than the smallest desired voltage, ensuring at least two digits of significance. Typically, VNTOL need not be changed unless the circuit is a high voltage circuit. For 1000 volt circuits, a reasonable value can be 5 to 50 millivolts. ABSV is the same as VNTOL. Default = 50 (microvolts).

ABSVAR = x	Sets the limit on the maximum voltage change from one time point to the next. Used with the DVDT algorithm. If the simulator produces a convergent solution that is greater than ABSVAR, the solution is discarded, the timestep is set to a smaller value, and the solution is recalculated. This is called a timestep reversal. Default = 0.5 (volts).
ACCURATE	Selects a time algorithm that uses LVLTIM = 3 and DVDT = 2 for circuits such as high-gain comparators. Circuits that combine high gain with large dynamic range should use this option to guarantee solution accuracy. When ACCURATE is set to 1, it sets the following control options: ■ LVLTIM = 3 ■ DVDT = 2 ■ RELVAR = 0.2 ■ ABSVAR = 0.2 ■ FT = 0.2 ■ RELMOS = 0.01 Default = 0.
BYTOL = x	Specifies the tolerance for the voltage at which a MOSFET, MESFET, JFET, BJT, or diode is considered latent. Star-Hspice does not update the status of latent devices. Default = MBYPASS x VNTOL.
CHGTOL = x	Sets the charge error tolerance when LVLTIM = 2 is set. CHGTOL, along with RELQ, sets the absolute and relative charge tolerance for all Star-Hspice capacitances. Default = 1e-15 (coulomb).

DI = x	Sets the maximum iteration-to-iteration current change through voltage defined branches (voltage sources and inductors). This option is only applicable when the value of the DI control option is greater than 0. Default = 0.0.
FAST	Speeds up simulation by not updating the status of latent devices. This option is applicable for MOSFETs, MESFETs, JFETs, BJTs, and diodes. Default = 0. A device is considered to be latent when its node voltage variation from one iteration to the next is less than the value of either the BYTOL control option or the BYPASSTOL element parameter. (When FAST is on, Star-Hspice sets BYTOL to different values for different types of device models.) In addition to the FAST option, the input preprocessing time can be reduced by the options NOTOP and NOELCK. Increasing the value of the MBYPASS option or the BYTOL option setting also helps simulations run faster, but can reduce accuracy.
MAXAMP = x	Sets the maximum current through voltage defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, an error is issued. Default = 0.0.

MBYPASS = x	Used to compute the default value for the BYTOL control option: `BYTOL = MBYPASS x VNTOL` Also multiplies voltage tolerance RELV. MBYPASS should be set to about 0.1 for precision analog circuits. Default = 1 for DVDT = 0, 1, 2, or 3. Default = 2 for DVDT = 4.
MU = x, *XMU = x*	The coefficient for trapezoidal integration. The range for MU is 0.0 to 0.5. XMU is the same as MU. Default = 0.5.
RELH = x	Sets relative current tolerance through voltage defined branches (voltage sources and inductors). RELH is used to check current convergence. This option is applicable only if the value of the ABSH control option is greater than zero. Default = 0.05.
RELI = x	Sets the relative error/tolerance change from iteration to iteration to determine convergence for all currents in diode, BJT, and JFET devices. (RELMOS sets the tolerance for MOSFETs). This is the percent change in current from the value calculated at the previous timepoint. Default = 0.01 for KCLTEST = 0, 1e-4 for KCLTEST = 1.
RELQ = x	Used in the local truncation error timestep algorithm (LVLTIM = 2). RELQ changes the size of the timestep. If the capacitor charge calculation of the present iteration exceeds that of the past iteration by a percentage greater than the value of RELQ, the internal timestep (Delta) is reduced. Default = 0.01.

RELTOL, RELV	Sets the relative error tolerance for voltages. RELV is used in conjunction with the ABSV control option to determine voltage convergence. Increasing RELV increases the relative error. RELV is the same as RELTOL. Options RELI and RELVDC default to the RELTOL value. Default = 1e-3.
RELVAR = x	Used with ABSVAR and the timestep algorithm option DVDT. RELVAR sets the relative voltage change for LVLTIM = 1 or 3. If the nodal voltage at the current time point exceeds the nodal voltage at the previous time point by RELVAR, the timestep is reduced and a new solution at a new time point is calculated. Default = 0.30 (30%).
SLOPETOL = x	Sets a lower limit for breakpoint table entries in a piecewise linear (PWL) analysis. If the difference in the slopes of two consecutive PWL segment is less than the SLOPETOL value, the breakpoint table entry for the point between the segments is ignored. Default = 0.5
TIMERES = x	Sets a minimum separation between breakpoint values for the breakpoint table. If two breakpoints are closer together in time than the TIMERES value, only one of them is entered in the breakpoint table. Default = 1 ps.

TRTOL = x	Used in the local truncation error timestep algorithm (LVLTIM = 2). TRTOL is a multiplier of the internal timestep generated by the local truncation error timestep algorithm. TRTOL reduces simulation time, while maintaining accuracy. It is a factor that estimates the amount of error introduced by truncating the Taylor series expansion used in the algorithm. This error is a reflection of what the minimum value of the timestep should be to reduce simulation time and maintain accuracy. The range of TRTOL is 0.01 to 100, with typical values being in the 1 to 10 range. If TRTOL is set to 1, the minimum value, a very small timestep is used. As the setting of TRTOL increases, the timestep size increases. Default = 7.0.
VNTOL = x, ABSV	Sets the absolute minimum voltage for DC and transient analysis. Decrease VNTOL if accuracy is of more concern than convergence. If voltages less than 50 microvolts are required, VNTOL can be reduced to two orders of magnitude less than the smallest desired voltage, ensuring at least two digits of significance. Typically, VNTOL need not be changed unless the circuit is a high voltage circuit. For 1000 volt circuits, a reasonable value can be 5 to 50 millivolts. ABSV is the same as VNTOL. Default = 50 (microvolts).
XMU = x	The coefficient for trapezoidal integration. The range for MU is 0.0 to 0.5. XMU is the same as MU. Default = 0.5.

Limit Options

AUTOSTOP	Stops the transient analysis when all TRIG-TARG and FIND-WHEN measure functions are calculated. This option can result in a substantial CPU time reduction. If the data file contains measure functions such as AVG, RMS, MIN, MAX, PP, ERR, ERR1,2,3, and PARAM, then AUTOSTOP is disabled.
BKPSIZ = x	Sets the size of the breakpoint table. Default = 5000.
DELMAX = x	Sets the maximum value for the internal timestep Delta.Star-Hspice automatically sets the DELMAX value based on various factors, which are listed in "Timestep Control for Accuracy" on page Chapter 1127. This means that the initial DELMAX value shown in the Star-Hspice output listing is generally not the value used for simulation.
DVTR	Allows the use of voltage limiting in transient analysis. Default = 1000.
FS = x	Sets the fraction of a timestep (TSTEP) that Delta (the internal timestep) is decreased for the first time point of a transient. Decreasing the FS value helps circuits that have timestep convergence difficulties. It also is used in the DVDT = 3 method to control the timestep. $Delta = FS \times [MIN(TSTEP, DELMAX, BKPT)]$ where DELMAX is specified and BKPT is related to the breakpoint of the source. TSTEP is set in the .TRAN statement. Default = 0.25.

FT = x	Sets the fraction of a timestep (TSTEP) by which Delta (the internal timestep) is decreased for an iteration set that does not converge. It is also used in DVDT = 2 and DVDT = 4 to control the timestep. Default = 0.25.
GMIN = x	Sets the minimum conductance allowed for in a transient analysis time sweep. Default = 1e-12.
IMAX = x, *ITL4 = x*	Determines the maximum timestep in the timestep algorithms used for transient analysis simulations. IMAX sets an upper limit on the number of iterations allowed to obtain a convergent solution at a timepoint. If the number of iterations needed is greater than IMAX, the internal timestep Delta is decreased by a factor equal to the transient control option FT, and a new solution is calculated using the new timestep. IMAX also works in conjunction with the transient control option IMIN. ITL4 is the same as IMAX. Default = 8.0.
IMIN = x, *ITL3 = x*	Determines the timestep in the algorithms used for transient analysis siimulations. IMIN sets a lower limit on the number of iterations required to obtain convergence. If the number of iterations is less than IMIN, the internal timestep, Delta, is doubled. This option is useful for decreasing simulation times in circuits where the nodes are stable most of the time, such as digital circuits. If the number of iterations is greater than IMIN, the timestep is kept the same unless the option IMAX is exceeded (see IMAX). ITL3 is the same as IMIN. Default = 3.0.

ITL3 = x	Determines the timestep in the algorithms used for transient analysis simulations. IMIN sets a lower limit on the number of iterations required to obtain convergence. If the number of iterations is less than IMIN, the internal timestep, Delta, is doubled. This option is useful for decreasing simulation times in circuits where the nodes are stable most of the time, such as digital circuits. If the number of iterations is greater than IMIN, the timestep is kept the same unless the option IMAX is exceeded (see IMAX). ITL3 is the same as IMIN. Default = 3.0.
ITL4 = x	Determines the maximum timestep in the timestep algorithms used for transient analysis simulations. IMAX sets an upper limit on the number of iterations allowed to obtain a convergent solution at a timepoint. If the number of iterations needed is greater than IMAX, the internal timestep Delta is decreased by a factor equal to the transient control option FT, and a new solution is calculated using the new timestep. IMAX also works in conjunction with the transient control option IMIN. ITL4 is the same as IMAX. Default = 8.0.
ITL5 = x	Sets the transient analysis total iteration limit. If a circuit uses more than ITL5 iterations, the program prints all results to that point. The default allows an infinite number of iterations. Default = 0.0.
RMAX = x	Sets the TSTEP multiplier, which determines the maximum value, DELMAX, that can be used for the internal timestep Delta: DELMAX = TSTEPxRMAX Default = 5 when dvdt = 4 and lvltim = 1, otherwise, default = 2.

RMIN = x	Sets the minimum value of Delta (internal timestep). An internal timestep smaller than RMINxTSTEP results in termination of the transient analysis with the error message “internal timestep too small”. Delta is decreased by the amount set by the FT option if the circuit has not converged in IMAX iterations. Default = 1.0e-9.
VFLOOR = x	Sets a lower limit for the voltages that are printed in the output listing. All voltages lower than VFLOOR are printed as 0. This only affects the output listing: the minimum voltage used in a simulation is set by VNTOL (ABSV).

Matrix Manipulation Options

After linearization of the individual elements within a Star-Hspice input netlist file, the linear equations are constructed for the matrix. User-controlled variables affecting the construction and solution of the matrix equation include options PIVOT and GMIN. GMIN places a variable into the matrix that prevents the matrix becoming ill-conditioned.

Pivot Option

Select the PIVOT option for a number of different pivoting methods to reduce simulation time and assist in both DC and transient convergence. Pivoting reduces the error resulting from elements in the matrix that are widely different in magnitude. The use of PIVOT results in a search of the matrix for the largest element value. This element value then is used as the pivot.

Controlling Simulation Speed and Accuracy

Convergence is defined as the ability to obtain a solution to a set of circuit equations within a given tolerance criteria and number of iterations. In numerical circuit simulation, the designer specifies a relative and absolute accuracy for the circuit solution and the simulator iteration algorithm attempts to converge to a solution that is within these set tolerances. In many cases, the speed of reaching a solution also is of primary interest to the designer, particularly for preliminary design trials, and some accuracy is willingly sacrificed.

Simulation Speed

Star-Hspice can substantially reduce the computer time needed to solve complex problems. The following user options alter internal algorithms to increase simulation efficiency.

- .OPTIONS FAST – sets additional options that increase simulation speed with little loss of accuracy
- .OPTIONS AUTOSTOP – terminates the simulation when all .MEASURE statements have completed. This is of special interest when testing corners.

The FAST and AUTOSTOP options are described in Understanding the Control Options on page 11-12.

Simulation Accuracy

Star-Hspice is shipped with control option default values that aim for superior accuracy while delivering good performance in simulation time. The control options and their default settings to maximize accuracy are:

```
DVDT = 4   LVLTIM = 1    RMAX = 5   SLOPETOL = 0.75
FT = FS = 0.25           BYPASS = 1
BYTOL = MBYPASSxVNTOL = 0.100m
```

Note: BYPASS is turned on (set to 1) only when DVDT = 4. For other DVDT settings, BYPASS is off (0). SLOPETOL is set to 0.75 when 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. Relationships between DVDT and other control options are discussed in “Selecting Timestep Control Algorithms” on page Chapter 1134.

The DELMAX control option also affects simulation accuracy. DELMAX specifies the maximum allowed timestep size. If DELMAX is not set in an .OPTIONS statement, Star-Hspice computes a DELMAX value. Factors that determine the computed DELMAX value are:

- .OPTIONS RMAX and FS
- Breakpoint locations for a PWL source
- Breakpoint locations for a PULSE source
- Smallest period for a SIN source
- Smallest delay for a transmission line component
- Smallest ideal delay for a transmission line component
- TSTEP value in a .TRAN analysis
- Number of points in an FFT analysis

The FS and RMAX control options provide some user control over the DELMAX value. The FS option, which defaults to 0.25, scales the breakpoint interval in the DELMAX calculation. The RMAX option, which defaults to 5 if DVDT = 4 and LVLTIM = 1, scales the TSTEP (timestep) size in the DELMAX calculation.

For circuits that contain oscillators or ideal delay elements, an .OPTIONS statement should be used to set DELMAX to one-hundredth of the period or less.

The ACCURATE control option tightens the simulation options to give the most accurate set of simulation algorithms and tolerances. When ACCURATE is set to 1, it sets the following control options:

DVDT = 2	LVLTIM = 3	FT = FS = 0.2	SLOPETOL = 0.5
BYTOL = 0	BYPASS = 0	RMAX = 2	
RELVAR = 0.2	ABSVAR = 0.2	RELMOS = 0.01	

Models and Accuracy

Simulation accuracy relies heavily on the sophistication and accuracy of the models used. More advanced MOS, BJT, and GaAs models give superior results for critical applications. Simulation accuracy is increased by:

- Algebraic models that describe parasitic interconnect capacitances as a function of the width of the transistor. The wire model extension of the resistor can model the metal, diffusion, or poly interconnects to preserve the relationship between the physical layout and electrical property.
- MOS model parameter ACM that calculates defaults for source and drain junction parasitics. Star-Hspice uses ACM equations to calculate the size of the bottom wall, the length of the sidewall diodes, and the length of a lightly doped structure. SPICE defaults with no calculation of the junction diode. Specify AD, AS, PD, PS, NRD, NRS to override the default calculations.
- MOS model parameter CAPOP = 4 that models the most advanced charge conservation, non-reciprocal gate capacitances. The gate capacitors and overlaps are calculated from the IDS model for LEVEL 49 or 53, however, the CAPOP parameter is ignored, model parameter CAPMOD with reasonable value should be used instead.

Guidelines for Choosing Accuracy Options

Use the ACCURATE option for

- Analog or mixed signal circuits
- Circuits with long time constants, such as RC networks
- Circuits with ground bounce

Use the default options (DVDT = 4) for

- Digital CMOS
- CMOS cell characterization
- Circuits with fast moving edges (short rise and fall times)

For ideal delay elements, use one of the following:

- ACCURATE
- DVDT = 3
- DVDT = 4, and, if the minimum pulse width of any signal is less than the minimum ideal delay, set DELMAX to a value smaller than the minimum pulse width.

Numerical Integration Algorithm Controls

When using Star-Hspice for transient analysis, you can select one of three options, Gear, Backward-Euler or Trapezoidal, to convert differential terms into algebraic terms.

Syntax

Gear algorithm:

```
.OPTION METHOD = GEAR
```

Backward-Euler:

```
.OPTION METHOD = GEAR MU = 0
```

Trapezoidal algorithm (default):

```
.OPTION METHOD = TRAP
```

Each of these algorithms has advantages and disadvantages, but the trapezoidal is the preferred algorithm overall because of its highest accuracy level and lowest simulation time.

The selection of the algorithm is not, however, an elementary task. The appropriate algorithm for convergence depends to a large degree on the type of circuit and its associated behavior for different input stimuli.

Gear and Trapezoidal Algorithms

The timestep control algorithm is automatically set by the choice of algorithm. In Star-Hspice, if the GEAR algorithm is selected (including Backward-Euler), the timestep control algorithm defaults to the truncation timestep algorithm. On the other hand, if the trapezoidal algorithm is selected, the DVDT algorithm is the default. You can change these Star-Hspice default by using the timestep control options.

Figure 11-2: Time Domain Algorithm

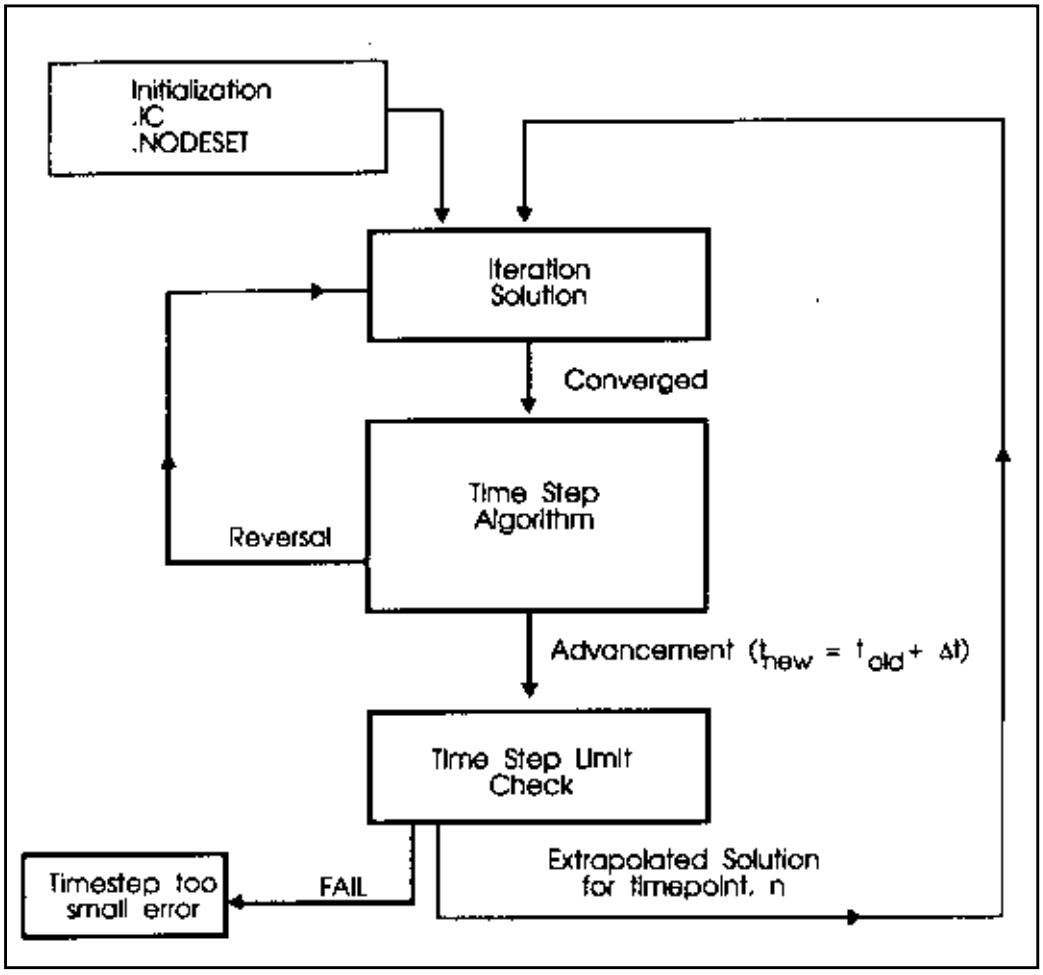
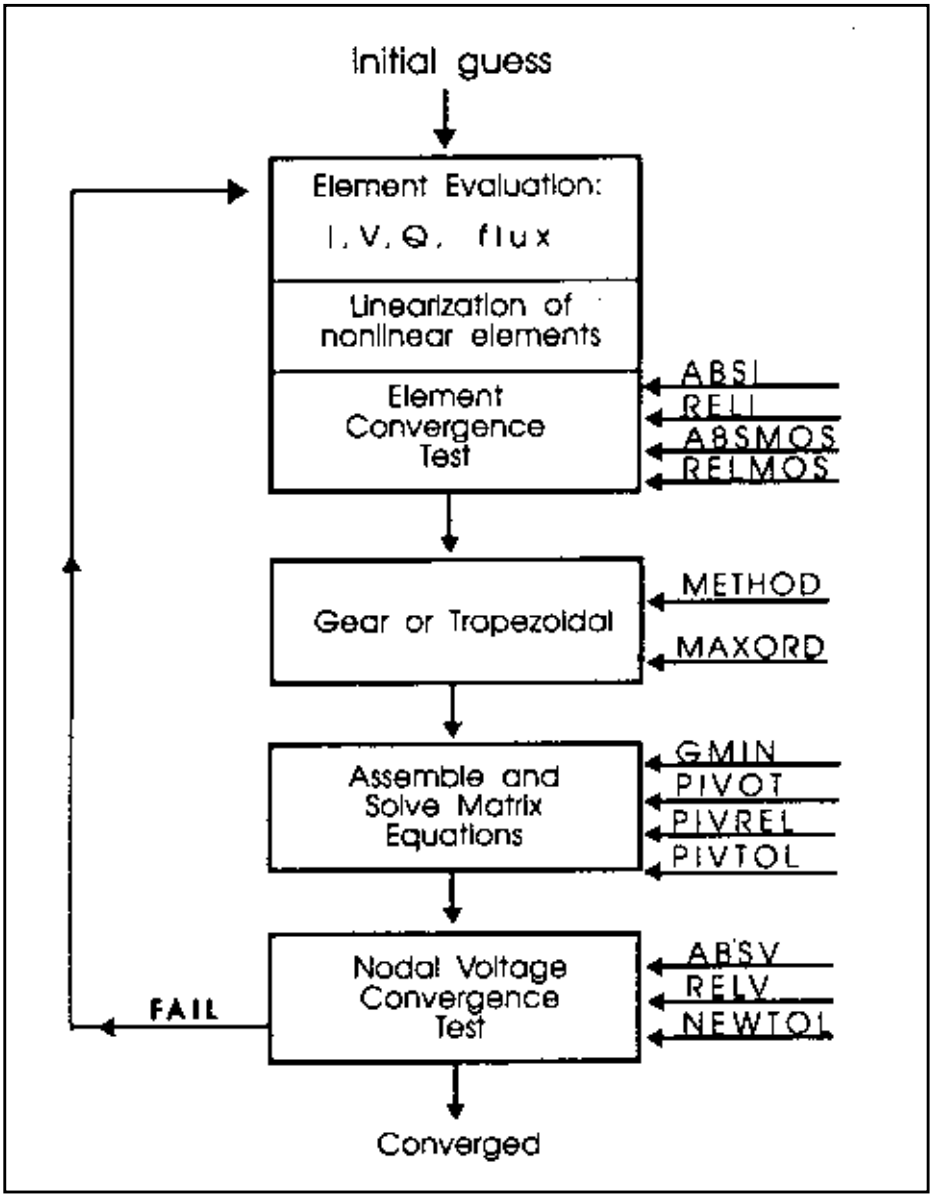


Figure 11-3: Iteration Algorithm



One limitation of the trapezoidal algorithm is that it can result in computational oscillation—that is, an oscillation caused by the trapezoidal algorithm and not by the circuit design. This also produces an unusually long simulation time. When this occurs in circuits that are inductive in nature, such as switching regulators, use the GEAR algorithm.

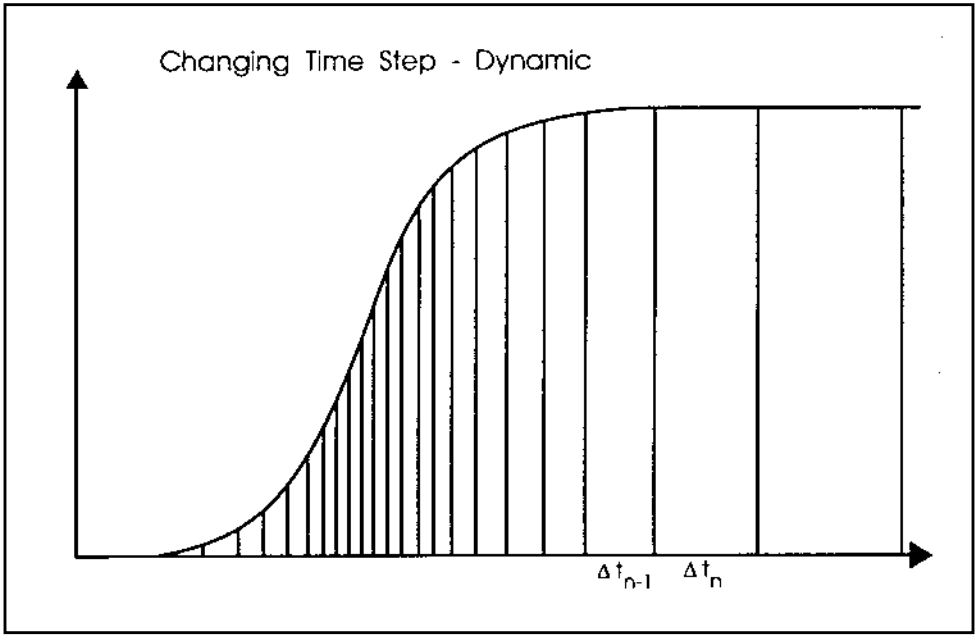
Selecting Timestep Control Algorithms

Star-Hspice allows the selection of three dynamic timestep control algorithms:

- Iteration Count Dynamic Timestep Algorithm
- Local Truncation Error (LTE) Dynamic Timestep Algorithm
- DVDT Dynamic Timestep Algorithm

Each of these algorithms uses a dynamically changing timestep. A dynamically changing timestep increases the accuracy of simulation and reduces the simulation time by varying the value of the timestep over the transient analysis sweep depending upon the stability of the output. Dynamic timestep algorithms increase the timestep value when internal nodal voltages are stable and decrease the timestep value when nodal voltages are changing quickly.

Figure 11-4: Internal Variable Timestep



In Star-Hspice, the timestep algorithm is selected by the LVLTIM option:

- LVLTIM = 0 selects the iteration count algorithm.
- LVLTIM = 1 selects the DVDT timestep algorithm, along with the iteration count algorithm. Operation of the timestep control algorithm is controlled by the setting of the DVDT control option. For LVLTIM = 1 and DVDT = 0, 1, 2, or 3, the algorithm does not use timestep reversal. For DVDT = 4, the algorithm uses timestep reversal.

  The DVDT algorithm is discussed further in “DVDT Dynamic Timestep Algorithm” on page Chapter 1136.
- LVLTIM = 2 selects the truncation timestep algorithm, along with the iteration count algorithm with reversal.
- LVLTIM = 3 selects the DVDT timestep algorithm with timestep reversal, along with the iteration count algorithm. For LVLTIM = 3 and DVDT = 0, 1, 2, 3, or 4, the algorithm uses timestep reversal.

Iteration Count Dynamic Timestep Algorithm

The simplest dynamic timestep algorithm used is the iteration count algorithm. The iteration count algorithm is controlled by the following options:

IMAX	Controls the internal timestep size based on the number of iterations required for a timepoint solution. If the number of iterations per timepoint exceeds the IMAX value, the internal timestep is decreased. Default = 8.
IMIN	Controls the internal timestep size based on the number of iterations required for the previous timepoint solution. If the last timepoint solution took fewer than IMIN iterations, the internal timestep is increased. Default = 3.

Local Truncation Error (LTE) Dynamic Timestep Algorithm

The local truncation error timestep method uses a Taylor series approximation to calculate the next timestep for a transient analysis. This method calculates an internal timestep using the allowed local truncation error. If the calculated timestep is smaller than the current timestep, then the timepoint is set back (timestep reversal) and the calculated timestep is used to increment the time. If the calculated timestep is larger than the current one, then there is no need for a reversal. A new timestep is used for the next timepoint.

The local truncation error timestep algorithm is selected by setting LVLTIM = 2. The control options available with the local truncation error algorithm are:

```
TRTOL (default = 7)
CHGTOL (default = 1e-15)
RELQ (default = 0.01)
```

DVDT Dynamic Timestep Algorithm

Select this algorithm by setting the option LVLTIM to 1 or 3. If you set LVLTIM = 1, the DVDT algorithm does not use timestep reversal. The results for the current timepoint are saved, and a new timestep is used for the next timepoint. If you set LVLTIM = 3, the algorithm uses timestep reversal. If the results are not converging at a given iteration, the results of current timepoint are ignored, time is set back by the old timestep, and a new timestep is used. Therefore, LVLTIM = 3 is more accurate and more time consuming than LVLTIM = 1.

The test the algorithm uses for reversing the timestep depends on the DVDT control option setting. For DVDT = 0, 1, 2, or 3, the decision is based on the SLOPETOL control option value. For DVDT = 4, the decision is based on the settings of 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 that is based on nonlinearity of node voltages, and employs timestep reversals if the LVLTIM option is set to either 1 or 3. The nonlinearity of node voltages is measured through changes in slopes of the voltages. If the change in slope is larger than the setting of the SLOPETOL control option, the timestep is reduced by a factor equal to the setting of the FT control option. The FT option defaults to 0.25. Star-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, the simulation accuracy can be controlled by SLOPETOL and FT. For LVLTIM = 3, the RELVAR and ABSVAR control options also affect the timestep, and therefore affect the simulation accuracy.

You can use options RELVAR and ABSVAR in conjunction with the DVDT option to improve simulation time or accuracy. For faster simulation time, RELVAR and ABSVAR should be increased (although this might decrease accuracy).

Note: If you need backward compatibility with Star-Hspice Release 95.3, use the following option values. Setting .OPTIONS DVDT = 3 sets all of these values automatically.

LVLTIM = 1	RMAX = 2	SLOPETOL = 0.5
FT = FS = 0.25	BYPASS = 0	BYTOL = 0.050

User Timestep Controls

The RMIN, RMAX, FS, FT, and DELMAX control options allow you to control the minimum and maximum internal timestep allowed for the DVDT algorithm. If the timestep falls below the minimum timestep default, the execution of the program halts. For example, an “internal timestep too small” error results when the timestep becomes less than the minimum internal timestep found by TSTEPxRMIN.

Note: RMIN is the minimum timestep coefficient and has a default value of 1e-9. TSTEP is the time increment, and is set in the .TRAN statement.

If DELMAX is set in an .OPTIONS statement, then DVDT = 0 is used. If DELMAX is not specified in an .OPTIONS statement, Star-Hspice computes a DELMAX value. For DVDT = 0, 1, or 2, the maximum internal timestep is:

min[(TSTOP/50), DELMAX, (TSTEPxRMAX)]

The TSTOP time is the transient sweep range set in the .TRAN statement.

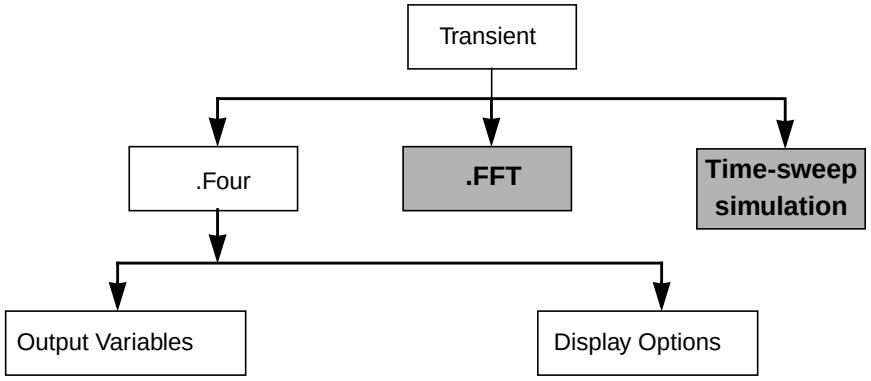
In circuits with piecewise linear (PWL) transient sources, the SLOPETOL option also affects the internal timestep. A PWL source with a large number of voltage or current segments contributes a correspondingly large number of entries to the internal breakpoint table. The number of breakpoint table entries that must be considered contributes to the internal timestep control.

If the difference in the slope of consecutive segments of a PWL source is less than the SLOPETOL value, the breakpoint table entry for the point between the segments is ignored. For a PWL source with a signal that changes value slowly, ignoring its breakpoint table entries can help reduce the simulation time. Since the data in the breakpoint table is a factor in the internal timestep control, reducing the number of usable breakpoint table entries by setting a high SLOPETOL reduces the simulation time.

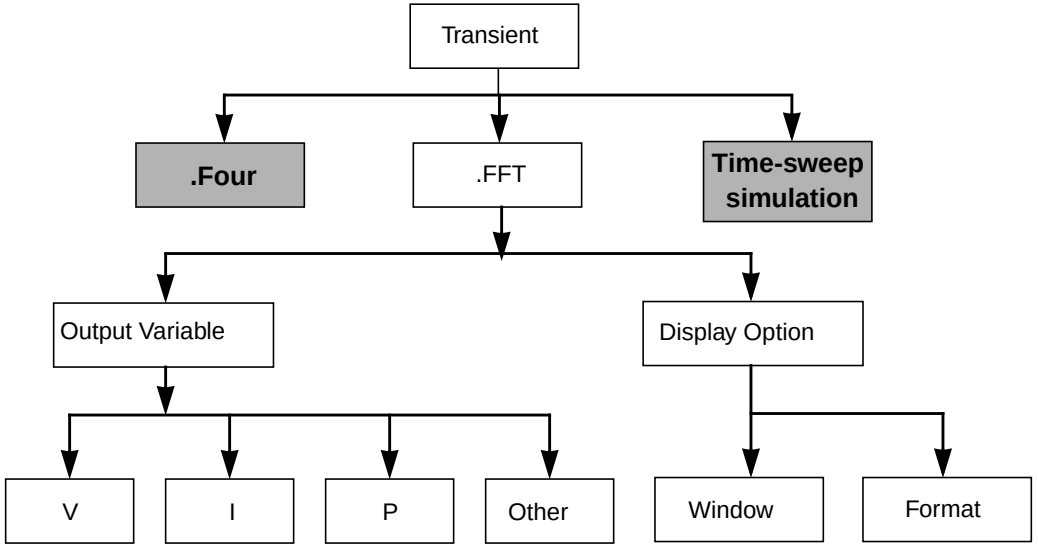
Performing Fourier Analysis

This section describes the flow for Fourier and FFT Analysis.

Figure 11-5: Fourier and FFT Analysis



.FOUR Statement



.FFT Statement

There are two different Fourier analyses available in Star-Hspice: .FOUR and .FFT. The former is the same as is available in SPICE 2G6, a standard, fixed-window analysis tool. The latter is a much more flexible Fourier analysis tool, and is recommended for analysis tasks requiring more detail and precision.

.FOUR Statement

This statement performs a Fourier analysis as a part of the transient analysis. The Fourier analysis is performed over the interval (tstop-fperiod, tstop), where tstop is the final time specified for the transient analysis (see .TRAN statement), and fperiod is one period of the fundamental frequency (parameter “freq”). Fourier analysis is performed on 101 points of transient analysis data on the last 1/f time period, where f is the fundamental Fourier frequency. Transient data is interpolated to fit on 101 points running from (tstop-1/f) to tstop. The phase, the normalized component, and the Fourier component are calculated using 10 frequency bins. The Fourier analysis determines the DC component and the first nine AC components.

Syntax

```
.FOUR freq ov1 <ov2 ov3 ...>
```

freq the fundamental frequency

ov1 … the output variables for which the analysis is desired

Example

```
.FOUR 100K V(5)
```

Accuracy and DELMAX

For maximum accuracy, .OPTION DELMAX should be set to (period/100). For circuits with very high resonance factors (high Q circuits such as crystal oscillators, tank circuits, and active filters) DELMAX should be set to less than (period/100).

Fourier Equation

The total harmonic distortion is the square root of the sum of the squares of the second through the ninth normalized harmonic, times 100, expressed as a percent:

$$THD = \frac{1}{R1} \cdot \left(\sum_{m=2}^{9} R_m^2 \right)^{1/2} \cdot 100\%$$

This interpolation can result in various inaccuracies. For example, if the transient analysis runs at intervals longer than 1/(101*f), the frequency response of the interpolation dominates the power spectrum. Furthermore, there is no error range derived for the output.

The Fourier coefficients are calculated from:

$$g(t) = \sum_{m=0}^{9} C_m \cdot \cos(mt) + \sum_{m=0}^{9} D_m \cdot \sin(mt)$$

where

$$C_m = \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} g(t) \cdot \cos(m \cdot t) \cdot dt$$

$$D_m = \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} g(t) \cdot \sin(m \cdot t) \cdot dt$$

$$g(t) = \sum_{m=0}^{9} C_m \cdot \cos(m \cdot t) + \sum_{m=0}^{9} D_m \cdot \sin(m \cdot t)$$

C and D are approximated by:

$$C_m = \sum_{n=0}^{101} g(n \cdot \Delta t) \cdot \cos\left(\frac{2 \cdot \pi \cdot m \cdot n}{101}\right)$$

$$D_m = \sum_{n=0}^{101} g(n \cdot \Delta t) \cdot \sin\left(\frac{2 \cdot \pi \cdot m \cdot n}{101}\right)$$

The magnitude and phase are calculated by:

$$R_m = (C_m^2 + D_m^2)^{1/2}$$

$$\Phi_m = \arctan\left(\frac{C_m}{D_m}\right)$$

Example

The following is Star-Hspice input for an .OP, .TRAN, and .FOUR analysis.

```
CMOS INVERTER
*
M1 2 1 0 0 NMOS W = 20U L = 5U
M2 2 1 3 3 PMOS W = 40U L = 5U
VDD 3 0 5
VIN 1 0 SIN 2.5 2.5 20MEG
.MODEL NMOS NMOS LEVEL = 3 CGDO = .2N CGSO = .2N
+ CGBO = 2N
.MODEL PMOS PMOS LEVEL = 3 CGDO = .2N CGSO = .2N
+CGBO = 2N
.OP
.TRAN 1N 100N
.FOUR 20MEG V(2)
.PRINT TRAN V(2) V(1)
.END
```

Output for the Fourier analysis is shown below.

```
******
cmos inverter
**** fourier analysis    tnom = 25.000 temp = 25.000 ****
fourier components of transient response v(2)
dc component = 2.430D+00
```

harmonic no	frequency (hz)	fourier component	normalized component	phase (deg)	normalized phase (deg)
1	20.0000x	3.0462	1.0000	176.5386	0.
2	40.0000x	115.7006m	37.9817m	-106.2672	-282.8057
3	60.0000x	753.0446m	247.2061m	170.7288	-5.8098
4	80.0000x	77.8910m	25.5697m	-125.9511	-302.4897
5	100.0000x	296.5549m	97.3517m	164.5430	-11.9956
6	120.0000x	50.0994m	16.4464m	-148.1115	-324.6501
7	140.0000x	125.2127m	41.1043m	157.7399	-18.7987
8	160.0000x	25.6916m	8.4339m	172.9579	-3.5807
9	180.0000x	47.7347m	15.6701m	154.1858	-22.3528

```
 total harmonic distortion =    27.3791   percent
```

For further information on Fourier analysis, see “Performing Fourier Analysis” on page Chapter 1139.

.FFT Statement

The syntax of the .FFT statement is shown below. The parameters are described in Table 11-1.

Syntax

```
.FFT <output_var> <START = value> <STOP = value> <NP = value>
+ <FORMAT = keyword> <WINDOW = keyword> <ALFA = value>
+ <FREQ = value> <FMIN = value> <FMAX = value>
```

Table 11-1: .FFT Statement Parameters

Parameter	Default	Description
output_var	—	Can be any valid output variable, such as voltage, current, or power
START	see Description	Specifies the beginning of the output variable waveform to be analyzed. Defaults to the START value in the .TRAN statement, which defaults to 0.
FROM	see START	In .FFT statements, FROM is an alias for START.
STOP	see Description	Specifies the end of the output variable waveform to be analyzed. Defaults to the TSTOP value in the .TRAN statement.
TO	see STOP	In .FFT statements, TO is an alias for STOP.
NP	1024	Specifies the number of points used in the FFT analysis. NP must be a power of 2. If NP is not a power of 2, Star-Hspice automatically adjusts it to the closest higher number that is a power of 2.
FORMAT	NORM	Specifies the output format: ■ NORM = normalized magnitude. ■ UNORM = unnormalized magnitude

Table 11-1: .FFT Statement Parameters *(Continued)*

Parameter	Default	Description
WINDOW	RECT	Specifies the window type to be used: ■ RECT = simple rectangular truncation window ■ BART = Bartlett (triangular) window ■ HANN = Hanning window ■ HAMM = Hamming window ■ BLACK = Blackman window ■ HARRIS = Blackman-Harris window ■ GAUSS = Gaussian window ■ KAISER = Kaiser-Bessel window
ALFA	3.0	Specifies the parameter used in GAUSS and KAISER windows to control the highest side-lobe level, bandwidth, and so on. 1.0 < = ALFA < = 20.0
FREQ	0.0 (Hz)	Specifies a frequency of interest. If FREQ is nonzero, the output listing is limited to the harmonics of this frequency, based on FMIN and FMAX. The THD for these harmonics also is printed.
FMIN	1.0/T (Hz)	Specifies the minimum frequency for which FFT output is printed in the listing file, or which is used in THD calculations. T = (STOP-START)
FMAX	0.5*NP*FMIN (Hz)	Specifies the maximum frequency for which FFT output is printed in the listing file, or which is used in THD calculations.

Example

Below are four examples of valid .FFT statements.

```
.fft v(1)

.fft v(1,2) np = 1024 start = 0.3m stop = 0.5m
+ freq = 5.0k window = kaiser alpha = 2.5

.fft I(rload) start = 0m to = 2.0m fmin = 100k
+ fmax = 120k format = unorm

.fft par('v(1) + v(2)') from = 0.2u stop = 1.2u
+ window = harris
```

Only one output variable is allowed in an .FFT command. The following is an *incorrect* use of the command.

```
.fft v(1) v(2) np = 1024
```

The correct use of the command is shown in the example below. In this case, a *.ft0* and a *.ft1* file are generated for the FFT of v(1) and v(2), respectively.

```
.fft v(1) np = 1024
.fft v(2) np = 1024
```

FFT Analysis Output

The results of the FFT analysis are printed in a tabular format in the *.lis* file, based on the parameters in the .FFT statement. The normalized magnitude values are printed unless you specify FORMAT = UNORM, in which case unnormalized magnitude values are printed. The number of printed frequencies is half the number of points (NP) specified in the .FFT statement. If you specify a minimum or a maximum frequency, using FMIN or FMAX, the printed information is limited to the specified frequency range. Moreover, if you specify a frequency of interest using FREQ, then the output is limited to the harmonics of this frequency, along with the percent of total harmonic distortion.

A *.ft#* file is generated, in addition to the listing file, for each FFT output variable, which contains the graphic data needed to display the FFT analysis waveforms. The magnitude in dB and the phase in degrees are available for display.

In the sample FFT analysis *.lis* file output below, notice that all the parameters used in the FFT analysis are defined in the header.

```
****** 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 the THD up to 10 kHz, which corresponds to the first ten harmonics.

Note: The highest frequency shown in the Star-Hspice FFT output might not be identical to the specified FMAX, due to Star-Hspice adjustments.

Table 11-2 describes the output of the Star-Hspice FFT analysis.

Table 11-2: .FFT Statement 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 index 0 is displayed independently.
frequency	Actual frequency associated with the index
fft_mag (db), fft_mag	There are two FFT magnitude columns: the first in dB and the second in the units of the output variable. The magnitude is normalized unless UNORM format is specified.
fft_phase	Associated phase, in degrees

Notes:

1. Use the following formula as a guideline when specifying a frequency range for FFT output:

   ```
   frequency increment = 1.0/(STOP - START)
   ```

   Each frequency index corresponds to a multiple of this increment. To obtain a finer frequency resolution, maximize the duration of the time window.

2. FMIN and FMAX have no effect on the *.ft0*, *.ft1*, ..., *.ftn* files.

For further information on the .FFT statement, see Using the .FFT Statement on page 19-7.

Avant!

Chapter 12

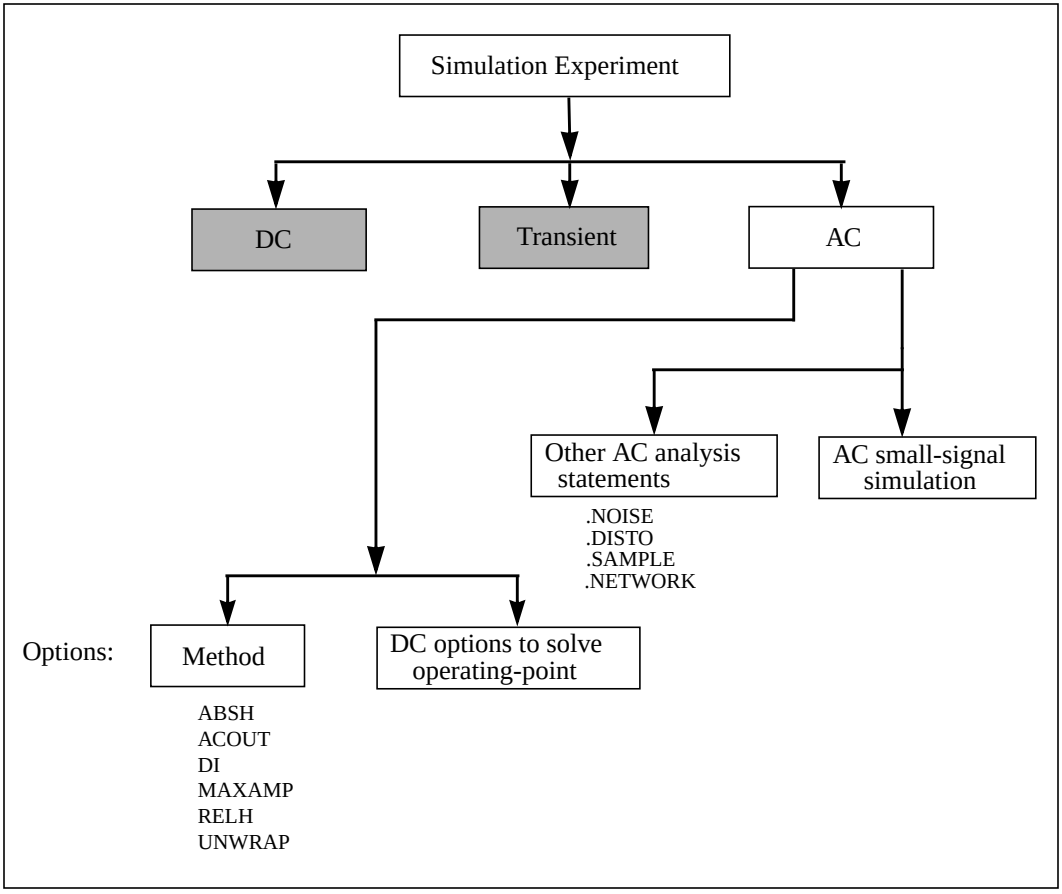
AC Sweep and Small Signal Analysis

This chapter describes performing an AC sweep and small signal analysis. It covers the following topics:

Understanding AC Small Signal Analysis

The AC small signal analysis portion of Star-Hspice computes (see Figure 12-1) AC output variables as a function of frequency. Star-Hspice first solves for the DC operating point conditions, which are used to develop linearized, small-signal models for all nonlinear devices in the circuit.

Figure 12-1: AC Small Signal Analysis Flow



Capacitor and inductor values are converted to their corresponding admittances:

$$Y_C = j\omega C \quad \text{for capacitors}$$

and

$$Y_L = 1/j\omega L \quad \text{for inductors}$$

Star-Hspice allows resistors to have different DC and AC values. If AC = <value> is specified in a resistor statement, the operating point is calculated using the DC value of resistance, but the AC resistance value is used in the AC analysis. This is convenient when analyzing operational amplifiers, since the operating point computation can be performed on the unity gain configuration using a low value for the feedback resistance. The AC analysis then can be performed on the open loop configuration by using a very large value for the AC resistance.

AC analysis of bipolar transistors is based on the small-signal equivalent circuit, as described in Chapter 4, “Using BJT Models”, in the *True-Hspice Device Models Reference Manual*. MOSFET AC equivalent circuit models are described in Chapter 8, “Introducing MOSFETs”, in the *True-Hspice Device Models Reference Manual*.

The AC analysis statement permits sweeping values for:

- Frequency
- Element
- Temperature
- Model parameter
- Randomized distribution (Monte Carlo)
- Optimization and AC design analysis

Additionally, as part of the small signal analysis tools, Star-Hspice provides:

- Noise analysis
- Distortion analysis
- Network analysis
- Sampling noise

Using the .AC Statement

You can use the .AC statement in several different formats, depending on the application, as shown in the examples below. The parameters are described below.

Syntax

Single/double sweep

```
.AC type np fstart fstop
```

or

```
.AC type np fstart fstop <SWEEP var start stop incr>
```

or

```
.AC type np fstart fstop <SWEEP var type np start stop>
```

or

```
.AC var1 START = <param_expr1> STOP = <param_expr2>
+  STEP = <param_expr3>
```

or

```
.AC var1 START = start1 STOP = stop1 STEP = incr1
```

Parameterized sweep

```
.AC type np fstart fstop <SWEEP DATA = datanm>
```

or

```
.AC DATA = datanm
```

Optimization

```
.AC DATA = datanm OPTIMIZE = opt_par_fun
+ RESULTS = measnames MODEL = optmod
```

Random/Monte Carlo

```
.AC type np fstart fstop <SWEEP MONTE = val>
```

The .AC statement keywords and parameters have the following descriptions:

DATA = datanm	Data name referred to in the .AC statement
incr	voltage, current, element or model parameter increment value **Note:** If "type" variation is used, the "np" (number of points) is specified instead of "incr".
fstart	Starting frequency **Note:** If type variation "POI" (list of points) is used, a list of frequency values is specified instead of "fstart fstop".
fstop	Final frequency
MONTE = val	Produces a number *val* of randomly-generated values that are used to select parameters from a distribution. The distribution can be *Gaussian*, *Uniform*, or *Random Limit*. See Performing Monte Carlo Analysis on page 13-14 for more information.
np	Number of points per decade or per octave, or just number of points, depending on the preceding keyword
start	Starting voltage, current, any element or model parameter value

stop	Final voltage, current, any element or model parameter value
SWEEP	Keyword to indicate a second sweep is specified in the .AC statement
TEMP	Keyword to indicate a temperature sweep
type	Can be any of the following keywords: ■ DEC – decade variation ■ OCT – octave variation ■ LIN – linear variation ■ POI – list of points
var	Name of an independent voltage or current source, any element or model parameter, or the keyword TEMP (indicating a temperature sweep). Star-Hspice supports source value sweep, referring to the source name (SPICE style). However, if parameter sweep, a .DATA statement, and temperature sweep are selected, a parameter name must be chosen for the source value and subsequently referred to in the .AC statement. The parameter name can not start with V or I.

Example

The following example performs a frequency sweep by 10 points per decade from 1 kHz to 100 MHz.

```
.AC DEC 10 1K 100MEG
```

The next line calls for a 100 point frequency sweep from 1 Hz to 100 Hz.

```
.AC LIN 100 1 100HZ
```

The following example performs an AC analysis for each value of cload, which results from a linear sweep of cload between 1 pF and 10 pF (20 points), sweeping frequency by 10 points per decade from 1 Hz to 10 kHz.

```
.AC DEC 10 1 10K SWEEP cload LIN 20 1pf 10pf
```

The following example performs an AC analysis for each value of rx, 5 k and 15 k, sweeping frequency by 10 points per decade from 1 Hz to 10 kHz.

```
.AC DEC 10 1 10K SWEEP rx n POI 2 5k 15k
```

The next example uses the DATA statement to perform a series of AC analyses modifying more than one parameter. The parameters are contained in the file *datanm*.

```
.AC DEC 10 1 10K SWEEP DATA = datanm
```

The following example illustrates a frequency sweep along with a Monte Carlo analysis with 30 trials.

```
.AC DEC 10 1 10K SWEEP MONTE = 30
```

When an .AC statement is included in the input file, Star-Hspice performs an AC analysis of the circuit over the specified frequency range for each parameter value specified in the second sweep.

For an AC analysis, at least one independent AC source element statement must be in the data file (for example, VI INPUT GND AC 1V). Star-Hspice checks for this condition and reports a fatal error if no such AC sources have been specified (see Using Sources and Stimuli on page 5-1).

AC Control Options

ABSH = x	Sets the absolute current change through voltage defined branches (voltage sources and inductors). In conjunction with DI and RELH, ABSH is used to check for current convergence. Default = 0.0.

ACOUT	AC output calculation method for the difference in values of magnitude, phase and decibels for prints and plots. Default = 1. The default value, ACOUT = 1, selects the Star-Hspice method, which calculates the difference of the magnitudes of the values. The SPICE method, ACOUT = 0, calculates the magnitude of the differences.
DI = x	Sets the maximum iteration-to-iteration current change through voltage defined branches (voltage sources and inductors). This option is only applicable when the value of the DI control option is greater than 0. Default = 0.0.
MAXAMP = x	Sets the maximum current through voltage defined branches (voltage sources and inductors). If the current exceeds the MAXAMP value, an error message is issued. Default = 0.0.
RELH = x	Sets relative current tolerance through voltage defined branches (voltage sources and inductors). It is used to check current convergence. This option is applicable only if the value of the ABSH control option is greater than zero. Default = 0.05.
UNWRAP	Displays phase results in AC analysis in unwrapped form (with a continuous phase plot). This allows accurate calculation of group delay. Note that group delay is always computed based on unwrapped phase results, even if the UNWRAP option is not set.

Using Other AC Analysis Statements

This section describes how to use other AC analysis statements.

.DISTO Statement — AC Small-Signal Distortion Analysis

The .DISTO statement causes Star-Hspice to compute the distortion characteristics of the circuit in an AC small-signal, sinusoidal, steady-state analysis.The program computes and reports five distortion measures at the specified load resistor. The analysis 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 parameter skw2, which is the ratio F2/F1.

DIM2	Intermodulation distortion, difference. The relative magnitude and phase of the frequency component (F1 - F2).
DIM3	Intermodulation distortion, second difference. The relative magnitude and phase of the frequency component (2 · F1 - F2).
HD2	Second-order harmonic distortion. The relative magnitude and phase of the frequency component 2 · F1 (ignoring F2).
HD3	Third-order harmonic distortion. The relative magnitude and phase of the frequency component 3 · F1 (ignoring F2).
SIM2	Intermodulation distortion, sum. The relative magnitude and phase of the frequency component (F1 + F2).

The .DISTO summary report includes a set of distortion measures for each contributing component of every element, a summary set for each element, and a set of distortion measures representing a sum over all the elements in the circuit.

Syntax

```
.DISTO Rload <inter <skw2 <refpwr <spwf>>>>
```

where:

Rload	The resistor element name of the output load resistor into which the output power is fed
inter	Interval at which a distortion-measure summary is to be printed. Specifies a number of frequency points in the AC sweep (see the np parameter in Using the .AC Statement on page 12-4). If inter is omitted or set to zero, no summary printout is made. In this case, the distortion measures can be printed or plotted with the .PRINT or .PLOT statement. If inter is set to 1 or higher, a summary printout is made for the first frequency, and once for each inter frequency increment thereafter. To obtain a summary printout for only the first and last frequencies, set inter equal to the total number of increments needed to reach fstop in the .AC statement. For a summary printout of only the first frequency, set inter to greater than the total number of increments required to reach fstop.
skw2	Ratio of the second frequency F2 to the nominal analysis frequency F1. The acceptable range is $1e\text{-}3 < skw2 \leq 0.999$. If skw2 is omitted, a value of 0.9 is assumed.

refpwr	Reference power level used in computing the distortion products. If omitted, a value of 1mW, measured in decibels magnitude (dbM), is assumed. The value must be ≥ 1e-10.
spwf	Amplitude of the second frequency F2. The value must be ≥ 1e-3. Default = 1.0.

Example

```
.DISTO RL 2 0.95 1.0E-3 0.75
```

Only one distortion analysis is performed per simulation. If more than one .DISTO statement is found, only the last is performed. The DISTO statement calculates distortions for diodes, BJTs, and MOSFETs (Level49 and Level53, Version 3.22).

Note: The summary printout from the distortion analysis for each frequency listed is extensive. Use the “inter” parameter in the .DISTO statement to limit the amount of output generated.

.NOISE Statement — AC Noise Analysis

Syntax

```
.NOISE ovv srcnam inter
```

where:

ovv	Nodal voltage output variable defining the node at which the noise is summed
srcnam	Name of the independent voltage or current source to be used as the noise input reference

inter	Interval at which a noise analysis summary is to be printed, inter specifies a number of frequency points summary in the AC sweep. If inter is omitted or set to zero, no summary printout is made. If inter is equal to or greater than one, a summary printout is made for the first frequency, and once for each inter frequency increment thereafter.

Example

```
.NOISE V(5) VIN 10
```

The .NOISE statement, used in conjunction with the AC statement, controls the noise analysis of the circuit.

Noise Calculations

The noise calculations in Star-Hspice are based on the complex AC nodal voltages, which in turn are based on the DC operating point. Noise models are described for each device type in the appropriate chapter in Volume II. A noise source is not assumed to be statistically correlated to the other noise sources in the circuit; each noise source is calculated independently. The total output noise voltage is the RMS sum of the individual noise contributions:

$$onoise = \sum_{n=1}^{n} |Z_n \cdot I_n|^2$$

where:

onoise	Total output noise
I	Equivalent current due to thermal noise, shot or flicker noise
Z	Equivalent transimpedance between noise source and the output
n	Number of noise sources associated with all resistors, MOSFETs, diodes, JFETs, and BJTs

The equivalent input noise voltage is the total output noise divided by the gain or transfer function of the circuit. The contribution of each noise generator in the circuit is printed for each inter frequency point. The output and input noise levels are normalized with respect to the square root of the noise bandwidth, and have the units volts/$Hz^{1/2}$ or amps/$Hz^{1/2}$.

You can simulate flicker noise sources in the noise analysis by including values for the parameters KF and AF on the appropriate device model statements.

Use the .PRINT or .PLOT statement to print or plot the output noise and the equivalent input noise.

You can only perform one noise analysis per simulation. If more than one NOISE statement is present, only the last one is performed.

.SAMPLE Statement — 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 causes Star-Hspice to perform a simple noise folding analysis at the output node.

Syntax

```
.SAMPLE FS = freq <TOL = val> <NUMF = val>
<MAXFLD = val>
+ <BETA = val>
```

where:

FS = freq	Sample frequency, in Hertz
TOL	Sampling error tolerance: the ratio of the noise power in the highest folding interval to the noise power in baseband. Default = 1.0e-3.

NUMF	Maximum allowed number of user-specified frequencies. The algorithm requires approximately ten times this number of internally generated frequencies, so it should be kept small. Default = 100.
MAXFLD	Maximum allowed number of folding intervals. The highest frequency (in Hertz) considered by the algorithm is given by: FMAX = MAXFLD · FS Default = 10.0.
BETA	Integrator duty cycle; specifies an optional noise integrator at the sampling node BETA = 0 no integrator BETA = 1 simple integrator (default) If the integrator is clocked (that is, it only integrates during a fraction of the sampling interval 1/FS), then BETA should be set to the duty cycle of the integrator.

.NET Statement - AC Network Analysis

The .NET statement computes the parameters for the impedance matrix Z, the admittance matrix Y, the hybrid matrix H, and the scattering matrix S. The input impedance, output impedance, and admittance are also computed. This analysis is a part of the AC small-signal analysis. Therefore, network analysis requires the specification of the AC statement frequency sweep.

Syntax

One-port network

```
.NET input <RIN = val>
```

or

```
.NET input <val >
```

Two-port network

```
.NET output input <ROUT = val> <RIN = val>
```

where:

input	AC input voltage or current source name
output	Output port. It can be an output voltage, V(n1,n2), or an output current, I(source), or I(element).
RIN	Input or source resistance keyword. The RIN value is used to calculate the output impedance and admittance, and also the scattering parameters. The RIN value defaults to 1 ohm.
ROUT	Output or load resistance keyword. The ROUT value is used to calculate the input impedance and admittance, and also the scattering parameters. The ROUT value defaults to 1 ohm.

Example

One-port network

```
.NET    VINAC   RIN = 50
.NET    IIN     RIN = 50
```

Two-port network

```
.NET V(10,30)      VINAC    ROUT = 75     RIN = 50
.NET I(RX)         VINAC    ROUT = 75     RIN = 50
```

AC Network Analysis - Output Specification

Syntax

```
Xij(z), ZIN(z), ZOUT(z), YIN(z), YOUT(z)
```

where:

X	Specifies Z for impedance, Y for admittance, H for hybrid, and S for scattering
ij	i and j can be 1 or 2. They identify which matrix parameter is to be printed.
z	Output type: ■ R: real part ■ I: imaginary part ■ M: magnitude ■ P: phase ■ DB: decibel ■ T: group time delay
ZIN	Input impedance. For the one port network, ZIN, Z11 and H11 are the same.
ZOUT	Output impedance
YIN	Input admittance. For the one port network, YIN and Y11 are the same.
YOUT	Output admittance

If you omit “z”, output includes the magnitude of the output variable.

Example

```
.PRINT  AC Z11(R)  Z12(R)  Y21(I)  Y22 S11 S11(DB) Z11(T)
.PRINT  AC ZIN(R) ZIN(I) YOUT(M) YOUT(P) H11(M) H11(T)
.PLOT   AC S22(M) S22(P) S21(R) H21(P) H12(R) S22(T)
```

Bandpass Netlist:[1] Star-Hspice Network Analysis Results

```
*FILE: FBP_1.SP
.OPTIONS DCSTEP = 1 POST
*BAND PASS FILTER

C1       IN       2        3.166PF
L1        2       3          203NH
C2        3       0        3.76PF
C3        3       4        1.75PF
C4        4       0        9.1PF
L2        4       0       36.81NH
C5        4       5        1.07PF
C6        5       0        3.13PF
L3        5       6      233.17NH
C7        6       7        5.92PF
C8        7       0        4.51PF
C9        7       8        1.568PF
C10       8       0        8.866PF
L4        8       0       35.71NH
C11       8       9        2.06PF
C12       9       0        4.3PF
L5        9      10      200.97NH
C13      10     OUT        2.97PF
RX      OUT       0        1E14
VIN      IN       0        AC 1

.AC LIN 41 200MEG 300MEG
.NET V(OUT) VIN ROUT = 50 RIN = 50
.PLOT AC S11(DB) (-50,10) S11(P) (-180,180)
.PLOT AC ZIN(M) (5,130) ZIN(P) (-90,90)
.END
```

Figure 12-2: S11 Magnitude and Phase Plots

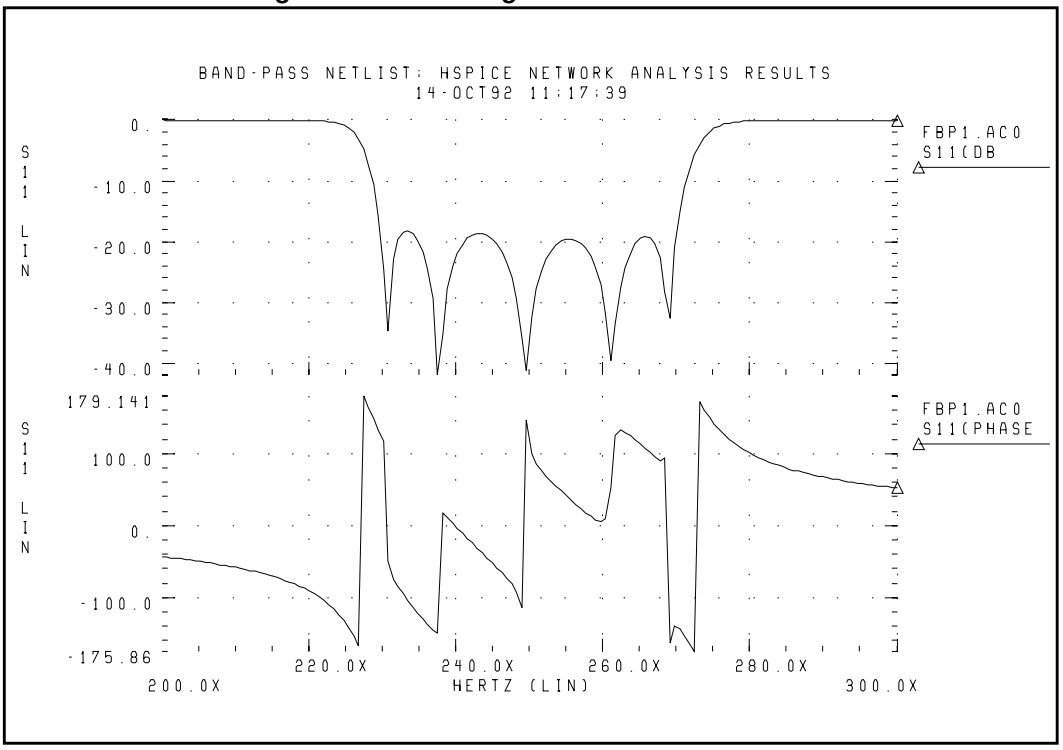
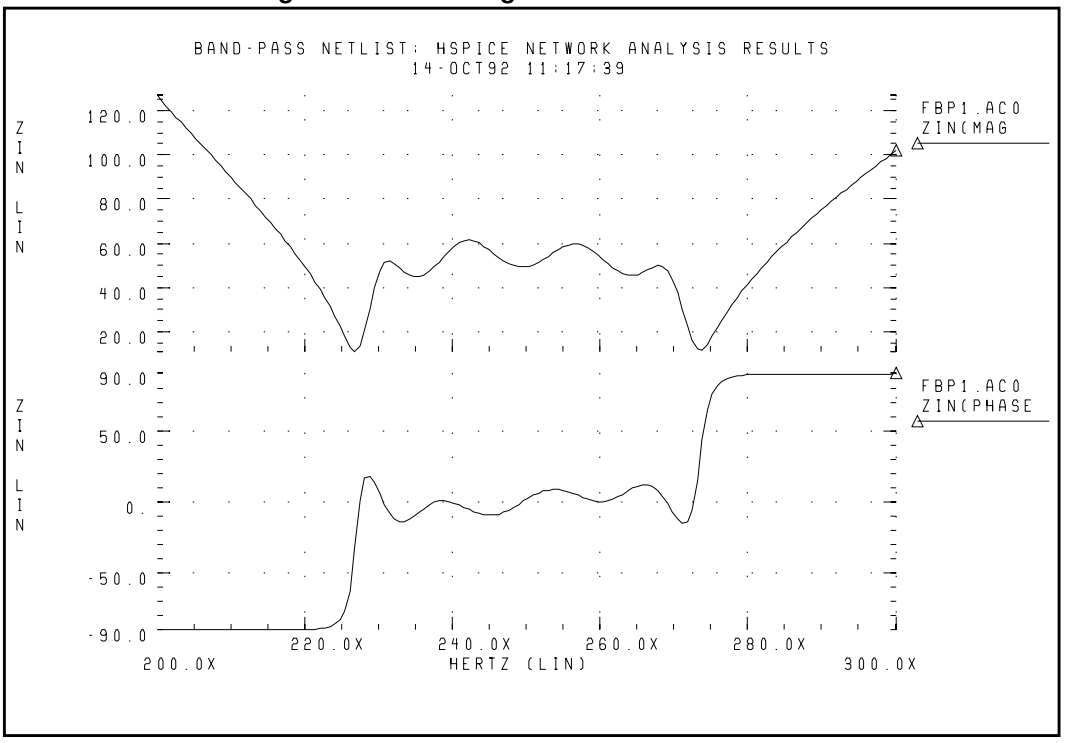


Figure 12-3: ZIN Magnitude and Phase Plots



NETWORK Variable Specification

Star-Hspice uses the results of AC analysis to perform network analysis. The .NET statement defines the Z, Y, H, and S parameters to be calculated.

The following list shows various combinations of the .NET statement for network matrices that are initially calculated in Star-Hspice:

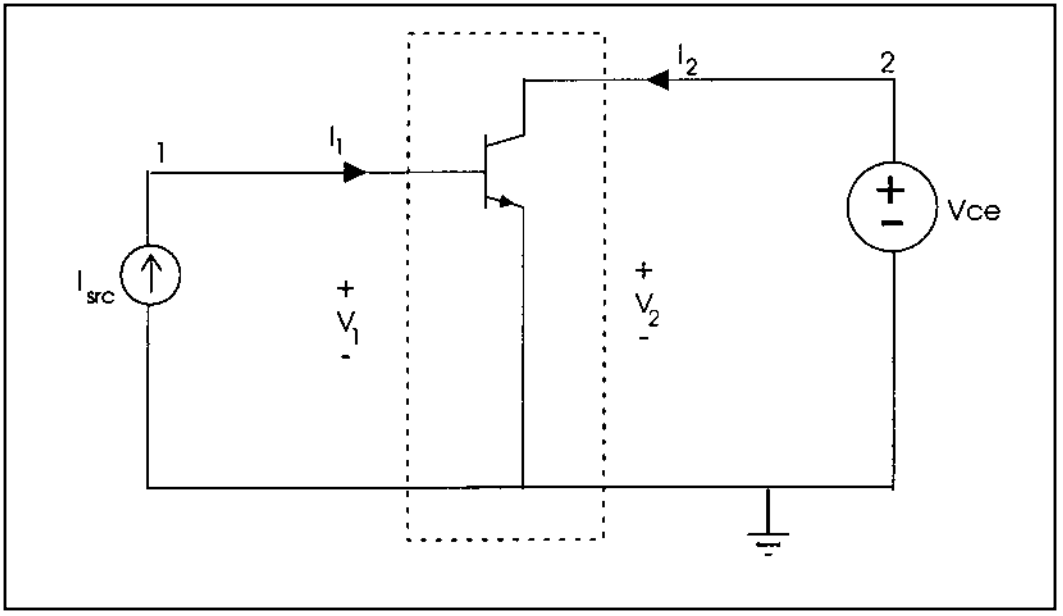
1.	.NET Vout Isrc	V	=	[Z]	[I]
2.	.NET Iout Vsrc	I	=	[Y]	[V]
3.	.NET Iout Isrc	$[V1\ I2]^T$	=	[H]	$[I1\ V2]^T$
4.	.NET Vout Vsrc	$[I1\ V2]^T$	=	[S]	$[V1\ I2]^T$

($[M]^T$ represents the *transpose* of matrix M)

Note: The preceding list does not mean that combination (1) must be used for calculating the Z parameters. However, if .NET Vout Isrc is specified, Star-Hspice initially evaluates the Z matrix parameters and then uses standard conversion equations to determine the S parameters or any other requested parameters.

The example in Figure 12-4 shows the importance of the variables used in the .NET statement. Here, *Isrc* and *Vce* are the DC biases applied to the BJT.

Figure 12-4: Parameters with .NET V(2) Isrc



This .NET statement provides an incorrect result for the Z parameters calculation:

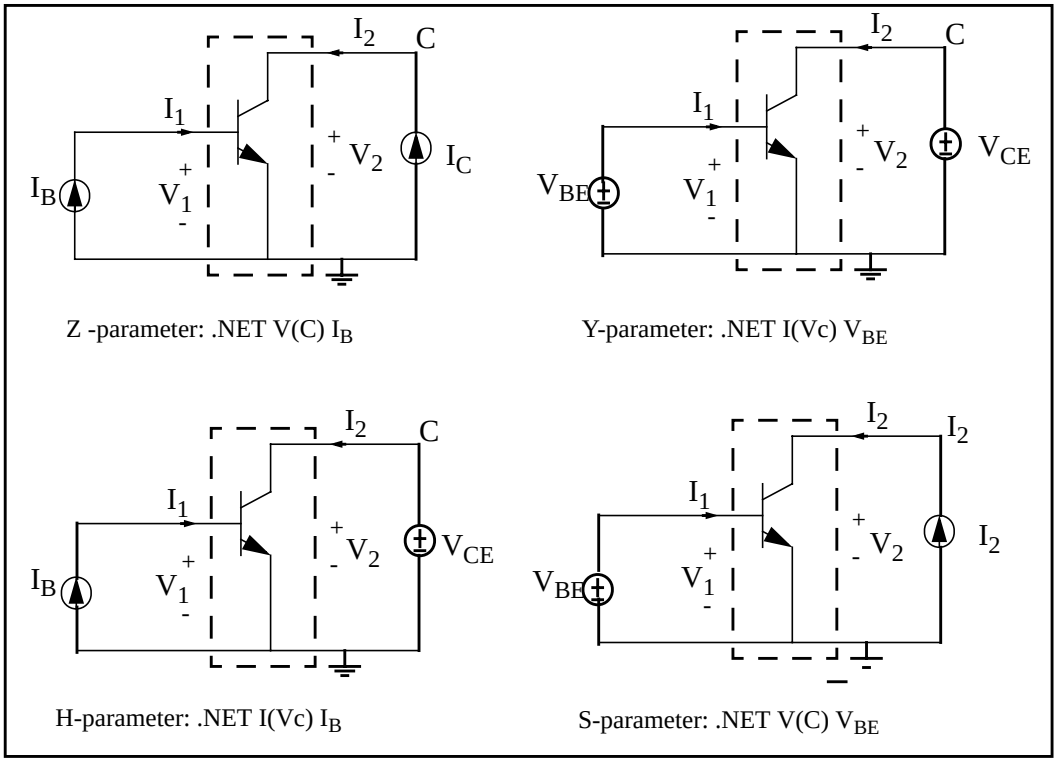
```
.NET V(2) Isrc
```

When Star-Hspice performs AC analysis, all the DC voltage sources are shorted and all the DC current sources are open-circuited. As a result, V(2) is shorted to ground, and its value is zero for AC analysis, directly affecting the results of the network analysis.

When Star-Hspice attempts to calculate the Z parameters Z11 and Z21, defined as Z11 = V1/I1 and Z21 = V2/I1 with I2=0, the requirement that I2 must be zero is not satisfied in the circuit above. Instead, V2 is zero, which results in incorrect values for Z11 and Z21.

The correct biasing configurations for performing network analysis for Z, Y, H, and S parameters are shown in Figure 12-5.

Figure 12-5: Network Parameter Configurations



As an example, the H parameters are calculated by using the .NET statement.

```
.NET I(VC) IB
```

Here, V_C denotes the voltage at node C, the collector of the BJT. With this statement, Star-Hspice calculates the H parameters immediately after AC analysis. The H parameters are calculated by:

$$V1 = H11 \cdot I1 + H12 \cdot V2$$

$$I2 = H21 \cdot I1 + H22 \cdot V2$$

For Hybrid parameter calculations of H11 and H21, V2 is set to zero (due to the DC voltage source V_{CE}), while for H12 and H22 calculations, I1 is set to zero (due to the DC current source I_B). Setting I1 and V2 equal to zero precisely meets the conditions of the circuit under examination; namely, that the input current source is open circuited and the output voltage source is shorted to ground.

External DC biases applied to a BJT can be driven by a data file of measured results. In some cases, not all of the DC currents and voltages at input and output ports are available. When performing network analysis, examine the circuit and select suitable input and output variables to obtain correctly calculated results. The following examples demonstrate the network analysis of a BJT using Star-Hspice.

Network Analysis Example: Bipolar Transistor

```
BJT network analysis
.option nopage list
+ newtol reli = 1e-5 absi = 1e-10 relv = 1e-5
+ relvdc = 1e-7 nomod post gmindc = 1e-12
.op
.param vbe = 0 ib = 0 ic = 0 vce = 0

$ H-parameter
.NET i(vc) ibb rin = 50 rout = 50
ve      e    0            0
ibb     0    b            dc = 'ib' ac = 0.1
vc      c    0            'vce'
q1      c    b    e 0     bjt

.model bjt npn subs = 1
+ bf = 1.292755e+02 br = 8.379600e+00
+ is = 8.753000e-18 nf = 9.710631e-01
+ nr = 9.643484e-01 ise = 3.428000e-16
+ isc = 1.855000e-17 iss = 0.000000e+00
```

```
+ ne = 2.000000e+00 nc = 9.460594e-01
+ ns = 1.000000e+00 vaf = 4.942130e+01
+ var = 4.589800e+00 ikf = 5.763400e-03
+ ikr = 5.000000e-03 irb = 8.002451e-07
+ rc = 1.216835e+02 rb = 1.786930e+04
+ rbm = 8.123460e+01 re = 2.136400e+00
+ cje = 9.894950e-14 mje = 4.567345e-01
+ vje = 1.090217e+00 cjc = 5.248670e-14
+ mjc = 1.318637e-01 vjc = 5.184017e-01
+ xcjc = 6.720303e-01 cjs = 9.671580e-14
+ mjs = 2.395731e-01 vjs = 5.000000e-01
+ tf = 3.319200e-11 itf = 1.457110e-02
+ xtf = 2.778660e+01 vtf = 1.157900e+00
+ ptf = 6.000000e-05 xti = 4.460500e+00
+ xtb = 1.456600e+00 eg = 1.153300e+00
+ tikf1 = -5.397800e-03 tirb1 = -1.071400e-03
+ tre1 = -1.121900e-02 trb1 = 3.039900e-03
+ trc1 = -4.020700e-03 trm1 = 0.000000e+00

.print ac par('ib') par('ic')
+ h11(m) h12(m) h21(m) h22(m)
+ z11(m) z12(m) z21(m) z22(m)
+ s11(m) s21(m) s12(m) s22(m)
+ y11(m) y21(m) y12(m) y22(m)

.ac Dec 10 1e6 5g sweep data = bias

.data bias
    vbe             vce             ib             ic
771.5648m       292.5047m       1.2330u     126.9400u
797.2571m       323.9037m       2.6525u     265.0100u
821.3907m       848.7848m       5.0275u     486.9900u
843.5569m       1.6596          8.4783u     789.9700u
864.2217m       2.4031         13.0750u       1.1616m
884.3707m       2.0850         19.0950u       1.5675m
.enddata
.end
```

Other possible biasing configurations for the network analysis follow.

$S-parameter

```
.NET v(c) vbb rin = 50 rout = 50

ve      e    0            0
vbb     b    0            dc = 'vbe' ac = 0.1
icc     0    c            'ic'
q1      c    b    e 0     bjt
```

$Z-parameter

```
.NET v(c) ibb rin = 50 rout = 50

ve      e    0            0
ibb     0    b            dc = 'ib' ac = 0.1
icc     0    c            'ic'
q1      c    b    e 0     bjt
```

```
$Y-parameter

.NET i(vc) vbb rin = 50 rout = 50

ve      e    0            0
vbb     b    0            'vbe' ac = 0.1
vc      c    0            'vce'
q1      c    b    e 0     bjt
```

References

1. Goyal, Ravender. “S-Parameter Output From SPICE Program”, *MSN & CT*, February 1988, pp. 63 and 66.

Avant!

Chapter 13
Statistical Analysis and Optimization

An electrical circuit must be designed to the tolerances associated with the specific manufacturing process. The electrical yield refers to the number of parts that meet the electrical test specifications. Maximizing the yield is important for the overall process efficiency. Star-Hspice analyzes and optimizes the yield by using statistical techniques and observing the effects of element and model parameter variation.

Specifying Analytical Model Types

You can model parametric and statistical variation in circuit behavior in Star Hspice by using:

- The .PARAM statement. Use to investigate the performance of a circuit as specified changes in circuit parameters are made. See “Simulation Input and Controls” on page Chapter 31 for details of the .PARAM statement.
- Temperature Variation Analysis. The circuit and component temperatures are varied and the circuit responses compared. The temperature-dependent effects of the circuit can be studied in detail.
- Monte Carlo Analysis. The statistical standard deviations of component values are known. Use for centering a design for maximum process yield, or for determining component tolerances.
- Worst Case Corners Analysis. The component value limits are known. Automates quality assurance for basic circuit function for process extremes, quick estimation of speed and power trade-offs, and best case and worst case model selection through parameter corners library files.
- Data Driven Analysis. Use for cell characterization, response surface, or Taguchi analysis. See “Performing Cell Characterization” on page Chapter 151. Automates cell characterization, including timing simulator polynomial delay coefficient calculation. There is no limit on the number of parameters simultaneously varied or number of analyses to be performed. Convenient ASCII file format for automated parameter value generation. Can replace hundreds or thousands of Star-Hspice runs.

Yield analyses are used to modify DC operating points, DC sweeps, AC sweeps, and transient analysis. They can generate scatter plots for operating point analysis and family of curves plots for DC, AC, and transient analysis.

The .MEASURE statement in Star-Hspice is used with yield analyses, allowing you to see distributions of delay times, power, or any other characteristic described with a .MEASURE statement. Often, this is more useful than viewing a family of curves generated by a Monte Carlo analysis.

When you use the .MEASURE statement, Star-Hspice generates a table of results as an *.mt#* file, which is in readable ASCII format, and can be displayed in AvanWaves. Also, when .MEASURE statements are used in a Monte Carlo or data driven analysis, the Star-Hspice output file includes calculations for standard statistical descriptors:

$$\text{Mean} = \frac{x_1 + x_2 + \ldots + x_n}{N}$$

$$\text{Variance} = \frac{(x_1 - Mean)^2 + \ldots (x_n - Mean)}{N - 1}$$

$$\text{Sigma} = \sqrt{Variance}$$

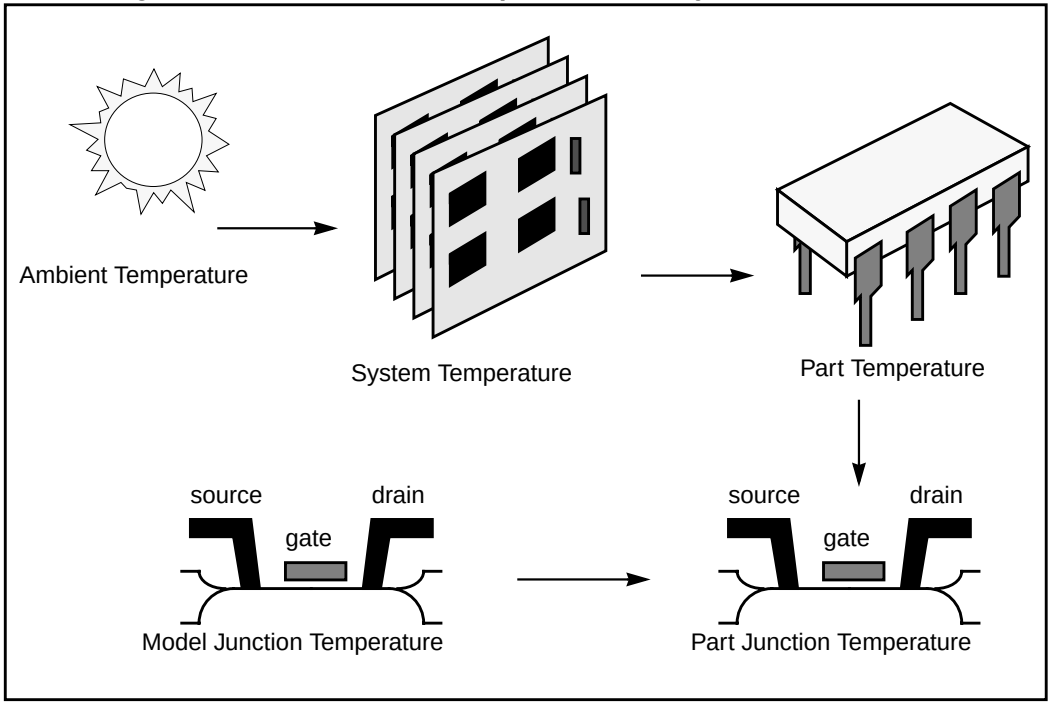
$$\text{Average Deviation} = \frac{\left|x_1 - Mean\right| + \ldots + \left|x_n - Mean\right|}{N - 1}$$

Simulating Circuit and Model Temperatures

Temperature affects all electrical circuits. The key temperature parameters associated with circuit simulation are (see Figure 13-1):

- Model reference temperature – each model might be measured at a different temperature and each model has a TREF parameter
- Element junction temperature – each resistor, transistor, or other element generates heat and will be 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 that has a local temperature.
- Ambient temperature – the ambient temperature is the air temperature of the system.

Figure 13-1: Part Junction Temperature Sets System Performance



Temperatures in Star-Hspice are calculated as differences from ambient temperature:

$$Tambient + \Delta system + \Delta part + \Delta junction = Tjunction$$

$$Ids = f(Tjunction, Tmodel)$$

Every element in Star-Hspice has a keyword DTEMP. This is the difference between junction and ambient temperature. An example of using DTEMP in a MOSFET element statement is shown below.

```
M1 drain gate source bulk Model_name W=10u
+ L=1u DTEMP=+20
```

Temperature Analysis

Star-Hspice allows you to specify three temperatures:

- Model reference temperature, specified in a .MODEL statement using the TREF parameter (or TEMP or TNOM, for some models). This is the temperature, in °C, at which the model parameters are measured and extracted. The value of TNOM can be set in a .OPTION statement. Its default value is 25 °C.
- Circuit temperature, specified using a .TEMP statement or the TEMP parameter. This is the temperature, in °C, at which all elements are simulated. To modify the temperature for a particular element, you can use the DTEMP parameter. The default circuit temperature is the value of TNOM.
- Individual element temperature, specified as the circuit temperature plus an optional amount specified using the DTEMP parameter

You can specify the temperature of a circuit for a Star-Hspice run with either the .TEMP statement or the TEMP parameter in the .DC, .AC, or .TRAN statements. The circuit simulation temperature set by any of these statements is compared against the reference temperature set by the TNOM option. TNOM defaults to 25 °C unless the option SPICE is used, in which case it defaults to 27 °C. The derating of component values and model parameters is calculated by using the difference of the circuit simulation temperature and the reference temperature, TNOM.

Since elements and models within a circuit can be operating at different temperatures (for example, a high-speed input/output buffer switching at 50 MHz will be much hotter than a low-drive NAND gate switching at 1 MHz), use an element temperature parameter, DTEMP, and a model reference parameter, TREF. Specifying DTEMP in an element statement causes the element temperature for the simulation to be:

```
element temperature = circuit temperature + DTEMP
```

Specify the DTEMP value in the element statement (resistor, capacitor, inductor, diode, BJT, JFET, or MOSFET statement). You can assign a parameter to DTEMP, then sweep the parameter using the .DC statement. The DTEMP value defaults to zero.

By specifying TREF in the model statement, the model reference temperature is changed (TREF overrides TNOM). The derating of the model parameters is based on the difference of the circuit simulator's temperature and TREF, instead of TNOM.

.TEMP Statement

The syntax is:

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

t1 t2 …	The temperatures, in °C, at which the circuit is simulated

Example

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

The .TEMP statement sets the circuit temperatures for the entire circuit simulation. Star-Hspice uses the temperature set in the .TEMP statement along with the TNOM option setting (or the TREF model parameter) and the DTEMP element temperature, and simulates the circuit with individual elements or model temperatures.

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

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

From the .TEMP statement, the circuit simulation temperature is given as 100°C. Since TNOM is not specified, it defaults to 25°C. The temperature of the diode is given as 30°C above the circuit temperature by the DTEMP parameter. That is, D1temp = 100°C + 30°C = 130°C. The diode, D2, is simulated at 100°C. R1 is simulated at 70°C. Since TREF is specified at 60°C in the diode model statement, the diode model parameters given are derated by 70°C (130°C - 60°C) for diode D1 and by 40°C (100°C - 60°C) for diode D2. The value of R1 is derated by 45°C (70°C - TNOM).

Performing Worst Case Analysis

Worst case analysis often is used for design and analysis of MOS and BJT IC circuits. The worst case is simulated by taking all variables to their 2-sigma or 3-sigma worst case values. Since it is unlikely that several independent variables will attain their worst case values simultaneously, this technique tends to be overly pessimistic, and can lead to over-designing the circuit. However, it is useful as a fast check.

Model Skew Parameters

Avant! has extended the models in Star-Hspice to include physically measurable model parameters. The parameter variations allow the circuit simulator to predict the actual circuit response to the extremes of the manufacturing process. The physically measurable model parameters are called “skew” parameters because they are skewed from a statistical mean to obtain the predicted performance variations.

Examples of skew parameters are the difference between the drawn and physical dimension of metal, polysilicon, or active layers of an integrated circuit.

Generally, skew parameters are chosen independent of each other, so that combinations of skew parameters can be used to represent worst cases. Typical skew parameters for CMOS technology include:

- XL – polysilicon CD (critical dimension of poly layer representing the difference between drawn and actual size)
- XW_n, XW_p – active CD (critical dimension of active layer representing the difference between drawn and actual size)
- TOX – gate oxide thickness
- RSH_n, RSH_p – active layer resistivity
- $DELVTO_n$, $DELVTO_p$– threshold voltage variation

These parameters are allowed in any level MOS model in Star-Hspice. The DELVTO parameter simply shifts the threshold value. It is added to VTO for the Level 3 model and is added to or subtracted from VFB0 for the BSIM model.

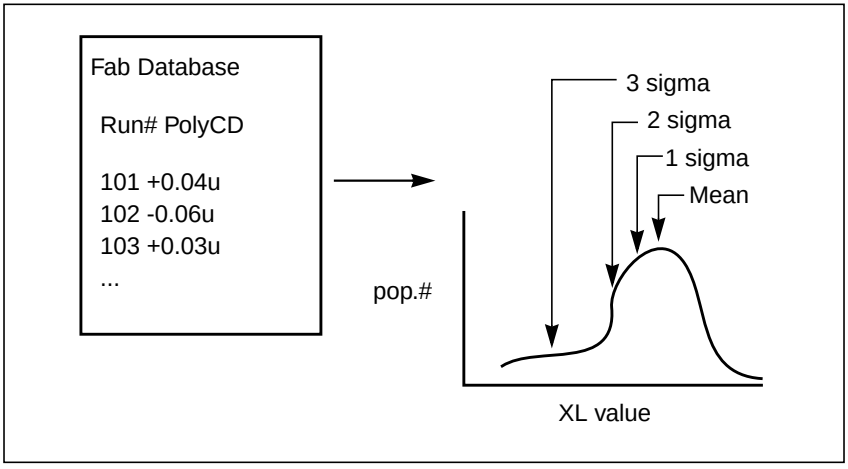
Table 13-1 shows whether deviations are added to or subtracted from the average.

Table 13-1: Sigma Deviations

Type	Param	Slow	Fast
NMOS	XL	+	-
	RSH	+	-
	DELVTO	+	-
	TOX	+	-
	XW	-	+
PMOS	XL	+	-
	RSH	+	-
	DELVTO	-	+
	TOX	+	-
	XW	-	+

Skew parameters are chosen based on the available historical data collected either during fabrication or electrical test. For example, the poly CD skew parameter XL is collected during fabrication. This parameter is usually the most important skew parameter for a MOS process. Historical records produce data as shown in Figure 13-2.

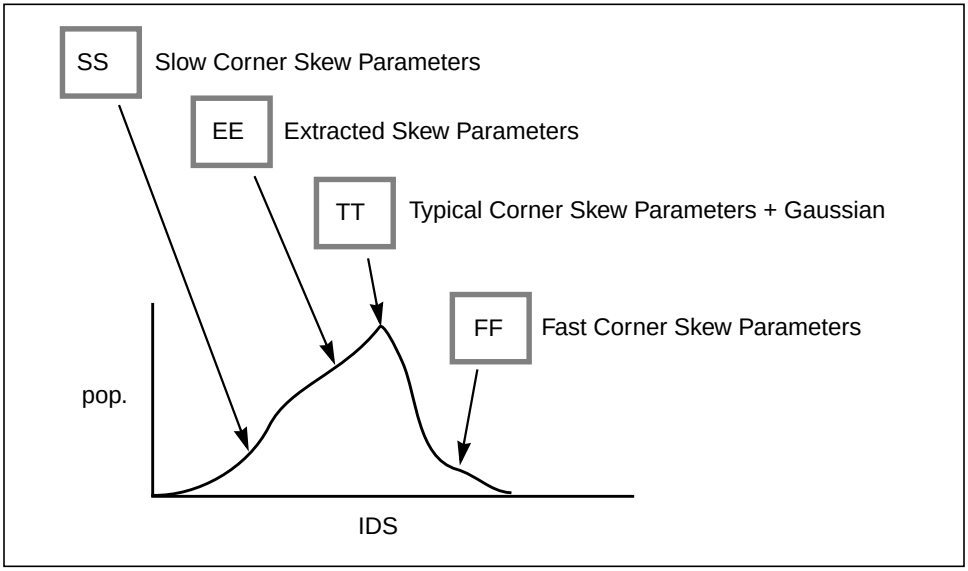
Figure 13-2: Historical Records for Skew Parameters in a MOS Process



Using Skew Parameters in Star-Hspice

The following example shows how to create a worst case corners library file for a CMOS process model. The physically measured parameter variations must be chosen so that their proper minimum and maximum values are consistent with measured current (IDS) variations. For example, a 3-sigma variation in IDS can be generated from a 2-sigma variation in the physically measured parameters.

Figure 13-3: Worst Case Corners Library File for a CMOS Process Model



The simulator accesses the models and skew through the .LIB library statement and the .INCLUDE include file statement. The library contains parameters that modify .MODEL statements. The following example of .LIB of model skew parameters features both worst case and statistical distribution data. The statistical distribution median value is the default for all non-Monte Carlo analysis.

Example

```
.LIB TT
$TYPICAL P-CHANNEL AND N-CHANNEL CMOS LIBRARY DATE:3/4/91
$ PROCESS: 1.0U CMOS, FAB22, STATISTICS COLLECTED 3/90-2/91
$ following distributions are 3 sigma ABSOLUTE GAUSSIAN

.PARAM
$ polysilicon Critical Dimensions
+ polycd=agauss(0,0.06u,1) xl='polycd-sigma*0.06u'
$ Active layer Critical Dimensions
+ nactcd=agauss(0,0.3u,1) xwn='nactcd+sigma*0.3u'
+ pactcd=agauss(0,0.3u,1) xwp='pactcd+sigma*0.3u'
$ Gate Oxide Critical Dimensions (200 angstrom +/- 10a at 1
$ sigma)
+ toxcd=agauss(200,10,1) tox='toxcd-sigma*10'
```

```
$ Threshold voltage variation
+ vtoncd=agauss(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=agauss(0,0.05v,1) delvtop='vtopcd+sigma*0.05'

.INC '/usr/meta/lib/cmos1_mod.dat'$ model include file

.ENDL TT
.LIB FF
$HIGH GAIN P-CH AND N-CH CMOS LIBRARY 3SIGMA VALUES

.PARAM TOX=230 XL=-0.18u DELVTON=-.15V DELVTOP= 0.15V
.INC '/usr/meta/lib/cmos1_mod.dat'$ model include file

.ENDL FF
```

The */usr/meta/lib/cmos1_mod.dat* include file contains the model.

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

Note: The model keyname (left side) equates to the skew parameter (right side). Model keynames and skew parameters can have the same names.

Skew File Interface to Device Models

The skew parameters are model parameters. They are used most often for transistor models, but they also apply to passive components. A typical device model set includes:

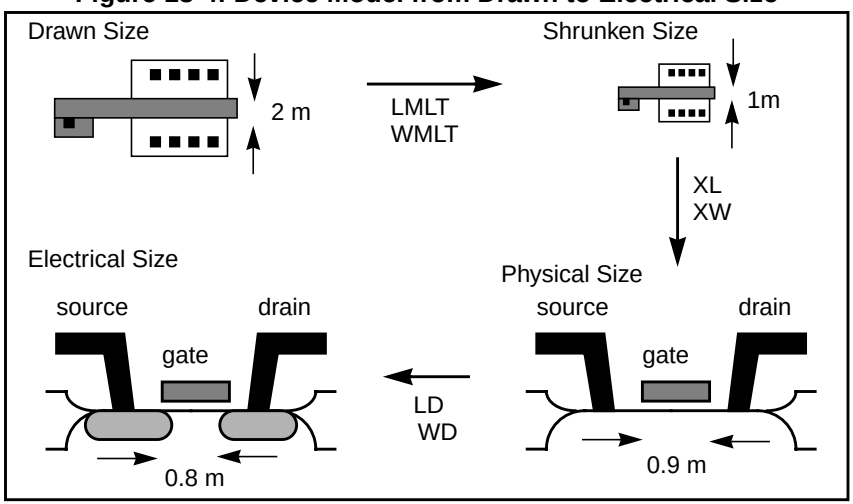
- MOSFET models for all device sizes using automatic model selector
- RC wire models for polysilicon, metal1, and metal2 layers (These models include temperature coefficients and fringing capacitance. They apply to drawn dimension)
- Single and distributed diode models for N+,P+, and well (includes temperature, leakage, and capacitance based on drawn dimension)
- BJT models for parasitic bipolar transistors, as well as 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 be able to accept elements with sizes based on drawn dimension. A cell might be drawn at 2 µ dimension, shrink to 1 µ, with a physical size of 0.9 µ and an effective electrical size of 0.8 µ. The following dimension levels must be accounted for:

  drawn size
  shrunken size
  physical size
  electrical size

Note: Most simulator models go directly from drawn size to the electrical size. Star-Hspice is designed to support all four size levels for MOS models. Figure 13-4 shows the importance of the four size levels.

Figure 13-4: Device Model from Drawn to Electrical Size



Performing 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 average
- Absolute variation — variation is added to average
- Bimodal – nominal parameters are statistically reduced by multiplication of distribution

Uniform Parameter Distribution

- Relative variation — variation is a ratio of average
- Absolute variation — variation is added to average
- Bimodal – nominal parameters are statistically reduced by multiplication of distribution

Random Limit Parameter Distribution

- Absolute variation — variation is added to average
- Min or max variation is randomly selected

The number of times the operating point, DC sweep, AC sweep, or transient analysis is performed is determined by the value of the analysis keyword MONTE.

Monte Carlo Setup

To set up a Monte Carlo analysis, use the following Star-Hspice statements:

- .PARAM statement – sets a model or element parameter to a Gaussian, Uniform, or Limit function distribution.
- .DC, .AC, or .TRAN analysis – enable MONTE.
- .MEASURE statement – calculates output mean, variance, sigma, and standard deviation.

Analysis Syntax

Select the type of analysis desired, such as operating point, DC sweep, AC sweep, or TRAN sweep.

Operating point:

```
.DC MONTE=val
```

DC sweep:

```
.DC vin 1 5 .25 SWEEP MONTE=val
```

AC sweep:

```
.AC dec 10 100 10meg SWEEP MONTE=val
```

TRAN sweep:

```
.TRAN 1n 10n SWEEP MONTE=val
```

The value “val” represents the number of Monte Carlo iterations to be performed. A reasonable number is 30. The statistical significance of 30 iterations is quite high. If the circuit operates correctly for all 30 iterations, there is a 99% probability that over 80% of all possible component values operate correctly. The relative error of a quantity determined through Monte Carlo analysis is proportional to $val^{-1/2}$.

Monte Carlo Output

Use .MEASURE statements as the most convenient way to summarize the results.

The .PRINT statement generates tabular results and prints all Monte Carlo parameter usage values. If one iteration is out of specification, obtain the component values from the tabular listing. A detailed resimulation of that iteration might help identify the problem.

.GRAPH generates a high-resolution plot for each iteration. In contrast, AvanWaves superimposes all iterations as a single plot and allows you to analyze each iteration individually.

.PARAM Distribution Function Syntax

A .PARAM parameter is assigned to the keywords of Star-Hspice elements and models. A distribution function is assigned to each .PARAM parameter. The distribution function is recalculated for each element or model keyword use of a parameter. This feature allows a parameterized schematic netlist to be used for Monte Carlo analysis with no additional modifications.

The syntax is:

```
.PARAM xx=UNIF(nominal_val, rel_variation
+ <, multiplier>)
```

or

```
.PARAM xx=AUNIF(nominal_val, abs_variation <, multiplier>)
```

or

```
.PARAM xx=GAUSS(nominal_val, rel_variation, sigma <,
+ multiplier>)
```

or

```
.PARAM xx=AGAUSS(nominal_val, abs_variation, sigma <,
+ multiplier>)
```

or

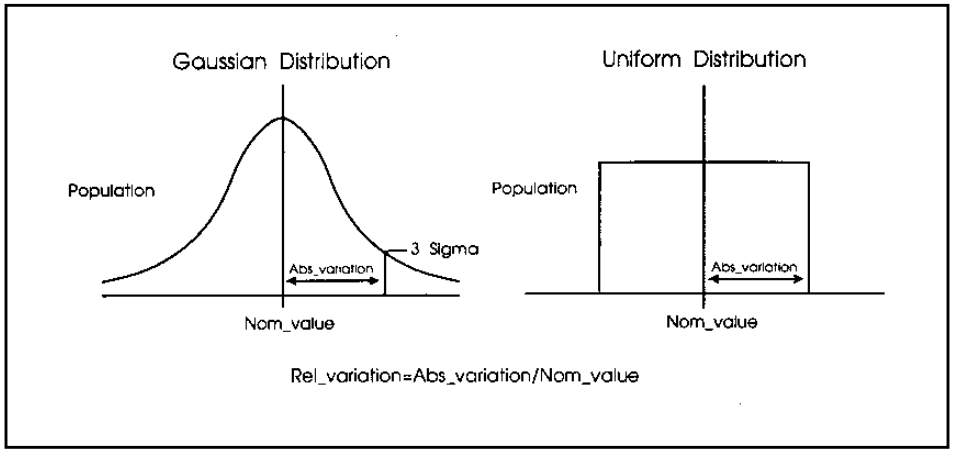
```
.PARAM xx=LIMIT(nominal_val, abs_variation)
```

where:

xx	The parameter whose value is calculated by distribution function
UNIF	Uniform distribution function using relative variation
AUNIF	Uniform distribution function using absolute variation
GAUSS	Gaussian distribution function using relative variation

AGAUSS	Gaussian distribution function using absolute variation
LIMIT	Random limit distribution function using absolute variation, +/- abs_variation is added to nominal_val based on whether the random outcome of a -1 to 1 distribution is greater or less than 0.
nominal_val	Nominal value for Monte Carlo analysis and default value for all other analyses
abs_variation	The AUNIF and AGAUSS vary the nominal_val by +/- abs_variation
rel_variation	The UNIF and GAUSS vary the nominal_val by +/- (nominal_val · rel_variation)
sigma	The abs_variation or rel_variation is specified at the sigma level. For example, if sigma=3, then the standard deviation is abs_variation divided by 3.
multiplier	If not specified, the default is 1. The calculation is repeated this many times and the largest deviation is saved. The resulting parameter value might be greater or less than nominal_val. The resulting distribution is bimodal.

Figure 13-5: Monte Carlo Distribution



Monte Carlo Parameter Distribution Summary

A new random variable is calculated each time a parameter is used. If no Monte Carlo distribution is specified, then the nominal value is assumed. When a Monte Carlo distribution is specified for only one analysis, the nominal value is used for all other analyses.

You can assign a Monte Carlo distribution to all elements that share a common model. The actual element value will vary by the element distribution. A Monte Carlo distribution also can be assigned to a model keyword, and all elements that share that model use the same keyword value. This allows double element and model distributions to be created.

For example, the MOSFET channel length varies from transistor to transistor by a small amount corresponding 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 given die site will vary by the wafer or fabrication run distribution, which is much larger than the die distribution, but affects all transistors the same. The wafer distribution is assigned to the MOSFET model; it sets the speed and power dissipation characteristics.

Monte Carlo Examples

Gaussian, Uniform, and Limit Functions

```
Test of monte carlo gaussian, uniform, and limit functions
.options post
.dc monte=60
* setup plots
.model histo plot ymin=80 ymax=120 freq=1

.graph model=HISTO aunif_1=v(au1)
.graph model=HISTO aunif_10=v(au10)
.graph model=HISTO agauss_1=v(ag1)
.graph model=HISTO agauss_10=v(ag10)
.graph model=HISTO limit=v(L1)

* uniform distribution relative variation +/- .2
.param ru_1=unif(100,.2)

Iu1 u1 0 -1
ru1 u1 0 ru_1

* absolute uniform distribution absolute variation +/- 20
* single throw and 10 throw maximum
.param rau_1=aunif(100,20)
.param rau_10=aunif(100,20,10)
Iau1 au1 0 -1
rau1 au1 0 rau_1

Iau10 au10 0 -1
rau10 au10 0 rau_10

* gaussian distribution relative variation +/- .2
* at 3 sigma
.param rg_1=gauss(100,.2,3)

Ig1 g1 0 -1
rg1 g1 0 rg_1

* absolute gaussian distribution absolute variation +/- .2
* at 3 sigma
* single throw and 10 throw maximum
.param rag_1=agauss(100,20,3)
.param rag_10=agauss(100,20,3,10)

Iag1 ag1 0 -1
rag1 ag1 0 rag_1
```

```
Iag10 ag10 0 -1
rag10 ag10 0 rag_10

* random limit distribution absolute variation +/- 20
.param RL=limit(100,20)

IL1 L1 0 -1
rL1 L1 0 RL
.end
```

Figure 13-6: Gaussian Functions

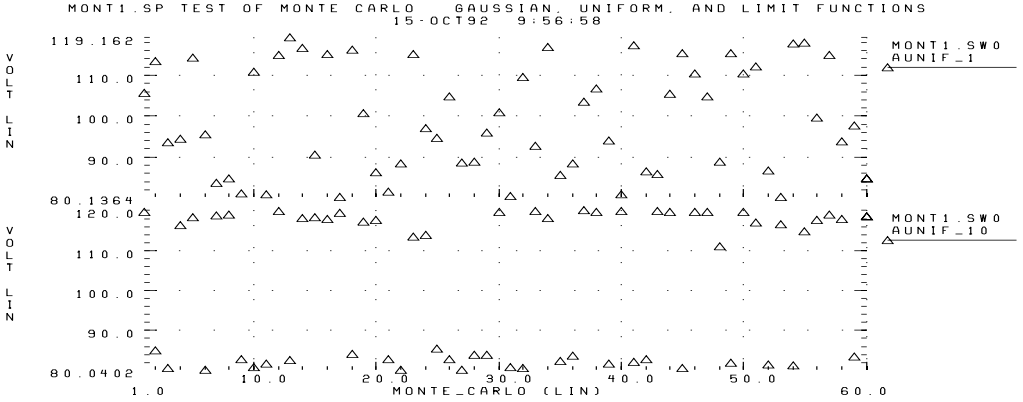


Figure 13-7: Uniform Functions

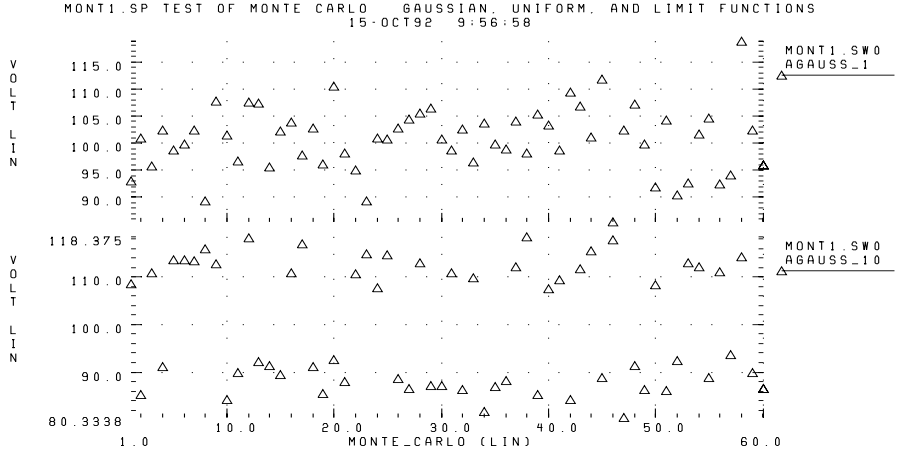
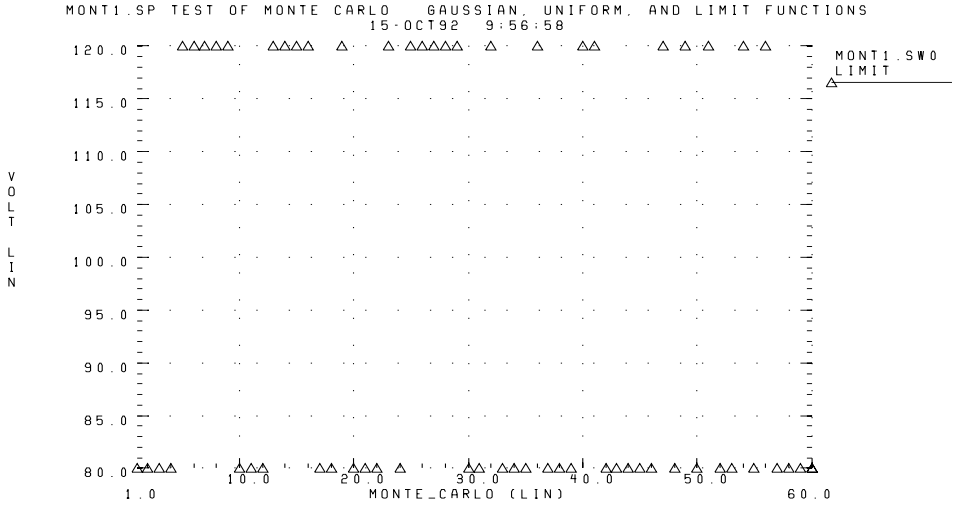


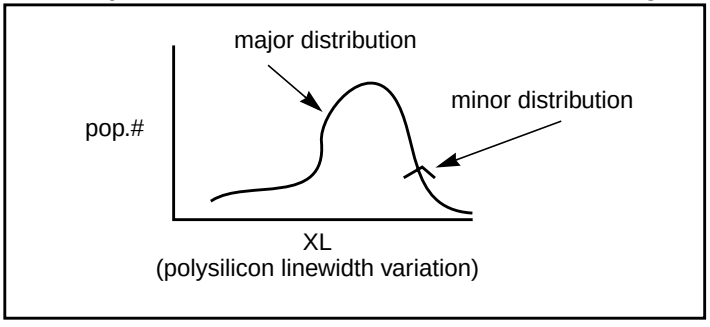
Figure 13-8: Limit Functions



Major and Minor Distribution

MOS IC processes have both a major and a minor statistical distribution of manufacturing tolerance parameters. The major distribution is the wafer-to-wafer and run-to-run variation. The minor distribution is the transistor-to-transistor process variation. The major distribution determines electrical yield. The minor distribution is responsible for critical second-order effects, such as amplifier offset voltage and flip-flop preference.

Figure 13-9: Major and Minor Distribution of Manufacturing Variations



The example below is a Monte Carlo analysis of a DC sweep of the supply voltage VDD from 4.5 volts to 5.5 volts. Transistors M1 through M4 form two inverters. The channel lengths for the MOSFETs are set by the nominal value of the parameter LENGTH, which is set to 1u. Since all of the transistors are on the same integrated circuit die, the distribution is given by the parameter LEFF, which is a ±5% distribution in the variation of the channel lengths at the ±3-sigma level. Each MOSFET gets an independent random Gaussian value.

The parameter PHOTO controls the difference between the physical gate length and drawn gate length. Because both n-channel and p-channel transistors use the same layer for the gates, the Monte Carlo distribution XPHOTO is set to the local parameter PHOTO.

PHOTO lithography for both NMOS and PMOS devices is controlled by XPHOTO, which is consistent with the physics of manufacturing.

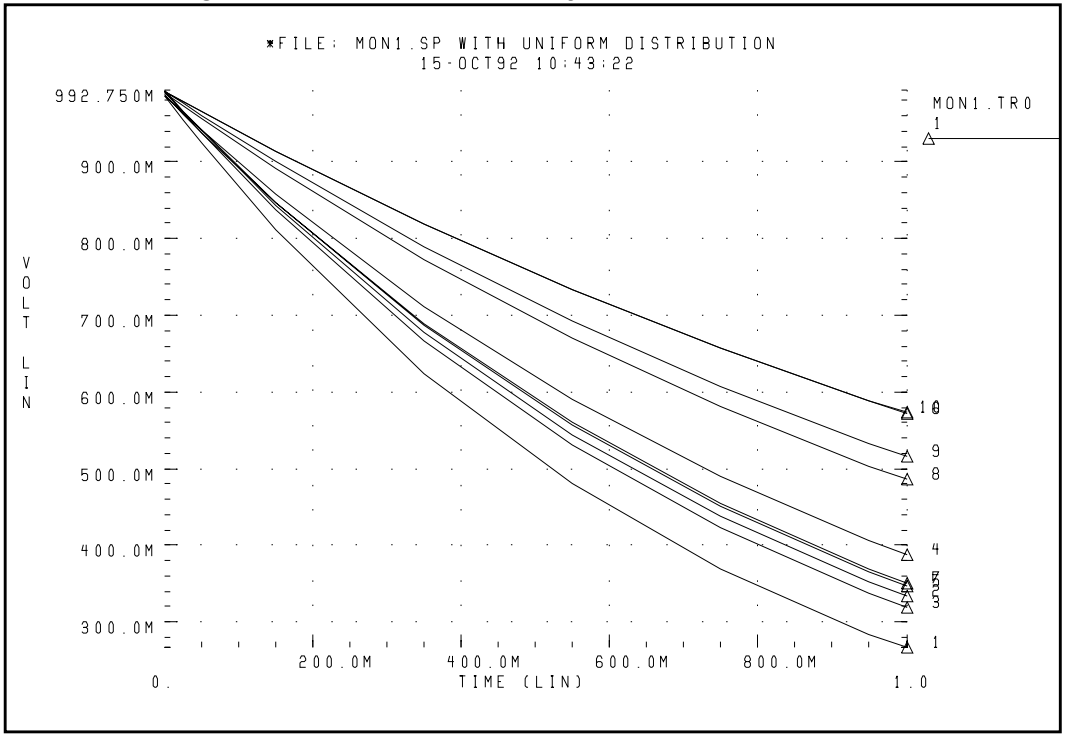
```
File: MONDC_A.SP
.DC VDD 4.5 5.5  .1   SWEEP MONTE=30
.PARAM LENGTH=1U LPHOTO=.1U
.PARAM LEFF=GAUSS (LENGTH, .05, 3)
+ XPHOTO=GAUSS (LPHOTO, .3, 3)
.PARAM PHOTO=XPHOTO

M1 1 2 GND GND NCH W=10U L=LEFF
M2 1 2 VDD‘ VDD PCH W=20U L=LEFF
M3 2 3 GND GND NCH W=10U L=LEFF
M4 2 3 VDD VDD PCH W=20U L=LEFF

.MODEL NCH NMOS LEVEL=2 UO=500 TOX=100 GAMMA=.7 VTO=.8
+ XL=PHOTO

.MODEL PCH PMOS LEVEL=2 UO=250 TOX=100 GAMMA=.5 VTO=-.8
+ XL=PHOTO
.INC Model.dat
.END
```

RC Time Constant

This simple example demonstrates the uniform distribution for resistance and capacitance and the resulting transient waveforms for 10 different random values.

```
*FILE: MON1.SP WITH UNIFORM DISTRIBUTION
.OPTION LIST POST=2
.PARAM RX=UNIF(1,  .5) CX=UNIF(1,  .5)
.TRAN  .1 1 SWEEP MONTE=10
.IC 1 1
R1 1 0   RX
C1 1 0   CX
.END
```

Figure 13-10: Monte Carlo Analysis of RC Time Constant



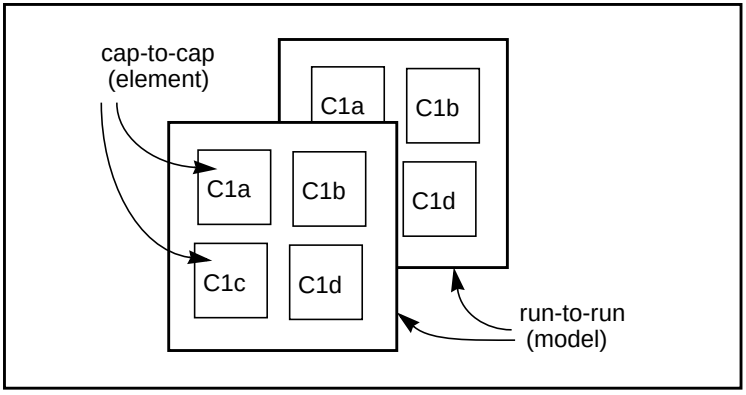
Switched Capacitor Filter Design

The capacitors used in switched capacitor filter applications are composed of parallel connections of a basic cell. Use Monte Carlo techniques to estimate the variation in total capacitance. There are two distributions involved in the capacitance calculation:

- Minor distribution of cell capacitance from cell-to-cell on a single die
- Major distribution of the capacitance from wafer-to-wafer or manufacturing run-to-run

The minor distribution is the element distribution, and the major distribution is the model distribution.

Figure 13-11: Monte Carlo Distribution



You can approach this problem from either physical or electrical levels. The physical level relies on physical distributions such as oxide thickness and polysilicon linewidth control. The electrical level relies on actual capacitor measurements.

Physical Approach

Assume that the variation in capacitance for adjacent cells is controlled by the local variation of polysilicon, since the oxide thickness control is excellent for small areas on a single wafer.

Next, define a local poly linewidth variation and a global or model level poly linewidth variation.

The local polysilicon linewidth control for a line 10 µ wide, manufactured with process A, is assumed to be ±0.02 µ for a 1-sigma distribution. The global or model level polysilicon linewidth control is much wider; use 0.1 µ for this example. The global oxide thickness is assumed to be 200 angstroms with a ±5 angstrom variation at 1 sigma.

The cap element is assumed to be square, with local poly variation in both directions. The cap model has two distributions, the poly linewidth distribution and the oxide thickness distribution. Because the effective length is

$$Leff = Ldrawn - 2 \cdot DEL$$

the model poly distribution is half the physical per-side values.

```
C1a 1 0 CMOD W=ELPOLY L=ELPOLY
C1b 1 0 CMOD W=ELPOLY L=ELPOLY
C1C 1 0 CMOD W=ELPOLY L=ELPOLY
C1D 1 0 CMOD W=ELPOLY L=ELPOLY
$ 10U POLYWIDTH,0.05U=1SIGMA
$ CAP MODEL USES 2*MODPOLY  .05u= 1 sigma
$ 5angstrom oxide thickness AT 1SIGMA
.PARAM ELPOLY=AGAUSS(10U,0.02U,1)
+ MODPOLY=AGAUSS(0,.05U,1)
+ POLYCAP=AGAUSS(200e-10,5e-10,1)
.MODEL CMOD C THICK=POLYCAP DEL=MODPOLY
```

Electrical Approach

The electrical approach assumes no physical interpretation, but requires a local or element distribution and a global or model distribution.

Assume that the capacitors can be matched to ±1% for the 2-sigma population. The process can maintain a ±10% variation from run to run for a 2-sigma distribution.

```
C1a 1 0 CMOD SCALE=ELCAP
C1b 1 0 CMOD SCALE=ELCAP
C1C 1 0 CMOD SCALE=ELCAP
C1D 1 0 CMOD SCALE=ELCAP
.PARAM ELCAP=Gauss(1,.01,2) $ 1% at 2 sigma
+ MODCAP=Gauss(.25p,.1,2) $10% at 2 sigma
.MODEL CMOD C CAP=MODCAP
```

Worst Case and Monte Carlo Sweep Example

The following example measures the delay of a pair of inverters. The input is buffered by an inverter, and the output is loaded by another inverter. The model was prepared according to the scheme described in the previous sections. The first .TRAN analysis statement sweeps from the worst case 3- sigma slow to 3-sigma fast. The second .TRAN does 100 Monte Carlo sweeps.

HSPICE Input File

The Star-Hspice input file can contain the following sections.

Analysis Setup Section

The simulation is accelerated by the use of the AUTOSTOP option, which automatically stops the simulation when the .MEASURE statements have achieved their target values.

```
$ inv.sp sweep mosfet -3 sigma to +3 sigma,
$ then Monte Carlo
.option nopage nomod acct
+ autostop post=2
.tran 20p 1.0n sweep sigma -3 3 .5
.tran 20p 1.0n sweep monte=20
.option post co=132
.param vref=2.5
.meas m_delay trig v(2) val=vref fall=1
+ targ v(out) val=vref fall=1
.meas m_power rms power to=m_delay
.param sigma=0
```

Circuit Netlist Section

```
.global 1
vcc 1 0 5.0
vin in 0 pwl 0,0 0.2n,5
x1 in 2 inv
x2 2 3 inv
x3 3 out inv
x4 out 5 inv
.macro inv in out
     mn out in 0 0 nch W=10u L=1u
     mp out in 1 1 pch W=10u L=1u
.eom
```

Skew Parameter Overlay for Model Section

```
* overlay of gaussian and algebraic for best case worst case
+ and + monte carlo
* +/- 3 sigma is the maximum value for parameter sweep

.param
+ mult1=1
+ polycd=agauss(0,0.06u,1) xl='polycd-sigma*0.06u'
+ nactcd=agauss(0,0.3u,1) xwn='nactcd+sigma*0.3u'
+ pactcd=agauss(0,0.3u,1) xwp='pactcd+sigma*0.3u'
+ toxcd=agauss(200,10,1) tox='toxcd-sigma*10'
+ vtoncd=agauss(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=agauss(0,0.05v,1) delvtop='vtopcd+sigma*0.05'
+ rshncd=agauss(50,8,1) rshn='rshncd-sigma*8'
+ rshpcd=agauss(150,20,1) rshp='rshpcd-sigma*20'
```

MOS Model for N-Channel and P-Channel Transistors Section

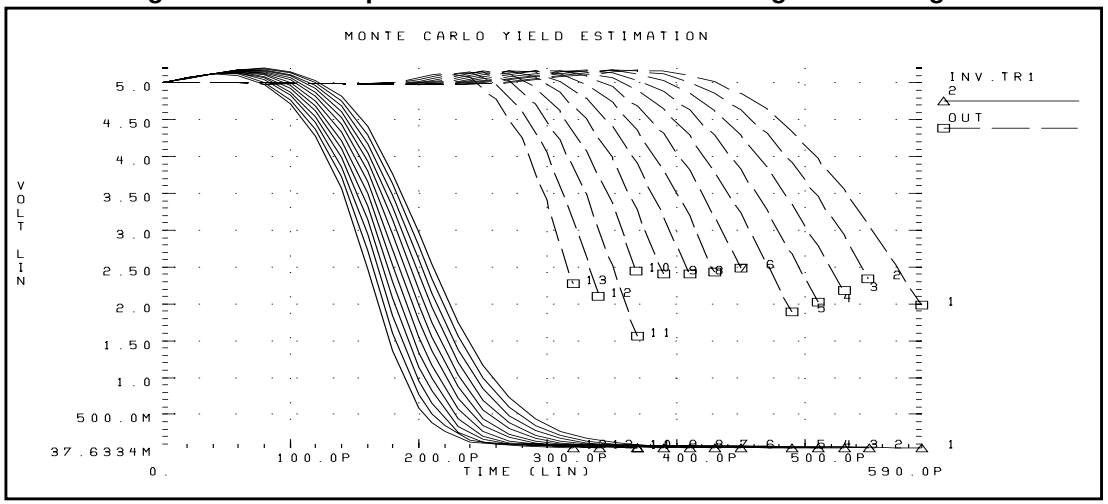
```
* LEVEL=28 example model for high accuracy model
.model nch nmos
+ LEVEL=28
+ lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl xw=xwn tox=tox delvto=delvton rsh=rshn
+ ld=0.06u wd=0.2u
+ acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0
+ js=3e-04 jsw=9e-10
+ cj=3e-04 mj=.5 pb=.8 cjsw=3e-10 mjsw=.3 php=.8 fc=.5
+ capop=4 xqc=.4 meto=0.08u
+ tlev=1 cta=0 ctp=0 tlevc=0 nlev=0
+ trs=1.6e-03 bex=-1.5 tcv=1.4e-03
* dc model
```

```
+ x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x2m=0
+ vfb0=-.5 phi0=0.65 k1=.9 k2=.1 eta0=0
+ muz=500 u00=.075
+ x3ms=15 u1=.02 x3u1=0
+ b1=.28 b2=.22 x33m=0.000000e+00
+ alpha=1.5 vcr=20
+ n0=1.6 wfac=15 wfacu=0.25
+ lvfb=0 lk1=.025 lk2=.05
+ lalpha=5
.model pch pmos
+ LEVEL=28
+ lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl xw=xwp tox=tox delvto=delvtop rsh=rshp
+ ld=0.08u wd=0.2u
+ acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0 rsh=rshp
+ js=3e-04 jsw=9e-10
+ cj=3e-04 mj=.5 pb=.8 cjsw=3e-10 mjsw=.3 php=.8 fc=.5
+ capop=4 xqc=.4 meto=0.08u
+ tlev=1 cta=0 ctp=0 tlevc=0 nlev=0
+ trs=1.6e-03 bex=-1.5 tcv=-1.7e-03
* dc model
+ x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x2m=5
+ vfb0=-.1 phi0=0.65 k1=.35 k2=0 eta0=0
+ muz=200 u00=.175
+ x3ms=8 u1=0 x3u1=0.0
+ b1=.25 b2=.25 x33m=0.0
+ alpha=0 vcr=20
+ n0=1.3 wfac=12.5 wfacu=.2
+ lvfb=0 lk1=-.05
.end
```

Transient Sigma Sweep Results

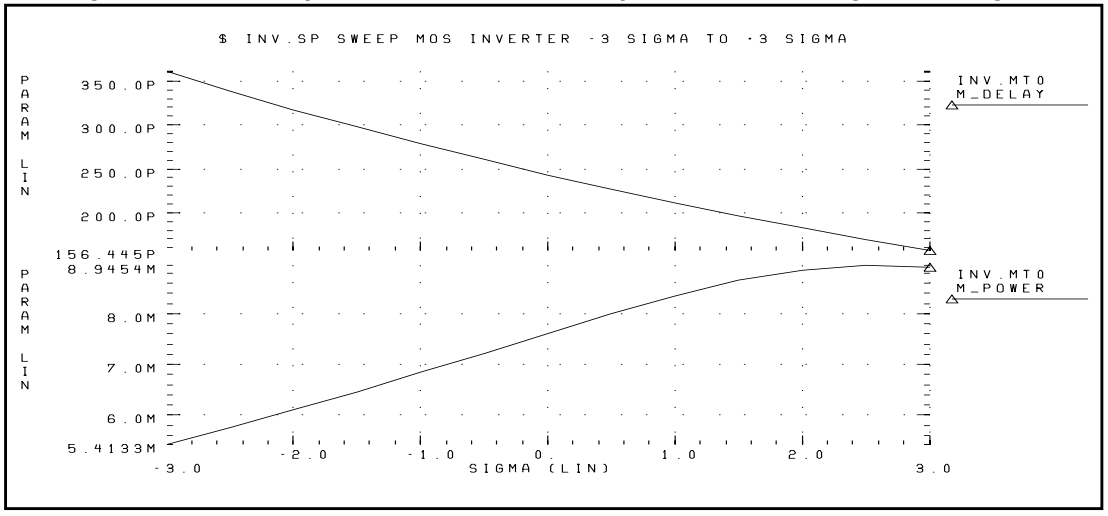
The plot in Figure 13-12 shows the family of transient analysis curves from the transient sweep of the sigma parameter from -3 to +3. Sigma is then algebraically coupled into the skew parameters and the resulting parameters modify the actual NMOS and PMOS models.

Figure 13-12: Sweep of Skew Parameters from -3 Sigma to +3 Sigma



You can view the transient family of curves by plotting the .MEASURE output file. The plot in Figure 13-13 shows the measured pair delay and the total dissipative power against the parameter SIGMA.

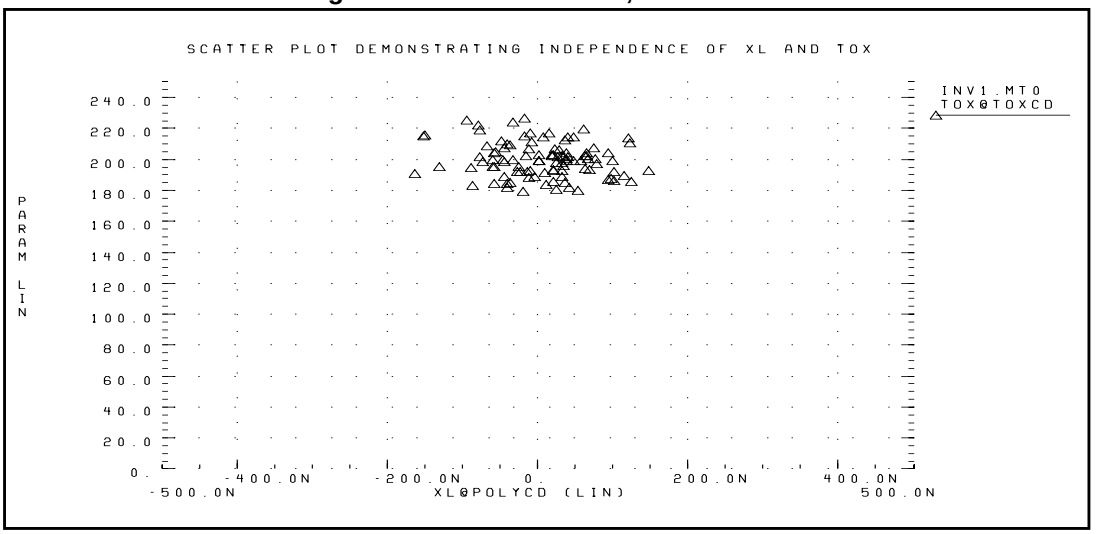
Figure 13-13: Sweep MOS Inverter, Pair Delay and Power: -3 Sigma to 3 Sigma



Monte Carlo Results

The output of the Monte Carlo analysis is evaluated in this section. The plot in Figure 13-14 is a quality control step that plots TOX against XL (polysilicon critical dimension). The cloud of points was obtained in Avant!'s graphing software by setting XL as the X-axis independent variable and plotting TOX with a symbol frequency of 1. This has the effect of showing the points without any connecting lines. The resulting graph demonstrates that the TOX model parameter is randomly independent of XL.

Figure 13-14: Scatter Plot, XL and TOX

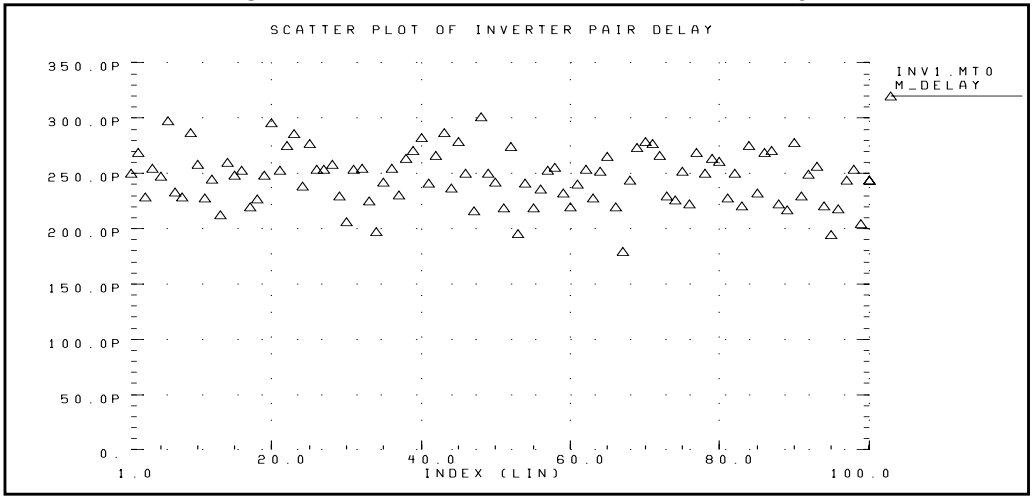


The next graph (see Figure 13-15) is a standard scatter plot of the measured inverter pair delay against the Monte Carlo index number. If a particular result looks interesting, such as if the very smallest delay was obtained in simulation 68 ("monte carlo index = 68"), you can read the output listing file and obtain the actual Monte Carlo parameters associated with that simulation.

```
*** monte carlo index = 68 ***
MONTE CARLO PARAMETER DEFINITIONS
polycd: xl = -1.6245E-07
nactcd: xwn = 3.4997E-08
pactcd: xwp = 3.6255E-08
toxcd: tox = 191.0
vtoncd: delvton = -2.2821E-02
vtopcd: delvtop = 4.1776E-02
rshncd: rshn = 45.16
rshpcd: rshp = 166.2

m_delay = 1.7946E-10 targ= 3.4746E-10 trig= 1.6799E-10
m_power = 7.7781E-03 from= 0.0000E+00 to= 1.7946E-10
```

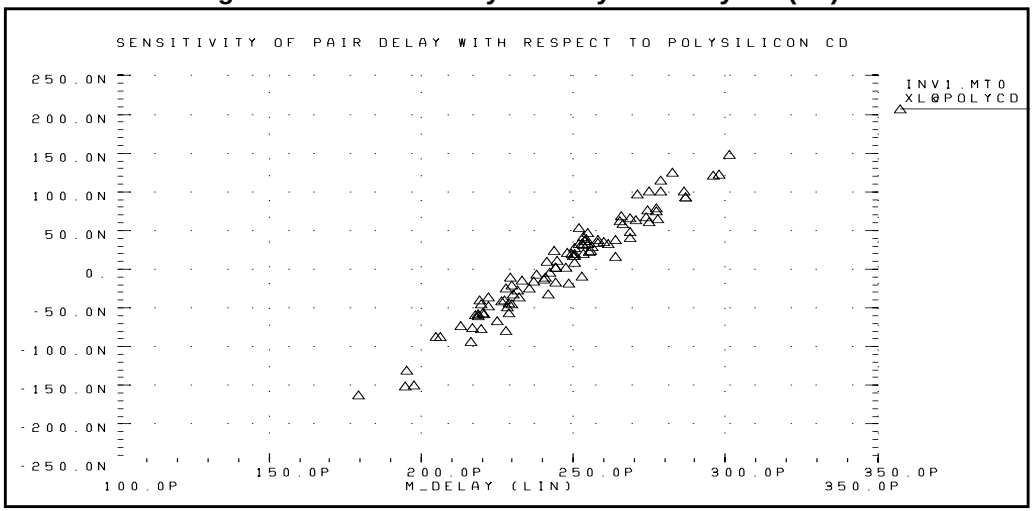
From the preceding listing, you can see that the *m_delay* value of 1.79e-10 seconds is the fastest pair delay. In addition, the Monte Carlo parameters can be examined.

Figure 13-15: Scatter Plot of Inverter Pair Delay



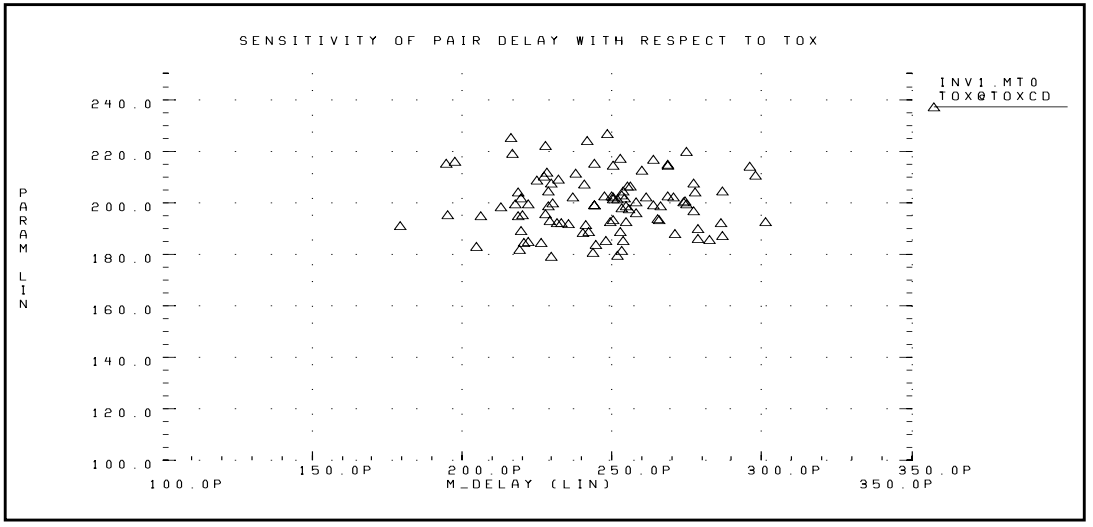
Plotting against the Monte Carlo index number does not help in centering the design. The first step in centering a design is to determine the most sensitive process variables. We can do this by graphing the various process parameters against the pair delay. Select the pair delay as the X-axis independent variable, and also set the symbol frequency to 1 to obtain the scatter plot. The graph in Figure 13-16 demonstrates the expected sensitivity of output pair delay to channel length variation (polysilicon variation).

Figure 13-16: Sensitivity of Delay with Poly CD (XL)



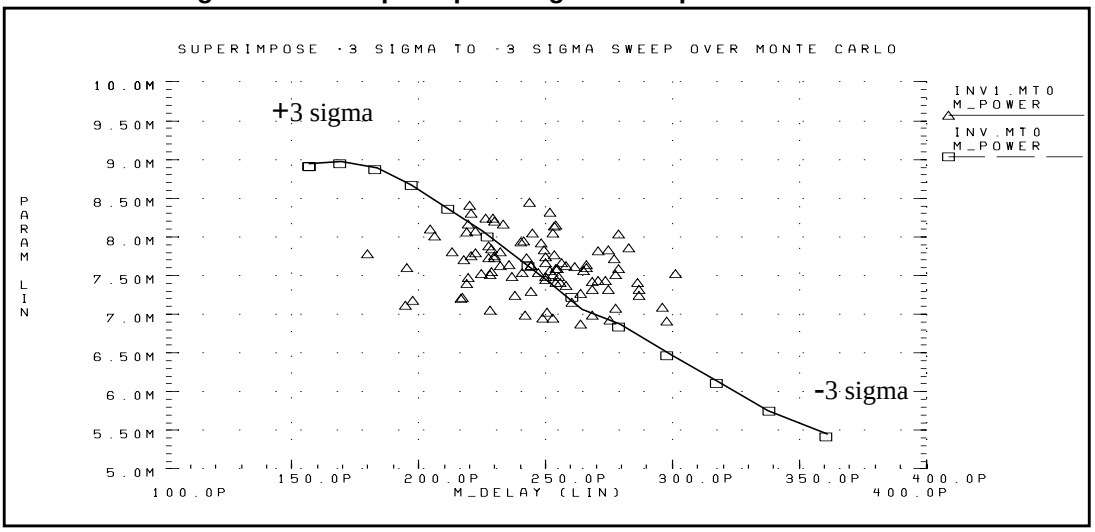
Now, the parameter TOX is plotted against pair delay (Figure 13-17). Note that there is no clear tilt to the scatter plot. This indicates that TOX is a secondary process parameter compared to XL. To explore this in more detail, set the skew parameter XL to a constant and simulate.

Figure 13-17: Sensitivity of Delay with TOX



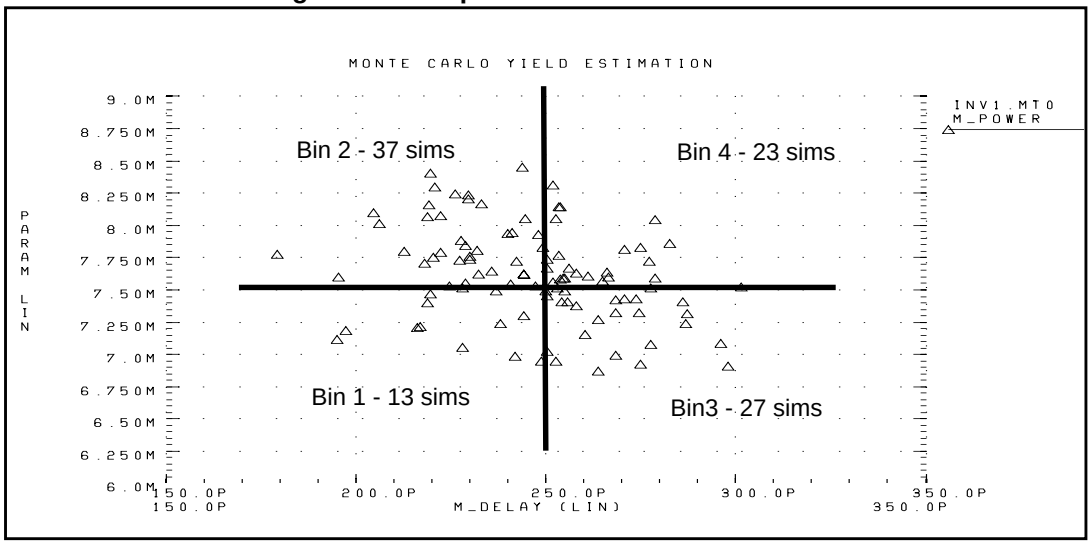
The plot in Figure 13-18 shows the overlay of a 3-sigma worst case corners response and the 100 point Monte Carlo. Notice that the actual (Monte Carlo) distribution for power/delay is very different than the +3 sigma to -3 sigma plot. The worst case was simulated in 0.5 sigma steps. The actual response is closer to ± 1.5 sigma instead of ± 3 sigma. This produces a predicted delay variation of 100 ps instead of 200 ps. Therefore, the advantage of using Monte Carlo over traditional 3-sigma worst case corners is a 100% improvement in accuracy of simulated-to-actual distribution. This is an example of how the worst-case procedure is overly pessimistic.

Figure 13-18: Superimpose Sigma Sweep over Monte Carlo



Now take the Monte Carlo plot and superimpose the assumed part grades from marketing studies (Figure 13-19). This example uses a 250 ps delay and 7.5 mW power dissipation to determine the 4 binning grades. A manual count gives: Bin1 - 13%, Bin2 - 37%, Bin3 - 27%, Bin4 - 23%. If this circuit were representative of the entire chip, we would predict a present yield of 13% for the premium Bin 1 parts, assuming the design and process variations.

Figure 13-19: Speed/Power Yield Estimation



Optimization

Optimization, the automatic generation of model parameters and component values from a given set of electrical specifications or measured data, is available in Star-Hspice. With a user-defined optimization program and a known circuit topology, Star-Hspice automatically selects the design components and model parameters to meet DC, AC, and transient electrical specifications.

Star-Hspice optimization is the result of more than ten years of research in both the optimizing algorithms and user interface. The optimizing function is integrated into the core of Star-Hspice, resulting in optimum efficiency. The circuit result targets are part of the .MEASURE command structure, and the parameters to be optimized are Star-Hspice-defined parameter functions. A .MODEL statement sets up the optimization.

Note: Star-Hspice computes the .MEASURE statements using the postprocessing output. Reducing postprocessing output by setting option INTERP=1 may lead to interpolation errors in measurement results. See Input and Output on page 9-50 for more information about using these options.

The most powerful feature of the Star-Hspice approach is its incremental optimization technique. Incremental optimization allows you to solve the DC parameters first, then the AC parameters, and finally the transient parameters. A set of optimizer measurement functions not only makes the task of transistor optimization easy, but significantly improves cell and whole circuit optimization.

To perform optimization, create an input netlist file specifying:

- Minimum and maximum parameter and component limits
- The variable parameters and components
- An initial estimate of the selected parameter and component values
- The circuit performance goals or model-versus-data error function

Given the input netlist file, optimization specifications, component limits, and initial guess, the optimizer reiterates the circuit simulation until the target electrical specification is met or an optimized solution is found.

For improved optimization and simulation time and to increase the likelihood of a convergent solution, the initial estimate of the component values should produce a circuit with specifications near those of the original target. This reduces the number of times the optimizer reselects component values and resimulates the circuit.

Optimization Control

The length of time to complete an optimization is a function of the number of iterations allowed, the relative input tolerance, the output tolerance, and the gradient tolerance. The default values are satisfactory for most applications. Generally, 10 to 30 iterations are sufficient to get accurate optimizations.

Simulation Accuracy

Set the simulator with tighter convergence options than normal for optimization. The following options are suggested:

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 user-defined DC, AC, or transient data. Use the .DATA statement to store the desired numeric data for curves in the data file, as in-line data. The .PARAM xxx=OPTxxx statement specifies the variable circuit components and parameter values of the netlist. The optimization analysis statements call the in-line data, using the DATA= keyword. The .MEASURE statement compares the simulation result to the values in the data file, using the ERR1 keyword to control the comparison. If the calculated value is not within the error tolerances specified in the optimization model, Star-Hspice selects a new set of component values, and simulates the circuit again. This is repeated until the closest fit to the curve is obtained, or the error tolerances set is satisfied.

Goal Optimization

Goal optimization differs from curve fit optimization in that it usually only applies to the optimization of a particular electrical specification, such as rise time or power dissipation.

Goal optimizations are specified using the GOAL keyword with a choice of relational operator in the .MEASURE statement, where GOAL is the target electrical specification being measured. This choice of relational operator is useful in multiple-constraint optimizations, when the absolute accuracy of some criteria is less important than for others.

Performing Timing Analysis

To analyze circuit timing violation, Star-Hspice uses a binary search algorithm 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 that can lead to a design guideline. Typical types of timing constraint violations include:

- Data setup time before clock
- Data hold time after clock
- Minimum pulse width required to allow a signal to propagate to the output
- Maximum toggle frequency of the component(s)

Bisection is a method of optimization that finds the value of an input variable (target value) associated with a goal value of an output variable. The input and output variables can be of various types (for example, voltage, current, delay time, or gain) related by some transfer function. You can use the bisection feature in a pass-fail mode or a bisection mode. The process is largely the same in each case.

Understanding the Optimization Syntax

Several Star-Hspice statements are required for optimization.

- .MODEL modname OPT ...
- .PARAM parameter=OPTxxx (init, min, max)
- A .DC, .AC, or .TRAN analysis statement with MODEL=modname, OPTIMIZE=OPTxxx, and RESULTS=measurename
- .MEASURE measurename ... <GOAL = | < | > val> – note that a space is required on either side of the relational operator =, <, or >

The .PARAM statement lets you specify initial, lower, and upper bound values. The types of .MEASURE statements available for optimization are described in Specifying Simulation Output on page 8-1.

Output statements .PRINT, .PLOT, and .GRAPH must be associated with the analysis statements .DC, .AC, or .TRAN. An analysis statement with the keyword OPTIMIZE is used for optimization only. To generate output for the optimized circuit, another analysis statement (.DC, .AC, or .TRAN) must be specified, along with the output statements. The proper specification order is:

1. Analysis statement with OPTIMIZE
2. .MEASURE statements specifying optimization goals or error functions
3. Ordinary analysis statement
4. Output statements

Analysis Statement (.DC, .TRAN, .AC)

The syntax is:

```
.DC <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ...
+ ierrn MODEL=optmod
```

or

```
.AC <DATA=filename> SWEEP OPTIMIZE=OPTxxx RESULTS=ierr1 ...
+ ierrn MODEL=optmod
```

or

```
.TRAN  <DATA=filename> SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
```

where:

DATA	Specifies the in-line file of parameter data to use in the optimization
OPTIMIZE	Indicates the analysis is for optimization. Specifies the parameter reference name used in the .PARAM optimization statement. All .PARAM optimization statements with the parameter reference name selected by OPTIMIZE will have their associated parameters varied during an optimization analysis.
MODEL	The optimization reference name that is also specified in the .MODEL optimization statement.
RESULTS	The measurement reference name that is also specified in the .MEASURE optimization statement. RESULTS is used to pass analysis data to the .MEASURE optimization statement.

.PARAM Statement

The syntax is:

```
.PARAM parameter=OPTxxx (initial_guess, low_limit,
+ upper_limit)
```

or

```
.PARAM parameter=OPTxxx (initial_guess, low_limit,
+ upper_limit, delta)
```

where:

OPTxxx	Specifies the optimization parameter reference name, referenced by the associated optimization analysis. This must agree with the OPTxxx name given in the analysis command associated with the keyname OPTIMIZE.
parameter	Specifies the parameter to be varied, the initial value estimate, the lower limit, and the upper limit allowed for the parameter. If the best solution does not exist within these constraints, the optimizer attempts to find the best solution.
delta	The final parameter value is the initial guess ± (*n*·delta). If delta is not specified, the final parameter value can be anything between low_limit and upper_limit. This is useful for optimizing transistor drawn widths and lengths, which must be quantized.

Example

```
.PARAM vtx=OPT1(.7,.3,1.0) uox=OPT1(650,400,900)
```

In the above example, the parameters *uox* and *vtx* are the variable model parameters to optimize a model for a given set of electrical specifications. The parameter *vtx* is given an initial value estimate of 0.7 volts and can be varied within the limits of 0.3 and 1.0 volts for the optimization procedure. The optimization parameter reference name, OPT1, is used to reference the associated optimization analysis statement (not shown).

.MODEL Statement

For each optimization within a data file, specify a .MODEL statement to allow for more than one optimization per simulation run to be executed. The optimization .MODEL statement defines the convergence criteria, number of iterations, and derivative methods.

The syntax is:

```
.MODEL mname OPT <parameter=val ...>
```

You can specify the following OPT parameters in the .MODEL statement:

mname	Model name. Elements refer to the model by this name.
CENDIF	Represents the point at which more accurate derivatives are required. When the gradient of the RESULTS functions are less than CENDIF, the more time-consuming derivative methods are used. Values of 0.1 to 0.01 are suitable for most applications. If too large a value is used, the optimizer requires more CPU time. If too small a value is used, it might not find as accurate an answer. Default=1.0e-9.
CLOSE	The initial estimate of how close the parameter initial value estimates are to the final solution. CLOSE multiplies the changes in the new parameter estimates. A large value for CLOSE causes the optimizer to take large steps toward the solution and a small value causes the optimizer to take smaller steps toward the solution. CLOSE should range from 0.01 for very close parameter estimates to 10 for rough initial guesses. Default=1.0. If CLOSE is greater than 100, the steepest descent part of the Levenburg-Marquardt algorithm dominates. For CLOSE less than 1, the Gauss-Newton method dominates. For further details, see L. Spruiell, “Optimization Error Surfaces,” *Meta-Software Journal,* Vol. 1, No. 4, December 1994.

CUT	Modifies CLOSE, depending on how successful the iterations toward the solution become. If the last iteration was successful, descent toward the solution CLOSE is decreased by CUT. If the last iteration was not a successful descent to the solution, CLOSE is increased by CUT squared. CUT drives CLOSE up or down depending on the relative success in finding the solution. The CUT value must be greater than 1. Default = 2.0.
DIFSIZ	Determines the increment change in a parameter value for gradient calculations ($\Delta x = DIFSIZ \cdot max(x, 0.1))$. If delta is specified in a .PARAM statement, then Δx = delta. Default = 1e-3.
GRAD	Represents a possible convergence when the gradient of the RESULTS function is less than GRAD. Values of 1e-6 to 1e-5 are suitable for most applications. If too large a value is used, the optimizer could stop before the best solution is found. Too small a value requires more iterations. Default=1.0e-6.
ITROPT	Sets the maximum number of iterations. Typically no more than 20-40 iterations are needed to find a solution. Too many iterations can imply the values for RELIN, GRAD, or RELOUT are too small. Default=20.
LEVEL	Selects the optimizing algorithm to use. Currently, the only option is LEVEL=1, a modified Levenburg-Marquardt algorithm.
MAX	Sets the upper limit on CLOSE. Values greater than 100 are recommended. Default=6000.

PARMIN

Allows better control of incremental parameter changes during error calculations. This produces more control over the trade-off between simulation time and optimization result accuracy. Star-Hspice calculates parameter increments using the relationship:

$$\Delta par\_val = DIFSIZ \cdot MAX(par\_val, PARMIN)$$

Default=0.1.

RELIN

Specifies the relative input parameter variation for convergence. If all the optimizing input parameters vary by no more than RELIN from one iteration to the next, then the solution is declared convergent. Since RELIN is a relative variance test, a value of 0.001 implies that the optimizing parameters are varying by less then 0.1% from one iteration to the next. Default=0.001.

RELOUT

Represents the relative output RESULTS function variance for convergence. For RELOUT=0.001, the difference in the RMS error of the RESULTS functions should vary less than 0.001. Default=0.001.

Optimization Examples

This section provides examples of the following types of Star-Hspice optimizations:

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*). The Level 3 parameters VTO, GAMMA, UO, VMAX, THETA, and KAPPA are optimized. After optimization, the model is compared separately to the data for the gate and drain curves. The option POST generates AvanWaves files for comparing the model to the data.

Level 3 Model DC Optimization Input Netlist File

```
$LEVEL 3 mosfet optimization
$..tighten the simulator convergence properties
.OPTION nomod post=2 newtol relmos=1e-5 absmos=1e-8
.MODEL optmod OPT itropt=30
```

Circuit Input

```
vds 30 0 vds
vgs 20 0 vgs
vbs 40 0 vbs
m1 30 20 0 40 nch w=50u l=4u
```

```
$..
$..process skew parameters for this data
.PARAM xwn=-0.3u xln=-0.1u toxn=196.6 rshn=67

$..the model and initial guess
.MODEL nch NMOS LEVEL=3
+ acm=2 ldif=0 hdif=4u tlev=1 n=2
+ capop=4 meto=0.08u xqc=0.4

$...note capop=4 is ok for H8907 and later, otherwise
$...use Capop=2
$...fixed parameters
+ wd=0.15u ld=0.07u
+ js=1.5e-04 jsw=1.8e-09
+ cj=1.7e-04 cjsw=3.8e-10
+ nfs=2e11 xj=0.1u delta=0 eta=0

$...process skew parameters
+ tox=toxn rsh=rshn
+ xw=xwn xl=xln
```

Optimized Parameters

```
+ vto=vto gamma=gamma
+ uo=uo vmax=vmax theta=theta kappa=kappa

.PARAM

+ vto    = opt1(1,0.5,2)
+ gamma  = opt1(0.8,0.1,2)
+ uo     = opt1(480,400,1000)
+ vmax   = opt1(2e5,5e4,5e7)
+ theta  = opt1(0.05,1e-3,1)
+ kappa  = opt1(2,1e-2,5)
```

Optimization Sweeps

```
.DC DATA=all optimize=opt1 results=comp1 model=optmod
.MEAS DC comp1 ERR1 par(ids) i(m1) minval=1e-04 ignor=1e-05
```

DC Sweeps

```
.DC DATA=gate
.DC DATA=drain
```

Print Sweeps

```
.PRINT DC vds=par(vds) vgs=par(vgs) im=i(m1) id=par(ids)
.PRINT DC vds=par(vds) vgs=par(vgs) im=i(m1) id=par(ids)
```

DC Sweep Data

```
$..data
.PARAM vds=0 vgs=0 vbs=0 ids=0
.DATA all vds vgs vbs ids
1.000000e-01 1.000000e+00 0.000000e+00 1.655500e-05
5.000000e+00 5.000000e+00 0.000000e+00 4.861000e-03
.ENDATA

.DATA gate vds vgs vbs ids
1.000000e-01 1.000000e+00 0.000000e+00 1.655500e-05
1.000000e-01 5.000000e+00 -2.000000e+00 3.149500e-04
.ENDDATA

.DATA drain vds vgs vbs ids
2.500000e-01 2.000000e+00 0.000000e+00 2.302000e-04
5.000000e+00 5.000000e+00 0.000000e+00 4.861000e-03
.ENDDATA

.END
```

The Star-Hspice input netlist shows:

- Using .OPTIONS to tighten tolerances increases the accuracy of Star-Hspice. This is recommended for I-V optimization.
- ".MODEL optmod OPT itropt=30" limits the number of iterations to 30.
- The circuit is simply one transistor. VDS, VGS, VBS are parameter names that match names used in the data statements.
- The process variation parameters, XL, XW, TOX, RSH are specified as constants in a .PARAM statement. These are measured parameters for the device being characterized.
- The model contains references to parameters. In "GAMMA= GAMMA", the left-hand side is a Level 3 model parameter name, while the right-hand side is a ".PARAM" parameter name.
- The long .PARAM statement specifies initial, min and max values for the optimized parameters. UO is initialized at 480 and kept within the range 400 to 1000 during optimization.
- The first .DC statement indicates that the data is in the in-line ".DATA all" block (which contains merged gate and drain curve data), optimization of parameters that were declared as OPT1 (in this case all of the optimized parameters), error function COMP1 (matches the name of a .MEASURE statement), and model OPTMOD (sets iteration limit).

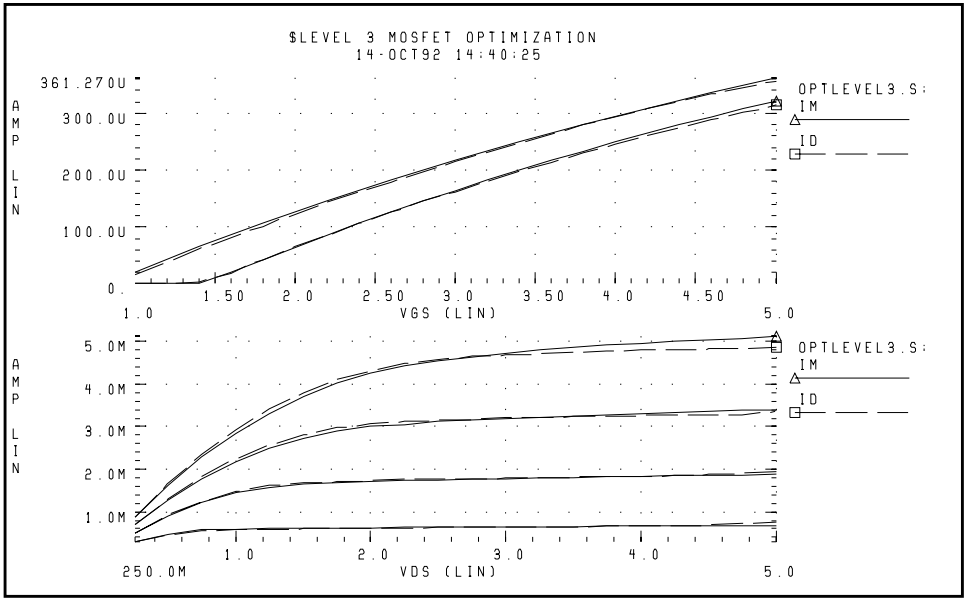
- The .MEASURE statement specifies least-squares relative error. The difference between data par(ids) and model i(m1) is divided by either the absolute value of par(ids), or by minval=10e-6, whichever is larger. Using minval keeps low current data from dominating the error.
- The remaining .DC and .PRINT statements are for printback after optimization. You can be place them anywhere in the netlist input file because they will be correctly assigned when the file is parsed.
- The ".PARAM VDS=0 VGS=0 VBS=0 IDS=0" statements simply declare these data column names as parameters.

The .DATA statements give data for IDS versus VDS, VGS, VBS. The selection of data should match the choice of model parameters to optimize. To optimize GAMMA, data with back bias must be provided (VBS= -2 in this case). To optimize KAPPA, the saturation region must have data. In this example, the data set "all" contain:

- gate curves: vds=0.1 vbs=0,-2 vgs=1 to 5 in steps 0.25
- drain curves: vbs=0 vgs=2,3,4,5 vds=0.25 to 5 in steps 0.25

The results are shown in Figure 13-20.

Figure 13-20: Level 3 MOSFET Optimization



MOS Level 13 Model DC Optimization

This example shows an optimization of I-V data 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. The Level 13 parameters vfb0, k1, muz, x2m, and u00 are optimized to the gate data. Then the Level 13 parameters MUS, X3MS, U1, and the impact ionization parameter ALPHA are optimized to the drain data. After optimization, the model is compared to the data. The option POST generates AvanWaves files for comparing the model to the data.

The results are shown in Figure 13-21.

Level 13 Model DC Optimization Input Netlist File

```
$LEVEL 13 mosfet optimization
$..tighten the simulator convergence properties
.OPTION nomod post=2
+ newtol relmos=1e-5 absmos=1e-8
.MODEL optmod OPT itropt=30
```

Circuit Input

```
vds 30 0 vds
vgs 20 0 vgs
vbs 40 0 vbs
m1 30 20 0 40 nch w=50u l=4u
$..
$..process skew parameters for this data
.PARAM xwn=-0.3u xln=-0.1u toxn=196.6 rshn=67

$..the model and initial guess
.MODEL nch NMOS LEVEL=13
+ acm=2 ldif=0 hdif=4u tlev=1 n=2 capop=4 meto=0.08u
+ xqc=0.4

$...parameters obtained from measurements
+ wd=0.15u ld=0.07u js=1.5e-04 jsw=1.8e-09
+ cj=1.7e-04 cjsw=3.8e-10
$...parameters not used for this data
+ k2=0 eta0=0 x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x3u1=0

$...process skew parameters
+ toxm=toxn rsh=rshn
+ xw=xwn xl=xln
```

```
$...optimized parameters
+ vfb0=vfb0 k1=k1 x2m=x2m muz=muz u00=u00
+ mus=mus x3ms=x3ms u1=u1

$...impact ionization parameters
+ alpha=alpha vcr=15
.PARAM
+ vfb0    = opt1(-0.5, -2, 1)
+ k1      = opt1(0.6,0.3,1)
+ muz     = opt1(600,300,1500)
+ x2m     = opt1(0,-10,10)
+ u00     = opt1(0.1,0,0.5)
+ mus     = opt2(700,300,1500)
+ x3ms    = opt2(5,0,50)
+ u1      = opt2(0.1,0,1)
+ alpha   = opt2(1,1e-3,10)
```

Optimization Sweeps

```
.DC DATA=gate optimize=opt1 results=comp1 model=optmod
.MEAS DC comp1 ERR1 par(ids) i(m1) minval=1e-04 ignor=1e-05
.DC DATA=drain optimize=opt2 results=comp2 model=optmod
.MEAS DC comp2 ERR1 par(ids) i(m1) minval=1e-04 ignor=1e-05
```

DC Data Sweeps

```
.DC DATA=gate
.DC DATA=drain
```

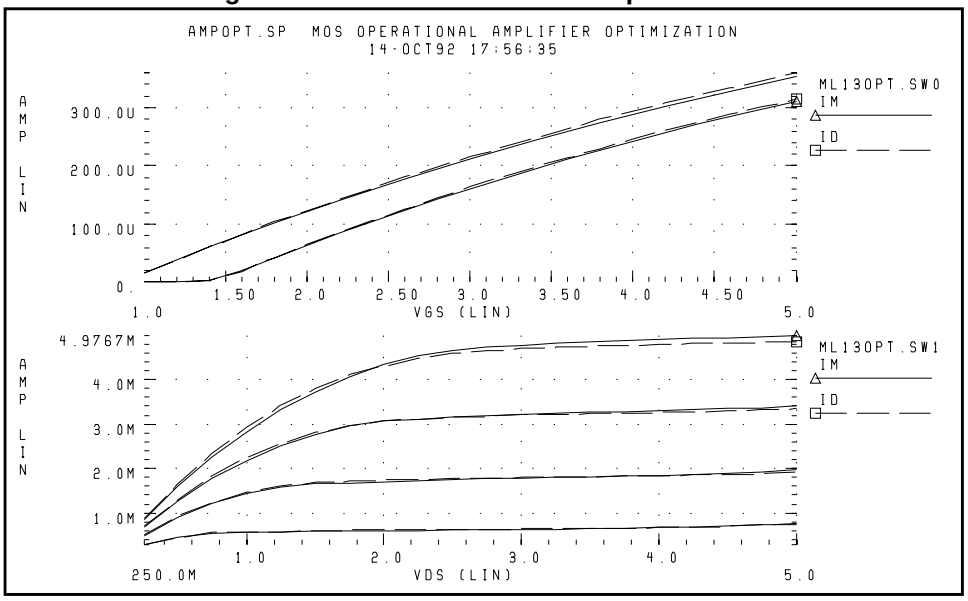
Print Sweeps

```
.PRINT DC vds=par(vds) vgs=par(vgs) im=i(m1) id=par(ids)
.PRINT DC vds=par(vds) vgs=par(vgs) im=i(m1) id=par(ids)
```

DC Sweep Data

```
$..data
.PARAM vds=0 vgs=0 vbs=0 ids=0
.DATA gate vds vgs vbs ids
1.000000e-01 1.000000e+00 0.000000e+00 1.655500e-05
1.000000e-01 5.000000e+00 -2.000000e+00 3.149500e-04
.ENDDATA
.DATA drain vds vgs vbs ids
2.500000e-01 2.000000e+00 0.000000e+00 2.809000e-04
5.000000e+00 5.000000e+00 0.000000e+00 4.861000e-03
.ENDDATA
.END
```

Figure 13-21: Level 13 MOSFET Optimization



RC Network Optimization

Following is an example of optimizing the power dissipation and time constant of an RC network. The circuit is a parallel resistor and capacitor. The following design targets are specified.

- 1 s time constant
- 50 mW rms power dissipation through the resistor

The Star-Hspice strategy is as follows:

- .MEASURE statement RC1 calculates RC time constant (GOAL of .3679 V corresponds to 1 s time constant e^{-rc}).
- .MEASURE statement RC2 calculates the rms power where the GOAL is 50 mW.
- OPTrc identifies RX and CX as optimization parameters and sets their starting, minimum, and maximum values.

Star-Hspice features used:

- Measure voltages and report times subject to goal
- Measure device power dissipation subject to goal

- Measure statements replace tabular or plot output
- Element value parameterization
- Parameter optimization function
- Transient with SWEEP optimize

RC Network Optimization Input Netlist File

```
.title RCOPT.sp
.option post

.PARAM RX=OPTRC(.5, 1E-2, 1E+2)
.PARAM CX=OPTRC(.5, 1E-2, 1E+2)

.MEASURE TRAN RC1 TRIG AT=0 TARG V(1) VAL=.3679 FALL=1
+ GOAL=1sec
.MEASURE TRAN RC2 RMS P(R1) GOAL=50mwatts

.MODEL OPT1 OPT

.tran .1 2        $ initial values
.tran .1 2 SWEEP OPTIMIZE=OPTrc RESULTS=RC1,RC2 MODEL=OPT1
.tran .1 2       $ analysis using final optimized values

.ic 1 1
R1 1 0 RX
c1 1 0 CX
```

Optimization Results

```
RESIDUAL SUM OF SQUARES     = 1.323651E-06
NORM OF THE GRADIENT        = 6.343728E-03
MARQUARDT SCALING PARAMETER = 2.799235E-06
NO. OF FUNCTION EVALUATIONS =       24
NO. OF ITERATIONS           =       12
```

Residual Sum of Squares

The residual sum of squares is a measure of the total error. The smaller this value is, the more accurate the optimization results are.

$$\text{residual sum of squares} = \sum_{i=1}^{ne} E_i^2$$

where *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 gradient G is found by

$$G_j = \sum_{i=1}^{ne} E_i \cdot (\Delta E_i / \Delta P_j)$$

and

$$\text{norm of the gradient} = 2 \cdot \sqrt{\sum_{i=1}^{np} G_j^2}$$

where *P* is the parameter and *np* is the number of parameters to be optimized.

Marquardt Scaling Parameter

This parameter is used in the Levenburg-Marquardt algorithm to find the actual solution of the optimizing parameters. The search direction is a combination of the Steepest Descent method and the Gauss-Newton method.

The Steepest Descent method is used initially to approach the solution because it is fast, and then the Gauss-Newton method is used 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 optimal solution is not attained, an error message is printed and a large Marquardt Scaling Parameter value is printed.

Number of Function Evaluations

This is the number of analyses (for example, finite difference or central difference) that were needed to find a minimum of the function.

Number of Iterations

This is the number of iterations needed to find the optimized or actual solution.

Optimized Parameters OPTRC

```
*                                          %NORM-SEN   %CHANGE
.PARAM RX              =    6.7937    $   54.5260     50.2976M
.PARAM CX              =  147.3697M $   45.4740     33.7653M
```

Figure 13-22: Power Dissipation and Time Constant (VOLT) RCOPT.TR0 = Before Optimization, RCOPT.TR1 = Optimized Result

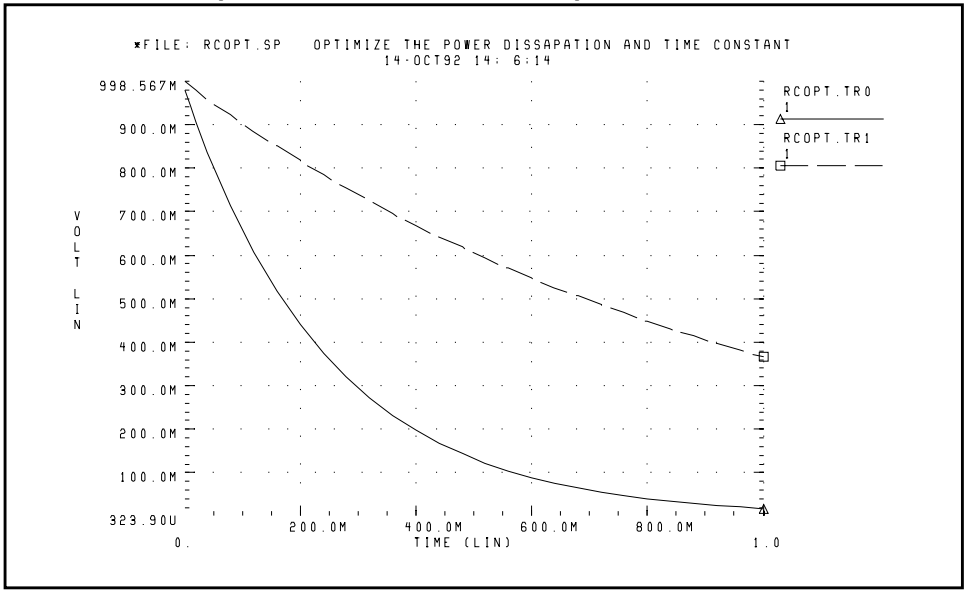
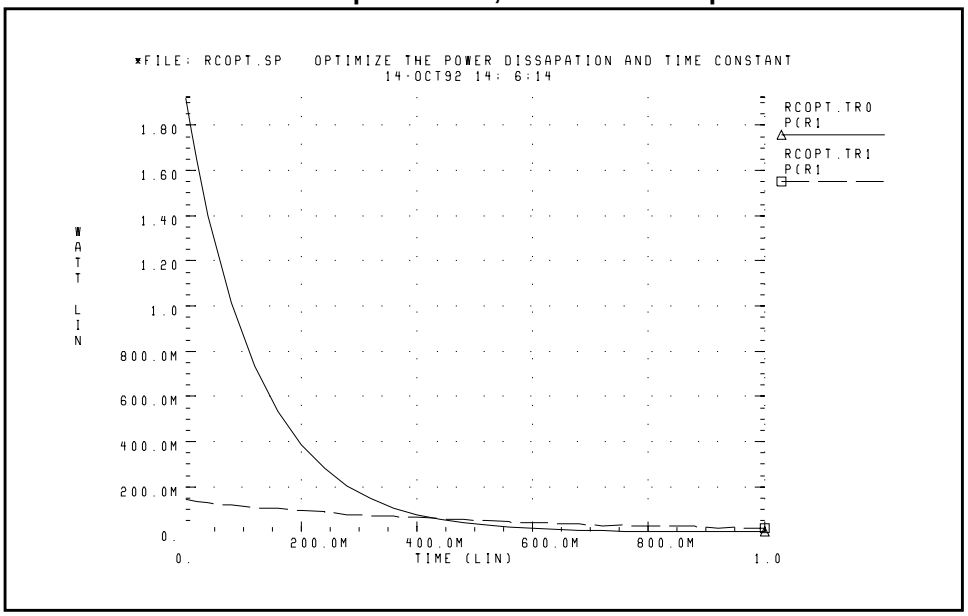


Figure 13-23: Power Dissipation and Time Constant (WATT)
RCOPT.TR0 = Before Optimization, RCOPT.TR1 = Optimized Result



CMOS Tristate Buffer Optimization

The example circuit is an inverting CMOS tristate buffer. The following design targets are specified:

1. Rising edge delay of 5 ns (input 50% voltage to output 50% voltage)
2. Falling edge delay of 5 ns (input 50% voltage to output 50% voltage)
3. RMS power dissipation should be as low as possible
4. Output load consists of
   - ❑ pad capacitance
   - ❑ leadframe inductance
   - ❑ 50 pF capacitive load

The Star-Hspice strategy is as follows:

- ■ Simultaneously optimize rising delay buffer and falling delay buffer.
- ■ Set up internal power supplies and tristate enable as global nodes.

- Optimize all device widths except:
  - Initial inverter (this is assumed to be standard size)
  - Tristate inverter and part of tristate control (the optimization is not sensitive to this path)
- Perform initial transient analysis for plotting purposes, then optimize and perform a final transient for plotting purposes.
- Use a weighted RMS power measure by specifying an unrealistically low power goal and using MINVAL to attenuate the error.

CMOS Tristate Buffer Optimization Input Netlist File

```
*Tri-State input/output Optimization
.options defl=1.2u nomod post=2
+ relv=1e-3 absvar=.5 relvar=.01
```

Circuit Input

```
.global lgnd lvcc enb
.macro buff data out
mp1 DATAN DATA LVCC LVCC p w=35u
mn1 DATAN DATA LGND LGND n w=17u

mp2 BUS DATAN LVCC LVCC p w=wp2
mn2 BUS DATAN LGND LGND n w=wn2

mp3 PEN PENN LVCC LVCC p w=wp3
mn3 PEN PENN LGND LGND n w=wn3

mp4 NEN NENN LVCC LVCC p w=wp4
mn4 NEN NENN LGND LGND n w=wn4

mp5 OUT PEN   LVCC LVCC p w=wp5 l=1.8u
mn5 OUT NEN   LGND LGND n w= wn5 l=1.8u

mp10 NENN BUS LVCC LVCC p w=wp10
mn12 PENN ENB NENN LGND n w=wn10
mn10 PENN BUS LGND LGND n w=wn10
mp11 NENN ENB LVCC LVCC p w=wp11
mp12 NENN ENBN PENN LVCC p w=wp11
mn11 PENN ENBN LGND LGND n w=80u
mp13 ENBN ENB LVCC LVCC p w=35u
mn13 ENBN ENB LGND LGND n w=17u

cbus BUS LGND 1.5pf
cpad OUT LGND 5.0pf

.ends
```

```
* * input signals *
vcc VCC GND 5V

lvcc vcc lvcc 6nh
lgnd lgnd gnd 6nh
vin DATA    LGND pl (0v 0n, 5v 0.7n)
vinb DATAbar LGND pl (5v 0n, 0v 0.7n)
ven ENB GND 5V

** circuit **
x1 data out buff
cext1 out GND 50pf
x2 databar outbar buff
cext2 outbar GND 50pf
```

Optimization Parameters

```
.param
+ wp2=opt1(70u,30u,330u)
+ wn2=opt1(22u,15u,400u)
+ wp3=opt1(400u,100u,500u)
+ wn3=opt1(190u,80u,580u)
+ wp4=opt1(670u,150u,800u)
+ wn4=opt1(370u,50u,500u)
+ wp5=opt1(1200u,1000u,5000u)
+ wn5=opt1(600u,400u,2500u)
+ wp10=opt1(240u,150u,450u)
+ wn10=opt1(140u,30u,280u)
+ wp11=opt1(240u,150u,450u)
```

Control Section

```
.tran 1ns 15ns
.tran .5ns 15ns sweep optimize=opt1
+ results=tfopt,tropt,rmspowo model=optmod

** put soft limit for power with minval setting (i.e. values
** less than 1000mw are less important)

.measure rmspowo rms power goal=100mw minval=1000mw
.mea tran tfopt trig v(data) val=2.5 rise=1 targ v(out)
+ val=2.5 fall=1 goal 5.0n

.mea tran tropt trig v(databar) val=2.5 fall=1 targ
+ v(outbar) val=2.5 rise=1 goal 5.0n

.model optmod opt itropt=30 max=1e+5
```

```
.tran 1ns 15ns
* output section *
*.plot tran v(data) v(out)
.plot tran v(databar) v(outbar)
```

Model Section

```
.MODEL N NMOS LEVEL=3 VTO=0.7 UO=500 KAPPA=.25 KP=30U
+ ETA=.03 THETA=.04 VMAX=2E5 NSUB=9E16 TOX=500E-10
+ GAMMA=1.5 PB=0.6 JS=.1M XJ=0.5U LD=0.0 NFS=1E11 NSS=2E10
+ CGSO=200P CGDO=200P CGBO=300P

.MODEL P PMOS LEVEL=3 VTO=-0.8 UO=150 KAPPA=.25 KP=15U
+ ETA=.03 THETA=.04 VMAX=5E4 NSUB=1.8E16 TOX=500E-10
+ NFS=1E11 GAMMA=.672 PB=0.6 JS=.1M XJ=0.5U LD=0.0
+ NSS=2E10 CGSO=200P CGDO=200P CGBO=300P

.end
```

Optimization Results

```
residual sum of squares     = 2.388803E-02
norm of the gradient        = 0.769765
marquardt scaling parameter =   12624.2
no. of function evaluations =      175
no. of iterations           =       23
```

Optimization Completed

```
Parameters relin= 1.0000E-03 on last iterations
```

Optimized Parameters OPT1

```
*                                    %norm-sen   %change

.param wp2           =  84.4981u  $   22.5877  -989.3733u
.param wn2           =  34.1401u  $    7.6568  -659.2874u
.param wp3           = 161.7354u  $  730.7865m -351.7833u
.param wn3           = 248.6829u  $    8.1362    -2.2416m
.param wp4           = 238.9825u  $    1.2798    -1.5774m
.param wn4           =  61.3509u  $  315.4656m   43.5213m
.param wp5           =   1.7753m  $    4.1713     2.1652m
.param wn5           =   1.0238m  $    5.8506   413.9667u
.param wp10          = 268.3125u  $    8.1917    -2.0266m
.param wn10          = 115.6907u  $   40.5975  -422.8411u
.param wp11          = 153.1344u  $  482.0655m  -30.6813m
```

```
*** optimize results measure names and values
* tfopt            =   5.2056n
* tropt            =   5.5513n
* rmspowo          = 200.1808m
```

Figure 13-24: Tristate Buffer Optimization Circuit

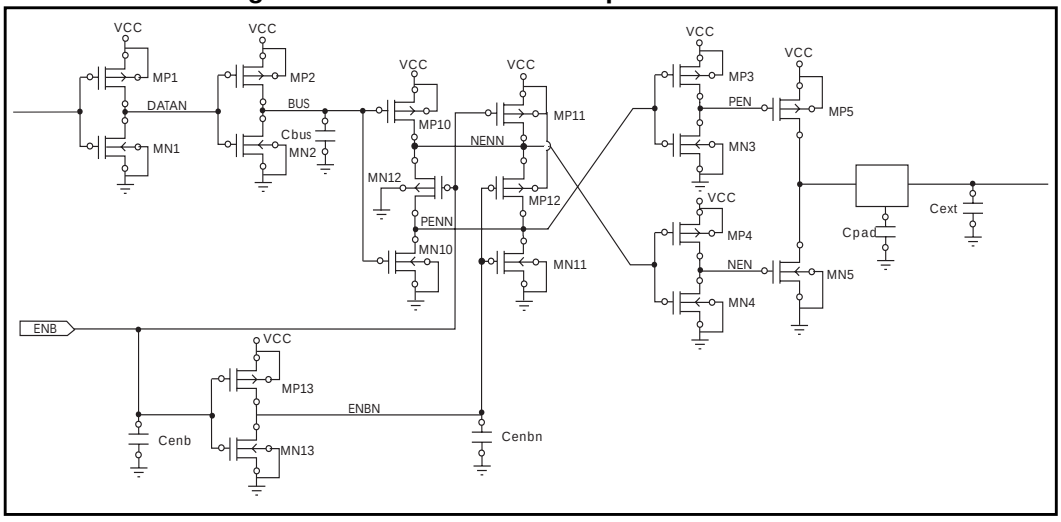
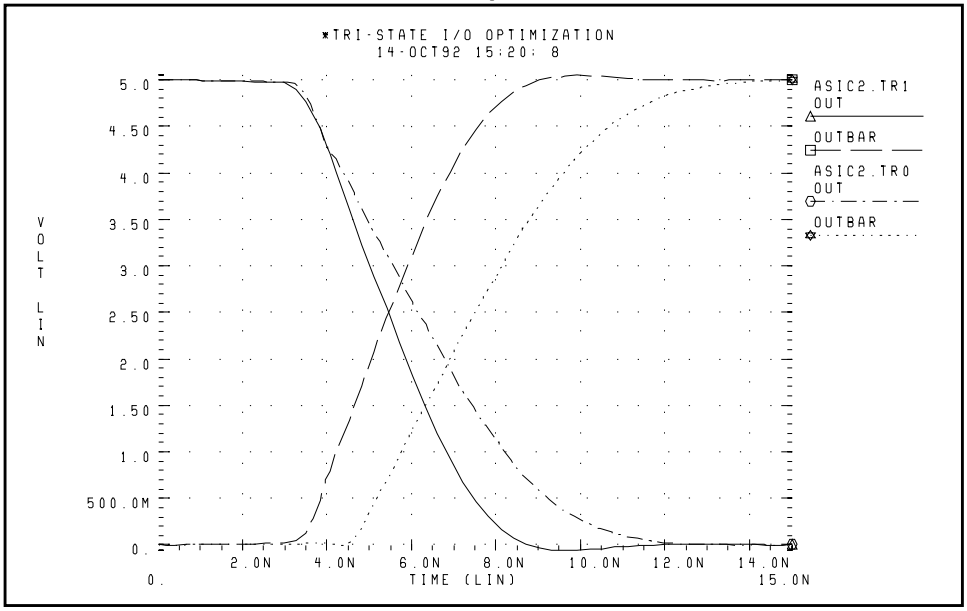


Figure 13-25: Tristate Input/Output Optimization ACIC2B.TR0 = Before Optimization, ACIC2B.TR1 = Optimized Result



BJT S Parameters Optimization

In the following example, the S parameters are optimized to match those given for a set of measurements. These measured S parameters, as a function of frequency, are in the ".DATA MEASURED" in-line data statement. The model parameters of the microwave transistor (LBB, LCC, LEE, TF, CBE, CBC, RB, RE, RC, and IS) are varied so that measured S parameters in the .DATA statement matches the calculated S parameters from the simulation results.

This optimization uses a 2n6604 microwave transistor and an equivalent circuit consisting 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 Star-Hspice Features Used

- NET command to simulate network analyzer action
- AC optimization
- Optimization of element and model parameters

- Optimization comparing measured S parameters versus calculated parameters
- S parameters used in magnitude and phase (real and imaginary available)
- Data-driven frequency versus S parameter table weighting used for phase domain

BJT S Parameters Optimization Input Netlist File

```
* BJTOPT.SP BJT S PARAMETER OPTIMIZATION
.OPTION ACCT NOMOD POST=2
```

BJT Equivalent Circuit Input

```
* THE NET COMMAND IS AUTOMATICALLY REVERSING THE SIGN OF
* THE POWER SUPPLY CURRENT FOR THE NETWORK CALCULATIONS
.NET I(VCE) IBASE ROUT=50 RIN=50
VCE VCE 0 10V
IBASE 0 IIN   AC=1 DC=.1MA
LBB IIN BASE LBB
LCC VCE COLLECT LCC
LEE EMIT 0 LEE
Q1 COLLECT BASE EMIT T2N6604

.MODEL T2N6604 NPN RB=RB BF=100 TF=TF CJE=CBE CJC=CBC
+ RE=RE RC=RC IS=IS

.PARAM
+ LBB= OPT1(100P, 1P, 10N)
+ LCC= OPT1(100P, 1P, 10N)
+ LEE= OPT1(100P, 1P, 10N)
+ TF = OPT1(1N, 5P, 5N)
+ CBE= OPT1(.5P, .1P, 5P)
+ CBC= OPT1(.4P, .1P, 5P)
+ RB= OPT1(10, 1, 300)
+ RE= OPT1(.4, .01, 5)
+ RC= OPT1(10, .1, 100)
+ IS= OPT1(1E-15, 1E-16, 1E-10)

.AC DATA=MEASURED OPTIMIZE=OPT1
+ RESULTS=COMP1,COMP3,COMP5,COMP6,COMP7
+ MODEL=CONVERGE

.MODEL CONVERGE OPT RELIN=1E-4 RELOUT=1E-4 CLOSE=100
+ ITROPT=25
```

```
.MEASURE AC COMP1 ERR1 PAR(S11M) S11(M)
.MEASURE AC COMP2 ERR1 PAR(S11P) S11(P) MINVAL=10
.MEASURE AC COMP3 ERR1 PAR(S12M) S12(M)
.MEASURE AC COMP4 ERR1 PAR(S12P) S12(P) MINVAL=10
.MEASURE AC COMP5 ERR1 PAR(S21M) S21(M)
.MEASURE AC COMP6 ERR1 PAR(S21P) S21(P) MINVAL=10
.MEASURE AC COMP7 ERR1 PAR(S22M) S22(M)

.AC DATA=MEASURED

.PRINT PAR(S11M) S11(M) PAR(S11P) S11(P)
.PRINT PAR(S12M) S12(M) PAR(S12P) S12(P)
.PRINT PAR(S21M) S21(M) PAR(S21P) S21(P)
.PRINT PAR(S22M) S22(M) PAR(S22P) S22(P)

.DATA MEASURED
FREQ   S11M   S11P   S21M    S21P   S12M   S12P S22M   S22P
100ME  .6    -52     19.75   148    .02     65    .87   - 21
200ME  .56   -95     15.30   127    .032    49    .69   - 33
500ME  .56   -149     7.69    97    .044    41    .45   - 41
1000ME .58   -174     4.07    77    .061    42    .39   - 47
2000ME .61    159     2.03    50    .095    40    .39   - 70

.ENDDATA

.PARAM FREQ=100ME S11M=0, S11P=0, S21M=0, S21P=0, S12M=0,
+ S12P=0, S22M=0, S22P=0

.END
```

Optimization Results

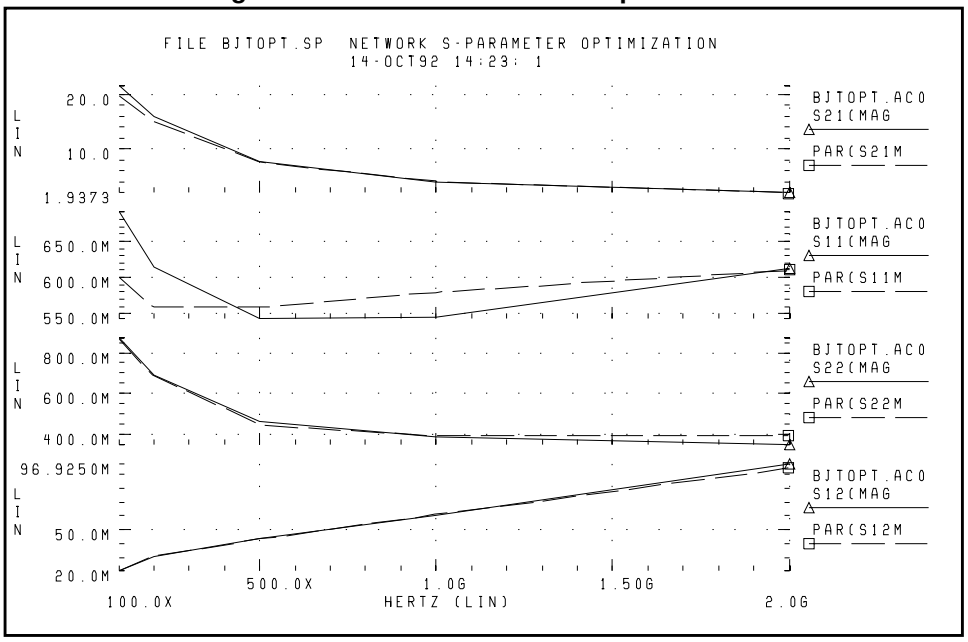
```
RESIDUAL SUM OF SQUARES     = 5.142639e-02
NORM OF THE GRADIENT        = 6.068882e-02
MARQUARDT SCALING PARAMETER = 0.340303
CO. OF FUNCTION EVALUATIONS = 170
NO. OF ITERATIONS           = 35
```

The maximum number of iterations (25) was exceeded. However, the results probably are accurate. Increase ITROPT accordingly.

Optimized Parameters OPT1– Final Values

```
***OPTIMIZED PARAMETERS OPT1 SENS  %NORM-SEN

.PARAM  LBB  =  1.5834N  $ 27.3566X  2.4368
.PARAM  LCC  =  2.1334N  $ 12.5835X  1.5138
.PARAM  LEE  =723.0995P  $254.2312X 12.3262
.PARAM  TF   = 12.7611P  $  7.4344G 10.0532
.PARAM  CBE  =620.5195F  $ 23.0855G  1.5300
.PARAM  CBC  =  1.0263P  $346.0167G 44.5016
.PARAM  RB   =  2.0582   $ 12.8257M  2.3084
.PARAM  RE   =869.8714M  $ 66.8123M  4.5597
.PARAM  RC   = 54.2262   $  3.1427M 20.7359
.PARAM  IS   = 99.9900P  $  3.6533X 34.4463M
```

Figure 13-26: BJT-S Parameter Optimization



BJT Model DC Optimization

The goal is to match the forward and reverse Gummel plots obtained from a HP4145 semiconductor analyzer with the Star-Hspice LEVEL=1 Gummel-Poon BJT model. Since the Gummel plots are at low base currents, the base resistance is not optimized. The forward and reverse Early voltages (VAF and VAR) are not optimized, since no VCE data was measured.

The key feature used in this optimization is incremental optimization. First the forward Gummel data points are optimized. The forward optimized parameters are updated into the model and not allowed to change. Then the reverse Gummel data points are optimized.

BJT Model DC Optimization Input Netlist File

```
* FILE OPT_BJT.SP BJT OPTIMIZATION T2N3947
* OPTIMIZE THE DC FORWARD AND REVERSE CHARACTERISTICS
+ FROM A GUMMEL PLOT
* ALL DC GUMMEL-POON DC PARAMETERS EXCEPT BASE
+ RESISTANCE AND EARLY VOLTAGES OPTIMIZED
*

$..TIGHTEN THE SIMULATOR CONVERGENCE PROPERTIES
.OPTION NOMOD INGOLD=2 NOPAGE VNTOL=1E-10 POST
+ NUMDGT=5 RELI=1E-4 RELV=1E-4

$..OPTIMIZATION CONVERGENCE CONTROLS
.MODEL OPTMOD OPT RELIN=1E-4 ITROPT=30 GRAD=1E-5 CLOSE=10
+ CUT=2 CENDIF=1E-6 RELOUT=1E-4 MAX=1E6
```

Room Temp Device

```
VBER BASE 0 VBE
VBCR BASE COL VBC
Q1 COL BASE 0 BJTMOD
```

Model and Initial Estimates

```
.MODEL BJTMOD NPN
+ ISS = 0. XTF = 1. NS = 1.
+ CJS = 0. VJS = 0.50000 PTF = 0.
+ MJS = 0. EG = 1.10000 AF = 1.
+ ITF = 0.50000 VTF = 1.00000
+ FC = 0.95000 XCJC = 0.94836
+ SUBS = 1
```

```
+ TF=0.0 TR=0.0 CJE=0.0 CJC=0.0 MJE=0.5 MJC=0.5 VJE=0.6
+ VJC=0.6 RB=0.3 RC=10 VAF=550 VAR=300
$..THESE ARE THE OPTIMIZED PARAMETERS
+ BF=BF IS=IS IKF=IKF ISE=ISE RE=RE
+ NF=NF NE=NE
$..THESE ARE FOR REVERSE BASE OPT
+ BR=BR IKR=IKR ISC=ISC
+ NR=NR NC=NC

.PARAM VBE=0 IB=0 IC=0 VCE_EMIT=0 VBC=0 IB_EMIT=0 IC_EMIT=0
+ BF= OPT1(100, 50, 350)
+ IS= OPT1(5E-15, 5E-16, 1E-13)
+ NF= OPT1(1.0, 0.9, 1.1)
+ IKF=OPT1(50E-3, 1E-3, 1)
+ RE= OPT1(10, 0.1, 50)
+ ISE=OPT1(1E-16, 1E-18, 1E-11)
+ NE= OPT1(1.5, 1.2, 2.0)
+ BR= OPT2(2, 1, 10)
+ NR= OPT2(1.0, 0.9, 1.1)
+ IKR=OPT2(50E-3, 1E-3, 1)
+ ISC=OPT2(1E-12, 1E-15, 1E-10)
+ NC= OPT2(1.5, 1.2, 2.0)

.DC DATA=BASEF SWEEP OPTIMIZE=OPT1 RESULTS=IBVBE,ICVBE
+ MODEL=OPTMOD

.MEAS DC IBVBE ERR1 PAR(IB) I2(Q1) MINVAL=1E-14
+ IGNORE=1E-16

.MEAS DC ICVBE ERR1 PAR(IC) I1(Q1) MINVAL=1E-14
+ IGNORE=1E-16

.DC DATA=BASER SWEEP OPTIMIZE=OPT2 RESULTS=IBVBER,ICVBER
+ MODEL=OPTMOD

.MEAS DC IBVBER ERR1 PAR(IB) I2(Q1) MINVAL=1E-14
+ IGNORE=1E-16

.MEAS DC ICVBER ERR1 PAR(IC) I1(Q1) MINVAL=1E-14
+ IGNORE=1E-16

.DC DATA=BASEF
.PRINT DC PAR(IC) I1(Q1) PAR(IB) I2(Q1)

.DC DATA=BASER
.PRINT DC PAR(IC) I1(Q1) PAR(IB) I2(Q1)
```

Optimization Results

```
RESIDUAL SUM OF SQUARES = 2.196240E-02
```

Optimized Parameters OPT1

```
+ %NORM-SEN %CHANGE
.PARAM BF = 1.4603E+02 $ 2.7540E+00 -7.3185E-07
.PARAM IS = 2.8814E-15 $ 3.7307E+00 -5.0101E-07
.PARAM NF = 9.9490E-01 $ 9.1532E+01 -1.0130E-08
.PARAM IKF = 8.4949E-02 $ 1.3782E-02 -8.8082E-08
.PARAM RE = 6.2358E-01 $ 8.6337E-02 -3.7665E-08
.PARAM ISE = 5.0569E-16 $ 1.0221E-01 -3.1041E-05
.PARAM NE = 1.3489E+00 $ 1.7806E+00 2.1942E-07
```

Optimization Results

```
RESIDUAL SUM OF SQUARES = 1.82776
```

Optimized Parameters OPT2

```
* %NORM-SEN %CHANGE
.PARAM BR = 1.0000E+01 $ 1.1939E-01 1.7678E+00
.PARAM NR = 9.8185E-01 $ 1.4880E+01 -1.1685E-03
.PARAM IKR = 7.3896E-01 $ 1.2111E-03 -3.5325E+01
.PARAM ISC = 1.8639E-12 $ 6.6144E+00 -5.2159E-03
.PARAM NC = 1.2800E+00 $ 7.8385E+01 1.6202E-03
```

Figure 13-27: BJT Optimization Forward Gummel Plots

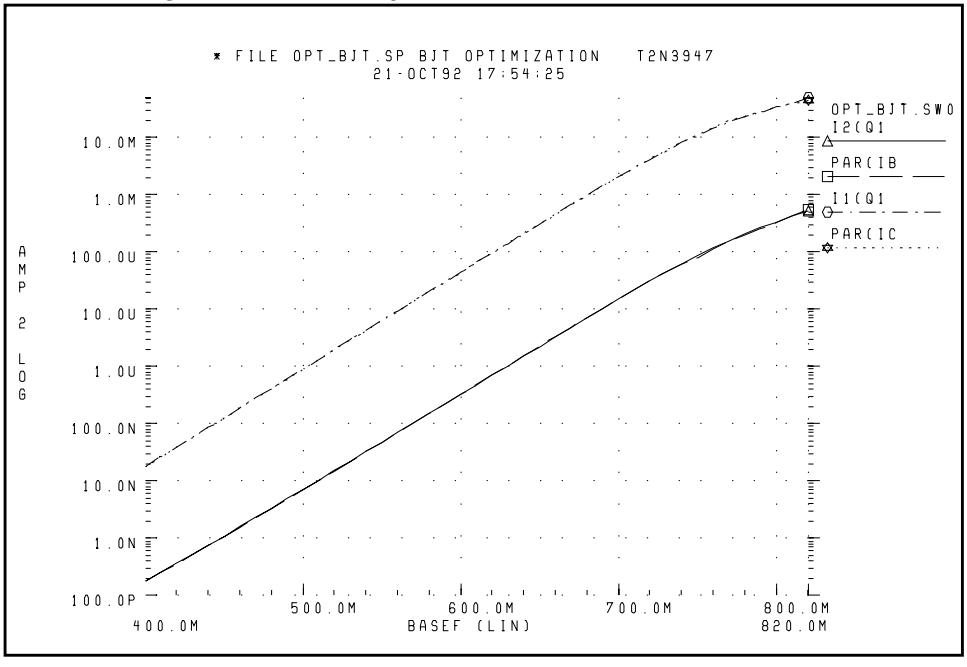
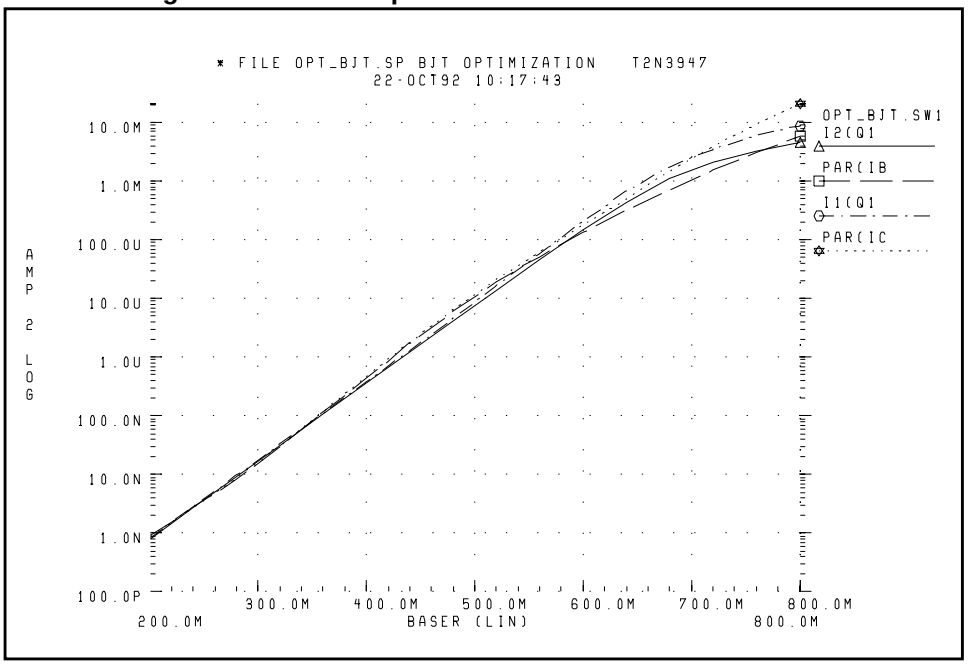


Figure 13-28: BJT Optimization Reverse Gummel Plots



GaAsFET Model DC Optimization

This example circuit is a high-performance GaAsFET transistor. The design target is to match HP4145 DC measured data to the Star-Hspice LEVEL=3 JFET model.

The Star-Hspice strategy is as follows:

- MEASURE IDSERR is an ERR1 type function providing linear attenuation of the error results, starting at 20 mA and ignoring all currents below 1 mA. The high current fit is the most important for this model.
- The OPT1 optimization function allows all DC parameters to be simultaneously optimized.
- The .DATA statement merges raw data files *TD1.dat* and *TD2.dat* together.
- The graph plot model sets the parameter MONO=1 to remove the retrace lines from the family of curves.

GaAsFET Model DC Optimization Input Netlist File

```
*FILE JOPT.SP JFET OPTIMIZATION
.OPTIONS ACCT NOMOD POST
+ RELI=2E-4 RELV=2E-4
VG GATE 0 XVGS
VD DRAIN 0 XVDS
J1 DRAIN GATE 0 JFETN1

.MODEL JFETN1 NJF LEVEL=3 CAPOP=1 SAT=3
+ NG=1
+ CGS=1P CGD=1P RG=1
+ VTO=VTO BETA=BETA LAMBDA=LAMBDA
+ RS=RDS RD=RDS IS=1E-15 ALPHA=ALPHA
+ UCRIT=UCRIT SATEXP=SATEXP
+ GAMDS=GAMDS VGEXP=VGEXP

.PARAM
+ VTO=OPT1(-.8,-4,0)
+ VGEXP=OPT1(2,1,3.5)
+ GAMDS=OPT1(0,-.5,0)
+ BETA= OPT1(6E-3, 1E-5,9E-2)
+ LAMBDA=OPT1(30M,1E-7,5E-1)
+ RDS=OPT1(1,.001,40)
+ ALPHA=OPT1(2,1,3)
+ UCRIT=OPT1(.1,.001,1)
+ SATEXP=OPT1(1,.5,3)

.DC DATA=DESIRED OPTIMIZE=OPT1 RESULTS=IDSERR MODEL=CONV
.MODEL CONV OPT RELIN=1E-4 RELOUT=1E-4 CLOSE=100 ITROPT=25

.MEASURE DC IDSERR ERR1 PAR(XIDS) I(J1) MINVAL=20M
+ IGNORE=1M

.DC DATA=DESIRED
.GRAPH PAR(XIDS) I(J1)
.MODEL GRAPH PLOT MONO=1
.PRINT PAR(XVGS) PAR(XIDS) I(J1)

.DATA DESIRED MERGE
+ FILE=JDC.DAT XVDS=1 XVGS=2 XIDS=3

.ENDDATA

.END
```

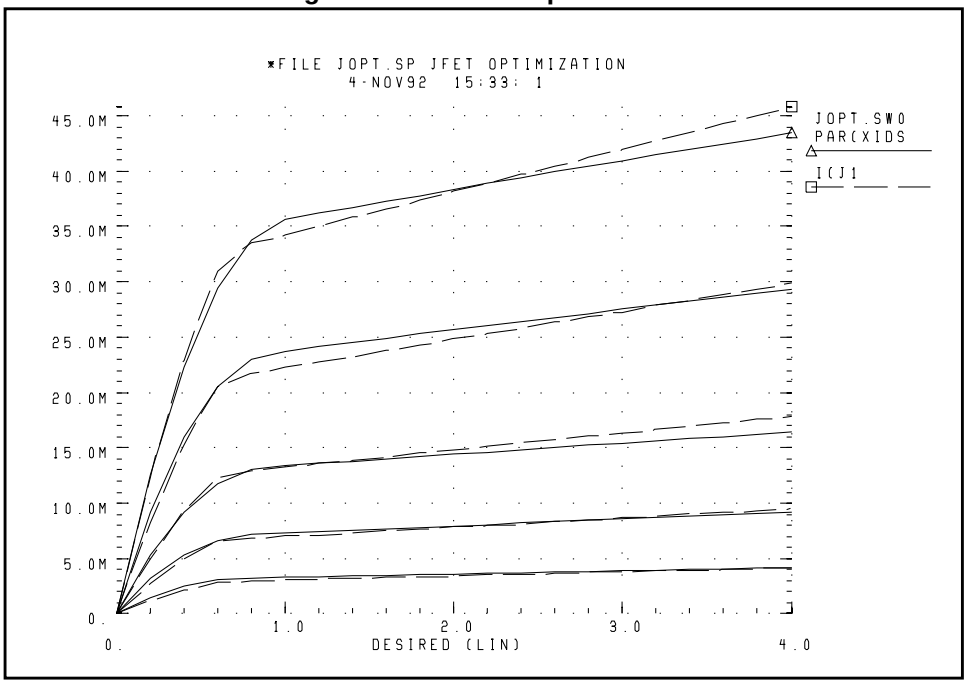
Optimization Results

```
RESIDUAL SUM OF SQUARES = 7.582202E-02
```

Optimized Parameters Opt1

```
*   %NORM-SEN      %CHANGE
.PARAM VTO         = -1.1067    $   64.6110      43.9224M
.PARAM VGEXP       = 2.9475     $   13.2024     219.4709M
.PARAM GAMDS       = 0.         $    0.           0.
.PARAM BETA        = 11.8701M   $   17.2347     136.8216M
.PARAM LAMBDA      = 138.9821M  $    2.2766      -1.5754
.PARAM RDS         = 928.3216M  $  704.3204M    464.0863M
.PARAM ALPHA       = 2.2914     $  728.7492M     168.4004M
.PARAM UCRIT       = 1.0000M    $   18.2438M    -125.0856
.PARAM SATEXP      = 1.4211     $    1.2241       2.2218
```

Figure 13-29: JFET Optimization



MOS Op-amp Optimization

The design goals for the MOS operational amplifier are:

- Minimize the gate area (and hence the total cell area).
- Minimize the power dissipation.
- Open-loop transient step response of 100 ns for rising and falling edges

The Star-Hspice strategy is:

- Simultaneous optimization of 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.
- Transistor area is calculated algebraically, a voltage value is used, and the resulting voltage is minimized.
- The transistor area measure statement uses MINVAL to give less weighting to the area minimization.
- Bias voltage is optimized.

MOS Op-amp Optimization Input Netlist File

```
AMPOPT.SP MOS OPERATIONAL AMPLIFIER OPTIMIZATION

.OPTION RELV=1E-3 RELVAR=.01 NOMOD ACCT POST
.PARAM VDD=5   VREF='VDD/2'
VDD VSUPPLY 0 VDD
VIN+    VIN+     0 PWL(0 ,'VREF-10M'   10NS 'VREF+10M' )
VINBAR+ VINBAR+  0 PWL(0 ,'VREF+10M'   10NS 'VREF-10M' )
VIN- VIN-  0  VREF
VBIAS VBIAS 0 BIAS
.GLOBAL VSUPPLY VBIAS

XRISE VIN+ VIN- VOUTR AMP
CLOADR VOUTR 0 .4P
XFALL VINBAR+ VIN- VOUTF AMP
CLOADF VOUTF 0 .4P

.MACRO AMP VIN+ VIN- VOUT
M1 2     VIN-   3       3       MOSN W=WM1 L=LM
M2 4     VIN+   3       3       MOSN W=WM1 L=LM
M3 2     2      VSUPPLY VSUPPLY MOSP W=WM1 L=LM
```

```
M4 4     2      VSUPPLY VSUPPLY MOSP W=WM1 L=LM
M5 VOUT  VBIAS  0       0       MOSN W=WM5 L=LM
M6 VOUT  4      VSUPPLY VSUPPLY MOSP W=WM6 L=LM
M7 3     VBIAS  0       0       MOSN W=WM7 L=LM

.ENDS

.PARAM AREA='4*WM1*LM + WM5*LM + WM6*LM + WM7*LM'
VX 1000 0 AREA
RX 1000 0 1K

.MODEL MOSP PMOS (VTO=-1 KP=2.4E-5 LAMBDA=.004
+ GAMMA =.37 TOX=3E-8 LEVEL=3)

.MODEL MOSN NMOS (VTO=1.2 KP=6.0E-5 LAMBDA=.0004
+ GAMMA =.37 TOX=3E-8 LEVEL=3)

.PARAM WM1=OPT1(60U,20U,100U)
+      WM5=OPT1(40U,20U,100U)
+      WM6=OPT1(300U,20U,500U)
+      WM7=OPT1(70U,40U,200U)
+       LM=OPT1(10U,2U,100U)
+     BIAS=OPT1(2.2,1.2,3.0)

.TRAN 2.5N 300N SWEEP OPTIMIZE=OPT1
+ RESULTS=DELAYR,DELAYF,TOT_POWER,AREA  MODEL=OPT
.MODEL OPT OPT CLOSE=100

.TRAN 2N 150N
.MEASURE DELAYR TRIG AT=0 TARG V(VOUTR) VAL=2.5 RISE=1
+ GOAL=100NS

.MEASURE DELAYF TRIG AT=0 TARG V(VOUTF) VAL=2.5 FALL=1
+ GOAL=100NS

.MEASURE TOT_POWER AVG POWER GOAL=10MW
.MEASURE AREA MIN PAR(AREA) GOAL=1E-9  MINVAL=100N
.PRINT V(VIN+) V(VOUTR) V(VOUTF)

.END
```

Optimization Results

```
RESIDUAL SUM OF SQUARES     =  4.654377E-04
NORM OF THE GRADIENT        =  6.782920E-02
```

Optimized Parameters Opt1

```
*      %NORM-SEN   %CHANGE
.PARAM WM1            =  47.9629U  $    1.6524  -762.3661M
.PARAM WM5            =  66.8831U  $   10.1048    23.4480M
.PARAM WM6            = 127.1928U  $   12.7991    22.7612M
.PARAM WM7            = 115.8941U  $    9.6104  -246.4540M
.PARAM LM             =   6.2588U  $   20.3279  -101.4044M
.PARAM BIAS           =   2.7180   $   45.5053     5.6001M

*** OPTIMIZE RESULTS MEASURE NAMES AND VALUES
* DELAYR           = 100.4231N
* DELAYF           =  99.5059N
* TOT_POWER        =  10.0131M
* AREA             =   3.1408N
```

Figure 13-30: CMOS Op-amp

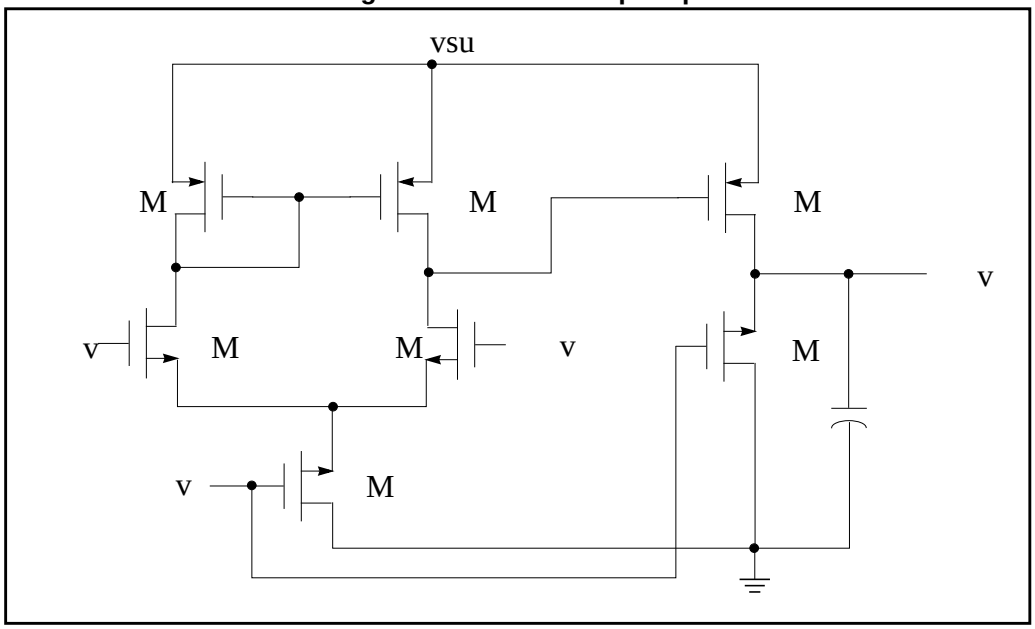
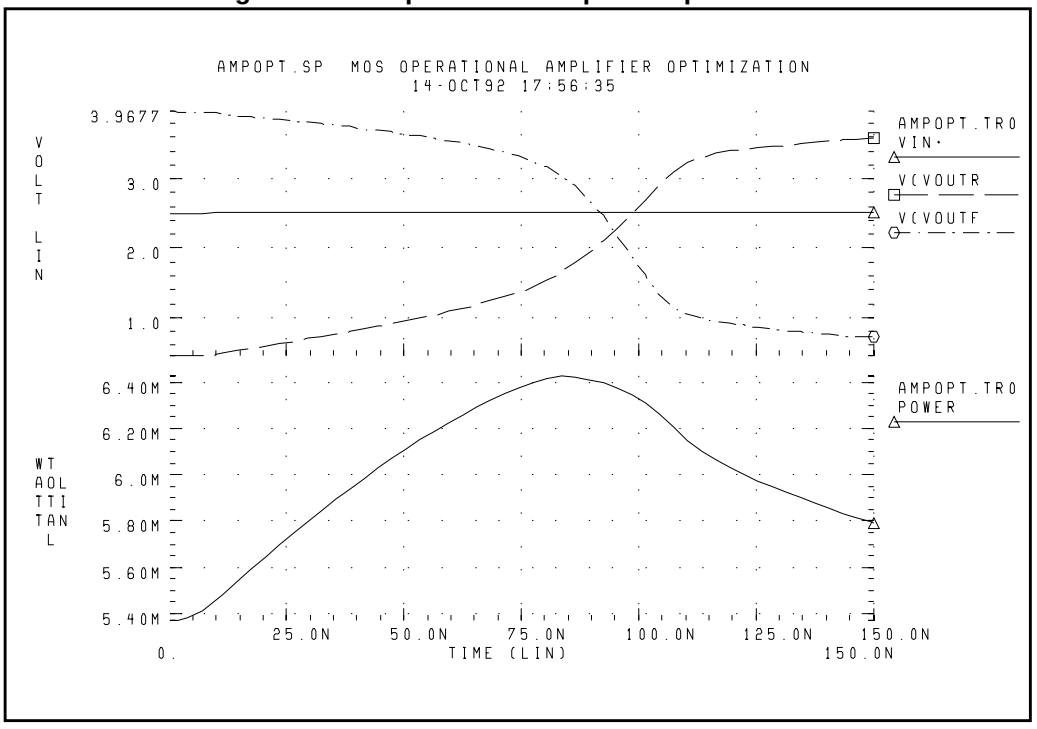


Figure 13-31: Operational Amplifier Optimization



Avant!

Chapter 14
Using the Common Model Interface

Avant!'s Common Model Interface (CMI) is a program interface for adding proprietary models into the Star-Hspice simulator.

This chapter covers the following topics:

Understanding CMI

Star-Hspice uses a dynamically linked shared library to integrate models with CMI. Add the global option *cmiflag*. It loads the dynamically linked CMI library, *libCMImodel*. Star-Hspice first searches for the shared library *libCMImodel* in the path *$hspice_lib_models.* If it does not find the library, it then searches the path in *$installdir/$ARCH/lib/models.*

Dynamic loading results in better resource sharing than static binding. While several Star-Hspice jobs are running concurrently, only one copy of the dynamically linked CMI model is needed in memory, which will speed up the process. Theoretically, the static linking version will always be slightly faster when only one Star-Hspice job is running at a time. However, the performance difference between dynamic loading and static binding is small, usually less than 5 percent.

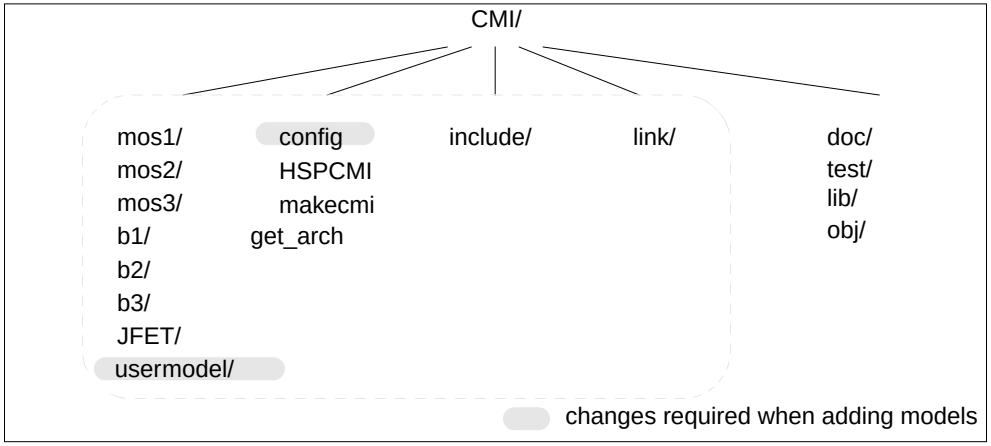
CMI is released with several source code examples for integration of MOS, JFET, and MESFET models in Star-Hspice. They are standard Berkeley SPICE MOSFET models (LEVEL 1, 2, 3, BSIM1, 2, 3) and JFET/MESFET models. To minimize the effort required for adding models, Avant! provides installation scripts to automate the shared library generation process.

If your proprietary models are derived from SPICE models, the integration process is similar to the examples with minimal modifications.

Note: The actual Star-Hspice program contains device equations and programs for bias calculation, numerical integration, convergence checking, and matrix loading. These programs are not needed to complete a new model integration and are excluded from the source code examples.

Examining the Directory Structure

The CMI distribution is structured as shown in the following diagram. You must modify the shaded files or add new ones for new models.



HSPCMI	Subdirectory containing utility to process configuration file and makefile
get_arch	C shell script for identifying platforms
config	Configuration file
doc/	CMI documentation
link/	Main CMI routines
include/	CMI header files
makecmi	Master makefile
test/	Model testing example
lib	Shared library directory
obj	Object code
mos1/, mos2/, mos3, b1,b2/,b3, JFET	Model directories

Running Simulations with CMI Models

CMI models are specified using model parameter *level*. Levels used by the example models are (same as those in Berkeley Spice-3):

- *LEVEL 1 (mos1/)* LEVEL 1 MOS model
- *LEVEL 2 (mos2/)* LEVEL 2 MOS model
- *LEVEL 3 (mos3/)* LEVEL 3 MOS model
- *LEVEL 4 (b1/)* BSIM model
- *LEVEL 5 (b2/)* BSIM2 model
- *LEVEL 8 (b3/)* BSIM3v3 model
- *LEVEL 9 (JFET)* JFET & MESFET model

To perform a simulation run on a CMI model, add the following line in the input netlist.

```
.option cmiflag
```

The LEVEL 8 example code located in the *b3* directory and Star-Hspice LEVEL 49 are both based on BSIM3, version 3. However, the speed of LEVEL 8 is sometimes 20 percent slower than LEVEL 49 in Star-Hspice. This occurs because LEVEL 49 in Star-Hspice is carefully implemented to ensure high accuracy and performance. In contrast, LEVEL 8 in the example code is created only as an example of CMI interface implementation. Therefore, the slower performance of the example code compared to Star-Hspice LEVEL 49 is expected.

The Level 9 example code is located in the JFET directory, and is based on Star-Hspice JFET&MESFET Level 3 (Statz model) and Spice3. This is example code, only for CMI interface implementation.

Supported Platforms

CMI supports the platforms:

- Sun SPARC: SunOS 4.1.3, Solaris 5.5X
- HP-7X: HP-UX 10.20
- RedHat Linux6.2, Linux7.0, Linux7.1

Adding Proprietary MOS Models

The CMI interface enables you to enter proprietary models into Star-Hspice. This section describes how to use CMI to add a new MOS model. Use CMI to simplify the integration process.

Note: In the following examples, the percent sign (%) represents the UNIX shell prompt, and $(installdir) points to the Star-Hspice release directory. $ARCH is the OS type for the computer. Star-Hspice 98.4 CMI release supports the platforms Sun4, Solaris, and HP.

Creating a CMI Shared Library

To add a new model:

1. Create a directory environment and modify the configuration file.
2. Prepare and modify model routines.
3. Compile the shared library.
4. Set up the runtime shared library path.

Creating the Directory Environment

To create the CMI directory environment:

1. Copy the CMI directory from the Star-Hspice release directory to a new location, as shown in the following example:

   ```
   % cp -r $(installdir)/cmi   /home/user1/userx/model
   ```

   The new CMI directory */home/user1/userx/model/cmi* is your working directory. Make sure that you have read and write access to your working directory.

2. Copy an existing model subdirectory to a new model directory and create a subdirectory for the new model under the working CMI directory. For example, if you have a MOSFET model whose LEVEL is 222, you can copy the subdirectory from the existing MOS model LEVEL 3, as follows:

```
%cp -r mos3 mos222
```

3. Add the following line in the configuration file *config*:

```
mos222      222      "my own MOSFET model"
```

where:

- *mos222* is the model name
- *222* is the model LEVEL;
- `"my own MOSFET model"` is the descriptive comment for the model.

The model name and level must be unique in the configuration file. For more information, see the in-line comment in the configuration file.

Preparing Model Routine Files

In the new model subdirectory *mos222*, rename *mos3* in all filenames to *mos222*, for example:

```
% mv CMImos3defs.h CMImos222defs.h
```

After renaming all the files, the new model subdirectory should contain the following group of files:

- CMImos222defs.h
- CMImos222.c
- CMImos222GetIpar.c
- CMImos222SetIpar.c
- CMImos222GetMpar.c
- CMImos222SetMpar.c
- CMImos222eval.c
- CMImos222set.c
- CMImos222temp.c

The purpose and detailed description of each routine can be found in Model Interface Routines on page 14-13. You must modify the functions as necessary. The majority of the work required to add a new model is for modification of these files.

Compiling the Shared Library

Follow the steps above to modify the model routine files and the configuration file, then compile the model routines and the shared library with a single *make* operation. Prior to the *make* operation, manually set the environment variable *HSPICE_CMI* to the working CMI directory (see Creating the Directory Environment on page 14-6):

```
% setenv HSPICE_CMI /home/user1/userx/model/cmi
```

With *HSPICE_CMI* set correctly, invoke the compilation process by entering the following:

```
% make -f makecmi
```

The new shared library called *libCMImodel* will be created in subdirectory *lib/* with all the object files generated in subdirectory *obj/*.

We recommend that you check the syntax of your C functions before launching the compilation process. Enter the following command:

```
% make -f makecmi lint
```

to list any syntax errors in your model routines.

Note: During compilation, CMI creates files (*makefile.SUN* on SUN, *makefile.HP* on HP), and subdirectories (*obj/*, *lib/*) in the CMI working directory. Do not manually modify these generated files.

Choosing a Compiler

You can use any functional compiler by properly setting the environment variable *CC* to the location of the compiler (*CC* is used in the auto-generated makefile, *makefile.SUN* and *makefile.HP).* Make sure that the appropriate compiler and link flags are properly set so the final CMI library build is in position independent code (PIC). PIC is needed for dynamic linking.

For SUNOS 5.4 platforms or later, the automatically generated compiler flag *-KPIC* and link flag *-G -z* are for the Sun workshop compiler, *cc* or *acc*. These compilers are typically installed in a directory such as */usr1/opt/SUNWspro/SC4.2/bin*. For HP9000/700 platforms, *cc* is installed in */opt/ansic/bin*. For more information, type `man cc` or `man acc` to display the on-line manual page for these commands.

There are no restrictions on the optimization flags you use for the CMI library. However, we recommend using the flag *-fast* for the Sun compiler *cc* or *acc* and the flag *-O* for HP compiler *cc*.

Using the gcc Compiler

If you use the *gcc* compiler, modify the makefile (*makefile.SUN* or *makefile.HP*) to set the compiler flags correctly.

For *gcc*, set the environment variable *CC* to *gcc* and modify the makefile to use the *-fPIC* flag for compiling and the *-r* flag for linking. For example:

```
gcc -c -I../include -fPIC CMImain.c
...
gcc -r -o ./lib/libCMImodel obj/*.o
```

Using the /usr/ucb/cc Compiler

If you use the */usr/ucb/cc* compiler, modify the makefile (*makefile.SUN* or *makefile.HP*) to set the compiler flags correctly.

The */usr/ucb/cc* compiler can not compile C source files until the “Language Optional Source Package” is installed. Verify that this source package is installed, then set the environment variable *CC* to */usr/ucb/cc*.

Note: The “Language Optional Source Package” is not installed in Solaris by default, but it is installed in SUNOS 4.1.x by default. You must either install the optional language software or use a workable compiler such as the Sun workshop cc or acc. The gcc compiler can also be used with some minor modifications to the makefile.

Setting up the Runtime Shared Library Path

The shared library is now ready for use. You must update the shared model search path defined by the environment variable *hspice_lib_models* so that the system dynamic loader can find the new CMI shared library for Star-Hspice. Enter the following:

```
setenv hspice_lib_models $HSPICE_CMI/lib
```

Troubleshooting

Sometimes you can successfully build the CMI dynamic library, but at run time Star-Hspice returns the following error message:

```
**error**: Unable to load
/home/ant/lib/models/libCMImodel and /home/ant/lib/
   models/libCMImodel.so
```

This problem is usually caused by undefined symbols at run time. The following is an example output of undefined symbols, using Sun workshop *cc* as a compiler on a SUNOS 5.5 machine (note that different compilers usually generate nm output in different formats)

Example

```
nm libCMImodel |grep UNDEF
[35]    |          0|       0|NOTY |LOCL |0    |UNDEF |
[34]    |          0|       0|NOTY |LOCL |0    |UNDEF |
[1779] |        0|      0|NOTY |GLOB |0    |UNDEF |.mul
[1853] |          0|       0|NOTY |GLOB |0    |UNDEF
   |__dtou
[1810] |       0|      0|NOTY |GLOB |0    |UNDEF |__iob
[1822] |          0|       0|NOTY |WEAK |0    |UNDEF
   |_ex_deregister
[1749] |          0|       0|NOTY |WEAK |0    |UNDEF
   |_ex_register
[1857] |        0|      0|NOTY |GLOB |0    |UNDEF |atan
[1878] |         0|      0|NOTY |GLOB |0    |UNDEF |cos
[1820] |         0|      0|NOTY |GLOB |0    |UNDEF |exp
[1777] |        0|      0|NOTY |GLOB |0    |UNDEF |fabs
[1872] |          0|       0|NOTY |GLOB |0    |UNDEF
   |fprintf
```

```
[1764] |         0|       0|NOTY |GLOB |0    |UNDEF |log
[1767] |          0|       0|NOTY |GLOB |0    |UNDEF
   |malloc
[1842] |          0|       0|NOTY |GLOB |0    |UNDEF
    |memset
[1772] |         0|       0|NOTY |GLOB |0    |UNDEF |pow
[1869] |         0|       0|NOTY |GLOB |0    |UNDEF |sin
[1790] |        0|       0|NOTY |GLOB |0    |UNDEF |sqrt
[1758] |          0|       0|NOTY |GLOB |0    |UNDEF
   |strcasecmp
[1848] |          0|       0|NOTY |GLOB |0    |UNDEF
   |strcpy
[1858] |          0|       0|NOTY |GLOB |0    |UNDEF
   |strlen
[1800] |          0|       0|NOTY |GLOB |0    |UNDEF
   |strncpy
```

The above undefined symbols can be satisfied at run time by libraries such as *libc* or *libm*. However, any unsatisfied symbols other than those shown in the example may cause the “Unable to load” problem.

Testing CMI Models

After creating the shared library, you may test the new model by running Star-Hspice on a sample input file *mos3.sp* under the subdirectory *test*. This file contains a simple CMOS inverter using MOS LEVEL-3 models. Modify transistor sizes and model cards as necessary.

```
%hspice mos3.sp >mos3.lis
```

Avanwaves can now be used to inspect the I-V and C-V characteristics at different biasing conditions.

Use AvanWaves to carefully check the following aspects:

- Sign and value of channel current (*ids)*
- The monotone characteristics of channel current versus *vgs* and *vds*
- Sign and value of capacitance (*cgs, cgs, cgb, csb, cdb*)

Refer to the *AvanWaves User Guide* for more information.

To verify the CMI integration of your new model, run a DC sweep analysis and transient analysis on the test netlist.

Note: LEVELs from 100 to 200 are reserved for CMI customer models. Please choose levels from this range so there will be no conflicts with existing Star-Hspice model levels. Also, please add a special prefix or suffix for some of the auxiliary functions used in CMI, especially those from the public domain, such as the function called modchk or dc3p1 from Berkeley Spice3. This will ensure that the function names are different from those used in the Star-Hspice core code.

After testing, if you are satisfied with your CMI library, put it in the default CMI library directory, *$installdir/$ARCH/lib/models*, where *$ARCH* is *sun4*, *sol4*, or *pa*, depending on the platform on which you compiled your CMI library.

The model interface routines accept input parameters from CMI. For each set of input conditions, the model routines are required to return transistor characteristics to CMI.

Model Interface Routines

Model interface routines accept input parameters from CMI. These input parameters include the following:

- Circuit and nominal model temperatures (*CKTtemp, CKTnomtemp*)
- Input biases (*vds, vgs, vbs*)
- Model parameters (*level, vto, tox, uo* ...)
- Instance parameters (*w, l, as, ad* ....)
- Mode of transistor (*mode*, 1 for normal, -1 for reverse)
- AC frequency (*freq*, this is passed from simulator to model code)
- Transient simulation integration order (*intorder*) (Star-Hspice returns the following codes: 0 - Trapezoidal, 1 - 1st order Gear, 2- 2nd order Gear)
- Transient time step (*timestep*)
- Transient time point (*timepoint*)

For each set of input conditions, the model routines are required to return the following transistor characteristics to CMI:

- Charge and capacitance computation flag (1 for computation, 0 for no computation, *qflag*)
- Charge, capacitance model selector (0 for Meyer capacitance model*, 13 for charge-based model, *capop*)
- Channel current (*ids*)
- Channel conductance (*gds*)
- Trans-conductance (*gm*)
- Substrate trans-conductance (*gmbs*)
- Turn-on voltage (*von)*
- Saturation voltage (*vsat)*
- Gate overlap capacitances (*cgso, cgdo, cgbo*)
- Intrinsic MOSFET charges (*qg, qd, qs*)
- Intrinsic MOSFET capacitances referenced to bulk (*cggb, cgdb, cgsb, cbgb, cbdb, cbsb, cdgb, cddb, cdsb*)
- Parasitic source & drain conductances (*gs, gd*)
- Substrate diode current (*ibd, ibs*)
- Substrate diode conductance (*gbd, gbs)*

- Substrate diode charge (*qbd, qbs)*
- Substrate diode junction capacitance (*capbd, capbs*)
- Substrate impact ionization current (*isub*)
- Substrate impact ionization trans-conductances *(gbgs=dIsub/dVgs, gbds=dIsub/dVds, gbbs=dIsub/dVbs*)
- Source resistance noise current squared (*nois_irs*)
- Drain resistance noise current squared (*nois_ird*)
- Thermal or Shot channel noise current squared (*nois_idsth*)
- Source resistance noise current squared (*nois_idsfl*)

Note: Currently, the Meyer capacitance model is not supported in Star-Hspice.

The transistor biases and output characteristics are transferred between CMI and model interface routines using variable type *CMI_VAR,* which can be found in the *include/CMIdef.h* file.

The entries *vds, vgs and vbs* are used to provide bias conditions, while the rest of the entries carry the results from evaluating the model equations.

```
/* must be consistent with its counterpart in HSPICE */
typedef struct CMI_var {

/* device input formation */
   int    mode;       /* device mode */
   int   qflag;     /* flag for charge/cap computing */
   double vds;        /* vds bias */
   double vgs;        /* vgs bias */
   double vbs;        /* vbs bias */

/* device DC information */
   double gd;         /* drain conductance */
   double gs;         /* source conductance */
   double cgso; /* gate-source overlap capacitance */
   double cgdo; /* gate-drain overlap capacitance */
   double cgbo;     /* gate-bulk overlap capacitance */
   double von;        /* turn-on voltage */
   double vdsat;      /* saturation voltage */
   double ids;        /* drain dc current */
   double gds; /* output conductance (dIds/dVds) */
   double gm;       /* trans-conductance (dIds/dVgs) */
```

```
      double gmbs; /* substrate trans-conductance
           (dIds/dVbs)*/
    ‘  /* MOSFET capacitance model selection */
      /* capop can have following values
      * 13          charge model
      * 0 or else   Meyer's model
```

Note: Currently, Meyer’s model is not supported.

```
   */
      int    capop;      /* capacitor selector */
/* Meyer's capacitances: intrinsic capacitance + overlap
capacitance Note: currently, these 3 capacitances are
ignored by Hspice. A charge-based model formulation is
required. */
double capgs; /* Meyer's gate capacitance
      (dQg/dVgs + cgso) */
double capgd; /* Meyer's gate capacitance
      (dQg/dVds + cgdo) */
double capgb; /* Meyer's gate capacitance
   (dQg/dVbs + cgbo) */
/* substrate-junction information */
double ibs;   /* substrate source-junction leakage
   current */
double ibd;   /* substrate drain-junction leakage
   current */
double gbs; /* substrate source junction-conductance */
double gbd;   /* substrate drain junction-conductance */
double capbs; /* substrate source-junction capacitance */
double capbd; /* substrate drain-junction capacitance */
double qbs;   /* substrate source-junction charge */
double qbd;   /* substrate drain-junction charge */
/* substrate impact ionization current */
double isub; /* substrate current */
double gbgs; /* substrate trans-conductance
      (dIsub/dVgs) */
```

```
double gbds; /* substrate trans-conductance
      (dIsub/dVds) */

double gbbs; /* substrate trans-conductance
      (dIsub/dVbs) */

/* charge-based model intrinsic terminal charges */
/* NOTE: these are intrinsic charges ONLY */
double qg;         /* gate charge */
double qd;         /* drain charge */
double qs;         /* source charge */

/* charge-based model intrinsic trans-capacitances*/
/* NOTE: these are intrinsic capacitances ONLY */

      double cggb;
      double cgdb;
      double cgsb;
      double cbgb;
      double cbdb;
      double cbsb;
      double cdgb;
      double cddb;
      double cdsb;

/* noise parameters */
      double nois_irs; /* Source noise current^2 */
      double nois_ird; /* Drain noise current^2 */
      double nois_idsth;/* channel thermal or shot
                 noise current^2 */
      double nois_idsfl; /* 1/f channel noise current^2 *
      double freq;           /* ac frequency */

/* extended model topology */
      char *topovar;  /* topology variables */
      double leff;    /* effective channel length */
      double weff;    /* effective channel width */
                      } CMI_VAR;
```

The nominal temperature and device temperature can be found in the global variable *CMIenv*. The CMIenv structure can be accessed via the global variable *pCMIenv* (pointer to the global CMIenv struct).

The structure for *CMIenv* is defined in type *CMI_ENV*, which can be found in the *include/CMIdef.h* file:

```
/* environment variables */
typedef struct CMI_env {
      double CKTtemp;     /* simulation temperature */
      double CKTnomTemp; /* nominal temperature */
      double CKTgmin;     /* GMIN for the circuit *
      int    CKTtempGiven;/* temp setting flag */
      /* following are hspice-specific options */
      double aspec;
      double spice;
      double scalm;
                } CMI_ENV;

/* model parameters for JFET&MESFET */
/* JFET&MESFET model parameter for CMI_VAR in
      CMIdef.h */

double gg;/* gate conductance */
double cigs;              /* gate-to-source current */
double gigs;              /* gate-to-source conductance */
double cigd;              /* gate-to-drain current */
double gigd;              /* gate-to-drain conductance */
double csat;              /* diode saturation current */
double capds;           /* drain-to-source capacitance */
double nois_irg;        /* Gate noise current^2 */
double qgso;            /* gate-to-source old charge */
double qgdo;            /* gate-to-drain old charge */
double qgs;              /* gate-to-source charge */
double qgd;              /* gate-to-drain charge */
double vgsold;          /* gate-to-source old voltage */
double vgdold;          /* gate-to-drain old voltage */
```

Interface Variables

To assign model/instance parameter values and evaluate I-V, C-V response, fifteen interface routines are required. For each new model, pointers to these routines and the model/instance variables are defined by an *interface variable* in type *CMI_MOSDEF*, which can be found in the *include/CMIdef.h* file.

```
typedef struct CMI_MosDef {
char  ModelName[100];
char  InstanceName[100];
char  *pModel;
char  *pInstance;
int   modelSize;
int   instSize;
int   (*CMI_ResetModel)(char*,int,int);
int   (*CMI_ResetInstance)(char*);
int (*CMI_AssignModelParm)(char*,char*,double);
int (*CMI_AssignInstanceParm)(char*,char*,double);
int   (*CMI_SetupModel)(char*);
int   (*CMI_SetupInstance)(char*,char*);
int   (*CMI_Evaluate)(CMI_VAR*,char*,char*);
int   (*CMI_DiodeEval)(CMI_VAR*,char*,char*);
int   (*CMI_Noise)(CMI_VAR *,char*,char*);
int   (*CMI_PrintModel)(char*);
int   (*CMI_FreeModel)(char*);
int   (*CMI_FreeInstance)(char*,char*);
int   (*CMI_WriteError)(int, char*);
int   (*CMI_Start)(void);
int   (*CMI_Conclude)(void);

/* extended model topology,0 is normal mos, 1 is
     berkeley SOI, etc. */
int     topoid;
} CMI_MOSDEF;
```

All routines return 0 on success, or nonzero integer (a user-defined error code) in case of warning or error. The following sections describe each entry. Examples for the first seven functions are extracted from the MOS3 implementation. Examples for the remaining eight functions are not part of the actual MOS3 code, but are included for demonstration. The example MOS3 implementation contains one header file and eight C files. All routines are derived from SPICE-3 code.

pModel, pInstance

Structure entries of the interface variable are initialized at compile time.

Example

```
/* function declaration */
int CMImos3ResetModel(char*,int,int);
int CMImos3ResetInstance(char*);
int CMImos3AssignMP(char*,char*,double);
int CMImos3AssignIP(char*,char*,double);
int CMImos3SetupModel(char*);
int CMImos3SetupInstance(char*,char*);
int CMImos3Evaluate(CMI_VAR*,char*,char*);
int CMImos3DiodeEval(CMI_VAR*,char*,char*);
int CMImos3Noise(CMI_VAR *,char*,char*);
int CMImos3PrintModel(char*);
int CMImos3FreeModel(char*);
int CMImos3FreeInstance(char*,char*);
int CMImos3WriteError(int, char*);
int CMImos3Start(void);
int CMImos3Conclude(void);
/* extended model topology, 0=normal mos, 1=berkeley SOI */
/* local */
static MOS3model    _Mos3Model;
static MOS3instance _Mos3Instance;
static CMI_MOSDEF CMI_mos3def = {
       (char*)&_Mos3Model,
       (char*)&_Mos3Instance,
       CMImos3ResetModel,
       CMImos3ResetInstance,
       CMImos3AssignMP,
       CMImos3AssignIP,
       CMImos3SetupModel,
       CMImos3SetupInstance,
       CMImos3Evaluate,
       CMImos3DiodeEval,
       CMImos3Noise,
       CMImos3PrintModel,
       CMImos3FreeModel,
       CMImos3FreeInstance,
       CMImos3,
       CMImos3Start,
       CMImos3Conclude
};
```

```
/* export */
CMI_MOSDEF *pCMI_mos3def = &CMI_mos3def;
```

*Note: the last 8 functions are optional. Any function not defined should be replaced with NULL.

CMI_ResetModel

This routine initializes all parameters of a model. All model parameters become undefined after initialization. Undefined means “not defined in a netlist model card”. The flag *pmos* is used to set transistor type after initialization.

```
int CMI_ResetModel(char* pmodel, int pmos, int level)
```

pmodel	Pointer to the model instance
pmos	1 if PMOS, 0 if NMOS
level	model level value passed from parser

Example

```
int
#ifdef __STDC__
   CMImos3ResetModel(
   char *pmodel,
   int   pmos)
#else
   CMImos3ResetModel(pmodel,pmos)
   char *pmodel;
   int   pmos;
#endif
   {
   /* reset all flags to undefined */
   (void)memset(pmodel, 0, sizeof(MOS3model));
/* Note: contains model value passed from parser */
   if(pmos)
   {
      ((MOS3model*)pmodel)->MOS3type = PMOS;
      ((MOS3model*)pmodel)->MOS3typeGiven = 1;
   }
   return 0;
   } /* int CMImos3ResetModel() */
```

CMI_ResetInstance

This routine initializes all parameter settings of an instance. All instance parameters should be undefined after initialization. “Undefined” means “not given in a netlist MOS instance”.

```
int CMI_ResetInstance(char* pinst)
```

pinst	Pointer to the instance

Example

```
int
#ifdef __STDC__
   CMImos3ResetInstance(
   char *ptran)
#else
   CMImos3ResetInstance(ptran)
   char *ptran;
#endif
   {
   (void)memset(ptran, 0, sizeof(MOS3instance));
   ((MOS3instance*)ptran)->MOS3w = 1.0e-4;
   ((MOS3instance*)ptran)->MOS3l = 1.0e-4;
   return 0;
   } /* int CMImos3ResetInstance() */
```

CMI_AssignModelParm

This routine sets the value of a model parameter.

```
int CMI_AssignModelParm(char* pmodel, char*
pname,double value)
```

pmodel	Pointer to the model instance
pname	String of parameter name
value	Parameter value

Example

```
int
#ifdef __STDC__
   CMImos3AssignMP(
   char   *pmodel,
   char   *pname,
   double value)
#else
   CMImos3AssignMP(pmodel,pname,value)
   char   *pmodel;
   char   *pname;
   double value;
#endif
   {
   int param;
   CMImos3GetMpar(pname, &param);
   CMImos3SetMpar(param, value, (MOS3model*)pmodel);
   return 0;
   } /* int CMImos3AssignMP() */
```

CMI_AssignInstanceParm

This routine sets the value of an instance parameter.

```
int CMI_AssignInstanceParm(char *pinst,char*
pname,double value)
```

pinst	Pointer to the instance
pname	String of parameter name
value	Parameter value

Example

```
int
#ifdef __STDC__
   CMImos3AssignIP(
   char   *ptran,
   char   *pname,
   double value)
#else
   CMImos3AssignIP(ptran,pname,value)
   char   *ptran;
   char   *pname;
   double value;
#endif
   {
   int param;
   CMImos3GetIpar(pname, &param);
   CMImos3SetIpar(param, value, (MOS3instance*)ptran);
   return 0;
   } /* int CMImos3AssignIP() */
```

CMI_SetupModel

This routine sets up a model after all the model parameters are specified.

```
int CMI_SetupModel(char* pmodel)
```

pmodel	Pointer to the model

Example

```
int
#ifdef __STDC__
   CMImos3SetupModel(
   char *pmodel)
#else
   CMImos3SetupModel(pmodel)
   char *pmodel;
#endif
   {
   CMImos3setupModel((MOS3model*)pmodel);
   return 0;
   } /* int CMImos3SetupModel() */
```

CMI_SetupInstance

This routine sets up an instance after all the instance parameters are specified. Typically temperature and geometry processing are performed here.

```
int CMI_SetupInstance(char* pinst)
```

pinst	Pointer to the instance

Example

```
int
#ifdef __STDC__
  CMImos3SetupInstance(
  char    *pmodel,
  char    *ptran)
#else
  CMImos3SetupInstance(pmodel,ptran)
  char    *pmodel;
  char    *ptran;
#endif
  {
  /* temperature modified parameters */
CMImos3temp((MOS3model*)pmodel,(MOS3instance*)ptran);
  return 0;
  } /* int CMImos3SetupInstance() */
```

CMI_Evaluate

Based on the bias conditions and model/instance parameter values, this routine evaluates the model equations and passes all transistor characteristics via the *CMI_VAR* variable.

```
int CMI_Evaluate(CMI_VAR *pvar,char *pmodel. char
*pinst)
```

pvar	Pointer to *CMI_VAR* variable.
pmodel	Pointer to the model.
pinst	Pointer to the instance.

Example

```
int
#ifdef __STDC__
  CMImos3Evaluate(
  CMI_VAR *pslot,
  char    *pmodel,
  char    *ptr)
#else
  CMImos3Evaluate(pslot,pmodel,ptr)
  CMI_VAR *pslot;
  char    *pmodel;
  char    *ptr;
#endif
  {
  CMI_ENV      *penv;
  MOS3instance *ptran;
  penv = pCMIenv; /* pCMIenv is a global */
  ptran = (MOS3instance*)ptr;
  /* call model evaluation */

(void)CMImos3evaluate(penv,(MOS3model*)pmodel,ptran,
pslot->vgs,pslot->vds,pslot->vbs);
pslot->gd = ptran->MOS3drainConductance;
pslot->gs = ptran->MOS3sourceConductance;
pslot->von    = ptran->MOS3von;
pslot->ids    = ptran->MOS3cd;
pslot->gds    = ptran->MOS3gds;
pslot->gm     = ptran->MOS3gm;
pslot->gmbs   = ptran->MOS3gmbs;
pslot->gbd    = ptran->MOS3gbd;
pslot->gbs    = ptran->MOS3gbs;
pslot->cgs    = ptran->MOS3capgs;
pslot->cgd    = ptran->MOS3capgd;
pslot->cgb    = ptran->MOS3capgb;
pslot->capdb  = ptran->MOS3capbd;
pslot->capsb  = ptran->MOS3capbs;
pslot->cbso   = ptran->MOS3cbs;
pslot->cbdo   = ptran->MOS3cbd;

...Additional CMI_VAR elements should be assigned here
for substrate model and overlap capacitances.

return 0;
} /* int CMImos3Evaluate() */
```

CMI_DiodeEval

Based on the bias conditions and model/instance parameter values, this routine evaluates the MOS junction diode model equations, and passes all transistor characteristics via the *CMI_VAR* variable.

```
int CMI_DiodeEval(CMI_VAR *pvar,char *pmodel. char *pinst)
```

pvar	Pointer to *CMI_VAR* variable
pmodel	Pointer to the model
pinst	Pointer to the instance

Example

```
int
#ifdef __STDC__
   CMImos3DiodeEval(
   CMI_VAR *pslot,
   char    *pmodel,
   char    *ptr)
#else
   CMImos3Diode(pslot,pmodel,ptr)
   CMI_VAR *pslot;
   char    *pmodel;
   char    *ptr;
#endif
   {
   CMI_ENV      *penv;
   MOS3instance *ptran;
   penv = pCMIenv; /* pCMIenv is global */
   ptran = (MOS3instance*)ptr;
   /* call model evaluation */
   (void)CMImos3diode(penv,(MOS3model*)pmodel,ptran,
   pslot->vgs,pslot->vds,pslot->vbs);
   pslot->ibs = ptran->MOS3ibs;
   pslot->ibd = ptran->MOS3ibd;
   pslot->gbs = ptran->MOS3gbs;
   pslot->gbd = ptran->MOS3gbd;
   pslot->capbs = ptran->MOS3capbs;
   pslot->capbd = ptran->MOS3capbd;
   pslot->qbs = ptran->MOS3qbs;
   pslot->qdb = ptran->MOS3qbd;
return 0;
   } /* int CMImos3DiodeEval() */
```

CMI_Noise

Based on the bias conditions, temperature, and model/instance parameter values, this routine evaluates the noise model equations and returns noise characteristics via the *CMI_VAR* variable.

The value passed to *pslot->nois_irs* should be the thermal noise associated with the parasitic source resistance expressed as a mean square noise current in parallel with Rs.

The value passed to *pslot->nois_ird* should be the thermal noise associated with the parasitic drain resistance expressed as a mean square noise current in parallel with Rd.

The value passed to *pslot->nois_idsth* should be the thermal noise associated with the MOSFET expressed as a mean square noise current referenced across the MOSFET channel.

The value passed to *pslot->nois_idsfl* should be the flicker noise associated with the MOSFET expressed as a mean square noise current referenced across the MOSFET channel. Frequency is passed into CMI_Noise via *pslot->freq.*

```
int CMI_Noise(CMI_VAR *pvar,char *pmodel. char *pinst)
```

pvar	Pointer to *CMI_VAR* variable
pmodel	Pointer to the model
pinst	Pointer to the instance

Example

```
int
#ifdef __STDC__
   CMImos3Noise(
   CMI_VAR *pslot,
   char    *pmodel,
   char    *ptr)
#else
   CMImos3Noise(pslot,pmodel,ptr)
   CMI_VAR *pslot;
   char    *pmodel;
   char    *ptr;
#endif
```

```
   {double freq,fourkt;
   CMI_ENV      *penv
MOS3instance *ptran;
   penv = pCMIenv; /* pCMIenv is a global */
   ptran = (MOS3instance*)ptr;
   fourkt = 4.0 * BOLTZMAN * ptran->temp; /* 4kT */
   freq = pslot->freq;

/* Drain resistor thermal noise as current^2 source*/
pslot->nois_ird = fourkt * ptran->gdpr;

/* Source resistor thermal noise as current^2 source */
pslot->nois_irs = fourkt * ptran->gspr;

/* thermal noise assumed to be current^2 source
referenced to channel. The source code for
thermalnoise() is not shown here*/

pslot->nois_idsth = thermalnoise(model, here, fourkt);

/* flicker (1/f) noise assumed to be current^2 source
referenced to channel. The source code for
flickernoise() is not shown here */

pslot->nois_idsfl = flickernoise(model, here, freq);

return 0;
} /* int CMImos3Noise() */
```

CMI_PrintModel

This routine prints all model parameter names, values and units to standard output and is called by Star-Hspice for each model after CMI_SetupModel.

```
int CMI_PrintModel(char *pmodel)
```

pmodel	Pointer to the model

Example

```
int
#ifdef __STDC__
   CMImos3PrintModel(
   char    *pmodel)
#else
   CMImos3PrintModel(pmodel)
   char    *pmodel;
```

```
#endif
  {
  CMI_ENV      *penv
/* Note: source for CMImos3printmodel() not shown*/
  (void)CMImos3printmodel((MOS3model*)pmodel);
  return 0;
  } /* int CMImos3PrintModel() */
```

CMI_FreeModel

This routine allows the user to free memory that previously was allocated for model related data. This routine is called during a loop over all models at post-simulation time.

```
int CMI_FreeModel(char *pmodel)
```

pmodel	Pointer to the model

Example

```
int
#ifdef __STDC__
  CMImos3FreeModel(
  char    *pmodel)
#else
  CMImos3FreeModel(pmodel)
  char    *pmodel;
#endif
  { /* free memory allocated for model data. Note
  CMImos3freemodel() source code not shown. */
(void)CMImos3freemodel((MOS3model*)pmodel);
  return 0;
  } /* int CMImos3FreeModel() */
```

CMI_FreeInstance

This routine allows the user to free memory that was previously allocated for storing instance-related data. This routine is called during an outer loop over all models and an inner loop over all instances associated with each model at post-simulation time.

```
int CMI_FreeInstance(char *pmodel. char *pinst)
```

pmodel	Pointer to the model
pinst	Pointer to the instance

Example

```
int
#ifdef __STDC__
   CMImos3FreeInstance(
   char    *pmodel,
   char    *ptr)
#else
   CMImos3FreeInstance(pmodel,ptr)
   char    *pmodel;
   char    *ptr;
#endif
   {
   CMI_ENV      *penv
   MOS3instance *ptran;
   ptran = (MOS3instance*)ptr;
/* free memory allocated for model data. Note
CMImos3freeinstance()source code not shown.
*/

(void)CMImos3freeinstance((MOS3model*)pmodel,ptran);
   return 0;
   } /* int CMImos3FreeInstance() */
```

CMI_WriteError

This routine writes user-defined error messages to standard output, when model evaluation detects an error. All CMI functions return an error code and pass it to CMI_WriteError(). In CMI_WriteError(), you define an error statement, and copy it to err_str (based on the error code value). CMI_WriteError() returns the error status: if err_status>0, Star-Hspice writes the error message and aborts. If err_status=0, Star-Hspice writes the warning message and continues. Star-Hspice calls CMI_WriteError() after every CMI function call.

```
int CMI_WriteError(int err_code, char *err_str)
```

err_code	Error code
err_str	Pointer to error message

Example

```
int
#ifdef __STDC__
  CMImos3WriteError(
  void err_code,
  char *err_str)
#else
  CMImos3WriteError(err_code,err_str)
  int err_code;
  char *err_str;
#endif
  {
/* */
int err_status=0;
switch err_code
  {
case 1:
  strcpyn(err_str,”User Err: Eval()”,CMI_ERR_STR_LEN);
  err_status=1;
case 2:
  strcpyn(err_str,”User Warn: Eval()”,CMI_ERR_STR_LEN);
  err_status=1;
default:
  strcpyn(err_str,”User Err:Generic”,CMI_ERR_STR_LEN);
  err_status=1;
return err_status;
  } /* int CMImos3WriteError() */
```

CMI_Start

This routine allows user-defined startup functions to be run once prior to simulation.

```
int CMI_Start(void)
```

Example

```
int
#ifdef __STDC__
  CMImos3Start(void)
#else
  CMImos3Start(void)
#endif
  {
(void)CMImos3start();
  return 0;
  } /* int CMImos3Start() */
```

CMI_Conclude

This routine allows user-defined conclude functions to be run once at post-simulation time.

```
int CMI_Conclude(void)
```

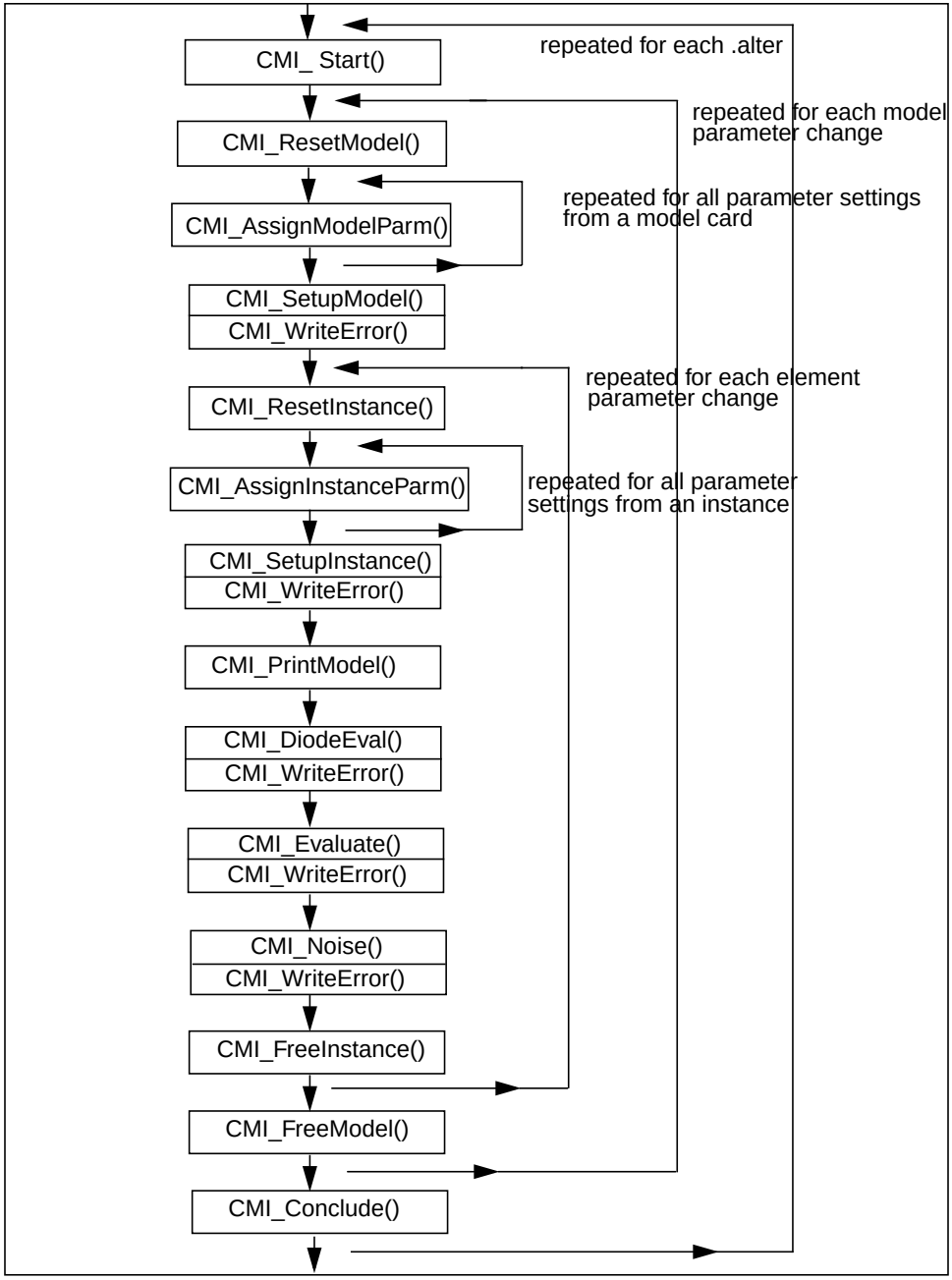
Example

```
int
#ifdef __STDC__
  CMImos3Conclude(void)
#else
  CMImos3Conclude(void)
#endif
  {
(void)CMImos3conclude();
  return 0;
  } /* int CMImos3Conclude() */Internal Routines */
```

CMI Function Calling Protocol

CMI calls the interface routines in the sequence as shown in Figure 14-1 .

Figure 14-1: Interface Routines Calling Sequence



Internal Routines

In the example MOS3 implementation, the interface routines in *CMImos3.c* also call the following internal routines:

CMImos3GetIpar.c	get instance parameter index
CMImos3SetIpar.c	set instance parameter
CMImos3GetMpar.c	get model parameter index
CMImos3SetMpar.c	set model parameter
CMImos3eval.c	evaluate model equations
CMImos3set.c	setup a model
CMImos3temp.c	setup an instance including temperature-effect

Figure 14-2 illustrates the hierarchical relationship between the interface routines and internal routines.

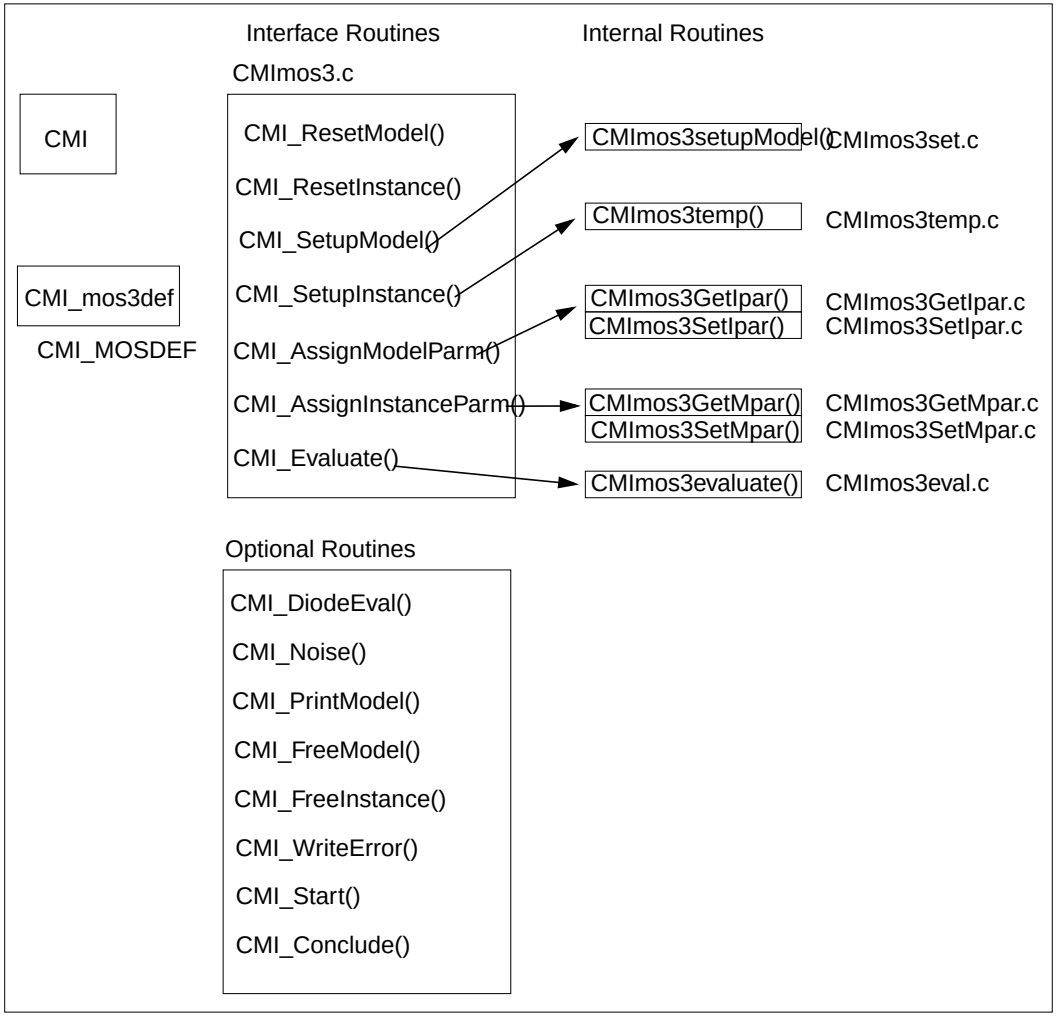
Note: For the automatic script to work, the name of the interface variable and all routine files must follow the naming convention, as follows:

pCMI_xxxdef

CMIxxx.c	CMIxxxGetIpar.c
CMIxxxSetIpar.c	CMIxxxGetMpar.c
CMIxxxSetMpar.c	CMIxxxeval.c

where *xxx* is the model name.

Figure 14-2: Hierarchy of Interface and Internal Routines



Supporting Extended Topology

In addition to conventional four terminal (topoid = 0) MOSFET topology, Star-Hspice can support other topologies. You must assign a unique topoid for different topologies.

BSIM SOI topology has been implemented in CMI and assigned a topoid of 1. If your own model *topovar* is the same as BSIM SOI, you can specify a topoid of 1 and use the Star-Hspice topology structure for stamping information. If your model topology is different from the conventional four-terminal model or the BSIM SOI, then you have to give Star-Hspice the topovar structure and Star-Hspice will assign a unique topoid for your topology.

For example, the following is the topovar structure for the BSIM SOI used in Star-Hspice CMI. The naming convention for the structure fields is the same as in BSIM SOI. For detailed information about fields in the structure, please refer to *"BSIM3PD2.0 MOSFET MODEL User' Manual,"* which can be found at "http://www-device.eecs.berkeley.edu/~bsim3soi".

```
struct TOPO1 {
  double vps;
  double ves;
  double delTemp;      /* T node */
  double selfheat;

  double qsub;
  double qth;
  double cbodcon;

  double gbps;
  double gbpr;
  double gcde;
  double gcse;

  double gjdg;
  double gjdd;
  double gjdb;
  double gjdT;
  double gjsg;
  double gjsd;
  double gjsb;
  double gjsT;
```

```
    double cdeb;
    double cbeb;
    double ceeb;
    double cgeo;

  /* add 4 for T */
    double cgT;
    double cdT;
    double cbT;
    double ceT;

    double rth;
    double cth;

    double gmT;
    double gbT;
    double gbpT;
    double gTtg;
    double gTtd;
    double gTtb;
    double gTtt;

  };
```

Conventions

Bias Polarity Conventions for N- and P-channel Devices

The input biases *vds*, *vgs*, and *vbs* in *CMI_VAR* are defined as:

```
vds = vd - vs
vgs = vg - vs
vbs = vb - vs
```

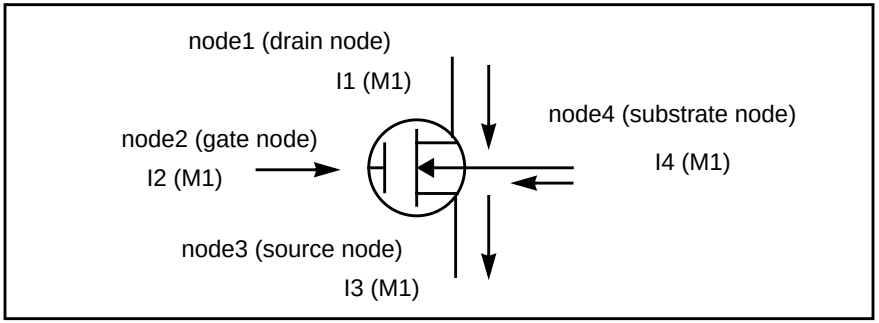
Negation of these biases for the P-channel device is required if your model code does not distinguish between n-channel and p-channel bias. In the example routines, the biases are multiplied by the model parameter “type”, which is 1 for N-device and -1 for P-device. See for example the MOS3 model code:

```
if (model->MOS3type < 0) { /* P-channel */
   vgs = -VgsExt;
   vds = -VdsExt;
   vbs = -VbsExt;
   }
else { /* N-channel */
   vgs = VgsExt;
   vds = VdsExt;
   vbs = VbsExt;
}
```

This code should be used in both the CMI_Evaluate() and CMI_DiodeEval() functions.

The convention for outputting current components is shown in Figure 14-3 . For channel current, drain-to-source is considered the positive direction. For substrate diodes, bulk-to-source/drain are considered the positive directions. The conventions are the same for both N-channel and P-channel devices.

Figure 14-3: MOSFET (node1, node2, node3, node4) - N-channel



The conventions for *von* are:

- N-channel, device is on if vgs > von
- P-channel, device is on if vgs < von

Derivatives (conductances and capacitances) should be provided based on the polarity conventions of the bias and current. The following code demonstrates the required polarity reversal for currents and *Von, Vdsat* for PMOS devices.

```
if (model->type < 0)
   {
   pslot->ids = -pslot->ids;
   pslot->ibs = -pslot->ibs;
   pslot->ibd = -pslot->ibd;
   pslot->von = -pslot->von;
   pslot->vdsat = -pslot->vdsat;
}
```

Source-Drain Reversal Conventions

Star-Hspice performs the appropriate computations when the MOSFET is operated in the reverse mode (i.e., when Vds < 0 for N-channel, Vds > 0 for P-channel). This includes a variable transformation (Vds -> -Vds, Vgs -> Vgd, Vbs->Vbd) and interchange of the source and drain terminals. This transformation is transparent to the model developer and simplifies the model coding task.

Thread-Safe Model Code

Star-Hspice uses shared-memory, multithreading algorithms during model evaluation. To ensure thread-safe model code, the following rules must be strictly adhered to:

- Do not use static variables in CMI_Evaluate(), CMIDiodeEval(), CMIWriteError(), and CMI_Noise() or in functions called by these routines.
- Never write to a global variable during execution of CMI_Evaluate(), CMIDiodeEval(), CMIWriteError(), and CMI_Noise().

Avant!

Chapter 15

Performing Cell Characterization

Most ASIC vendors use Star-Hspice to characterize their standard cell libraries and prepare data sheets by using the basic capabilities of the .MEASURE statement. Input sweep parameters and the resulting measure output parameters are stored in the measure output data files *design.mt0*, *design.sw0*, and *design.ac0*. Multiple sweep data is stored in this file, and you can plot it by using AvanWaves. This lends itself to generating fanout plots of delay versus load. The slope and intercept of the loading curves can be used to calibrate VHDL, Verilog, Lsim, TimeMill, and Synopsys models.

This chapter covers:

- Determining Typical Data Sheet Parameters

  A series of typical data sheet examples show the flexibility of the MEASURE statement.

- Cell Characterization Using Data Driven Analysis

  Automates cell characterization, including timing simulator polynomial delay coefficient calculation. There is no limit on the number of parameters simultaneously varied or the number of analyses to be performed. Convenient ASCII file format for automated parameter input to Star-Hspice.

Determining Typical Data Sheet Parameters

This section describes how to determine typical data sheet parameters.

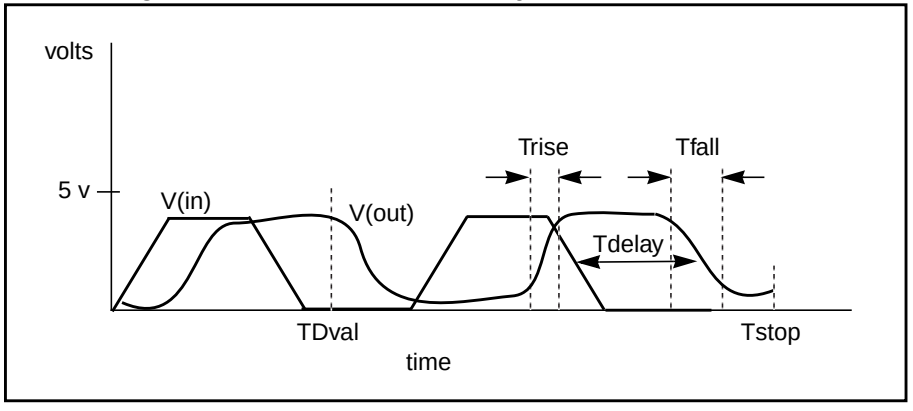
Rise, Fall, and Delay Calculations

The following example first calculates *vmax*, using the MAX function over the time region of interest. Then it calculates *vmin* using the MIN function. Finally, the measured parameters can be used in subsequent calculations for accurate 10% and 90% points in the determination of the rise and fall time. Note that the RISE=1 is relative to the time window formed by the delay *TDval.* Finally, the delay *Tdelay* is calculated using a fixed value for the measure threshold.

Example

```
.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 15-1: Rise, Fall, and Delay Time Demonstration



Ripple Calculation

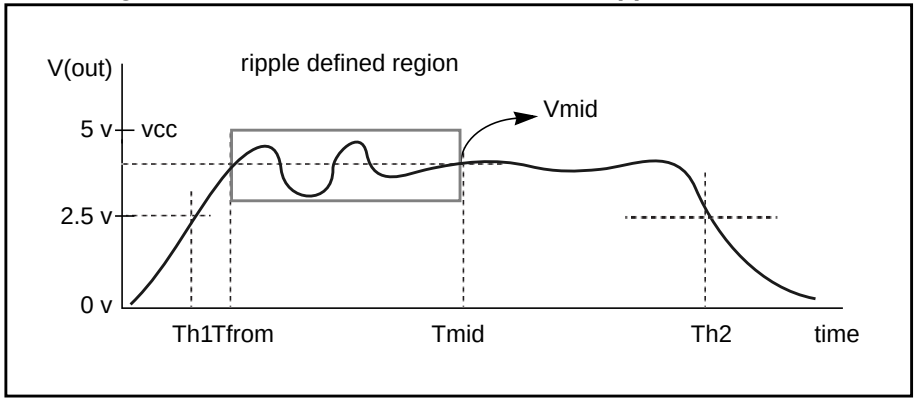
This example performs the following:

- Delimits the wave at the 50% of VCC points
- Finds the midpoint *Tmid*
- 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*

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 15-2: Waveform to Demonstrate Ripple Calculation



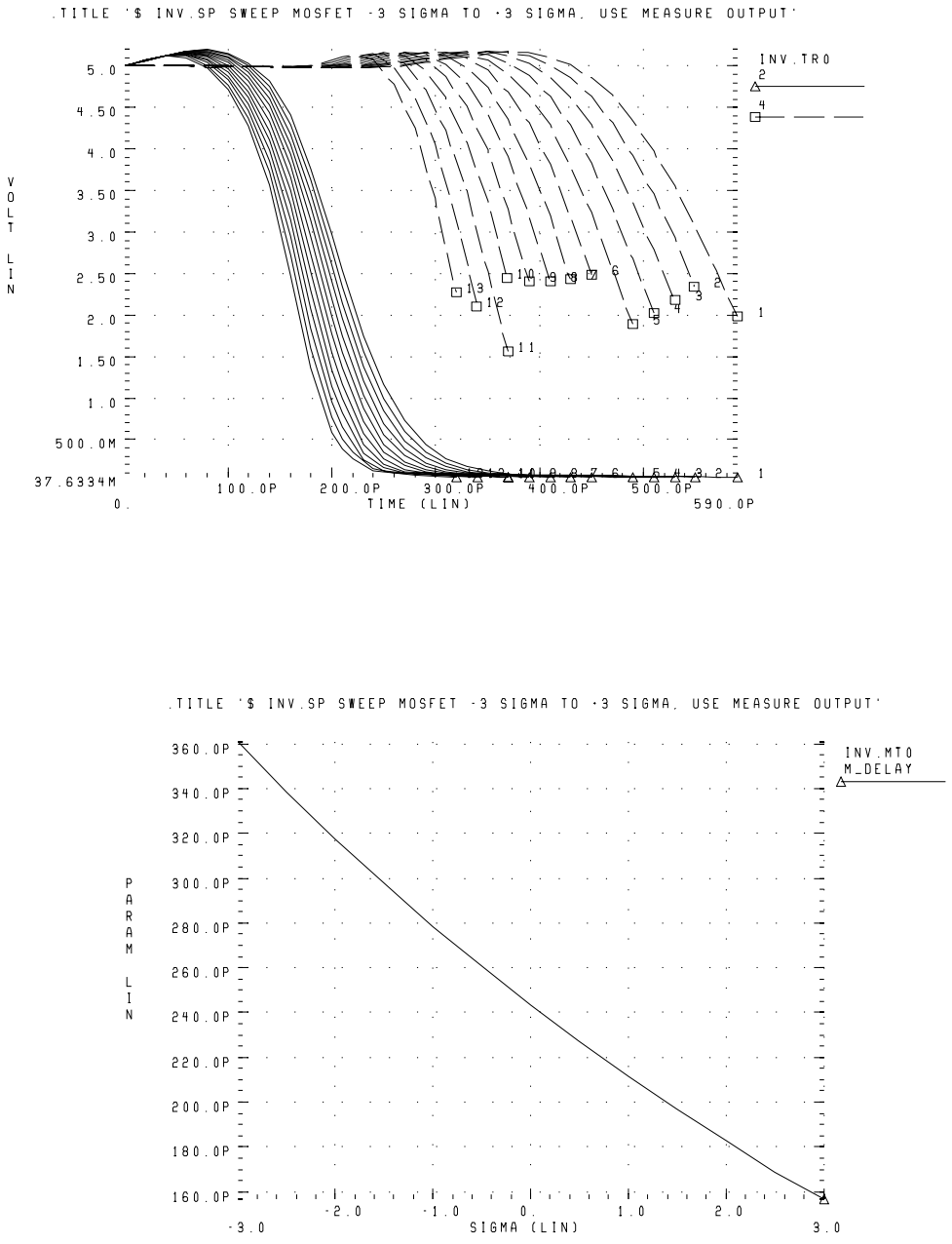
Sigma Sweep versus Delay

This file is set up to sweep sigma of the model parameter distribution while looking at the delay, giving the designer the delay derating curve for the model worst cases. This example is based on the demonstration file in *$installdir/demo/hspice/cchar/sigma.sp*. This technique of building a worst case sigma library is described in Performing Worst Case Analysis on page 13-8.

Example

```
.tran 20p 1.0n sweep sigma -3 3 .5
.meas m_delay trig v(2) val=vref fall=1 targ v(4)
+ val=vref fall=1
.param xlnew =’polycd-sigma*0.06u’
+ toxnew=’tox-sigma*10’
.model nch nmos LEVEL=28 xl = xlnew tox=toxnew
```

Figure 15-3: Inverter Pair Transfer Curves and Sigma Sweep vs. Delay



Delay versus Fanout

This example sweeps the sub-circuit multiplier to quickly generate a family of five load curves. You can obtain more accurate results, by buffering the input source with one stage. The following example calculates the mean, variance, sigma, and average deviance for each of the second sweep variables (*m_delay* and *rms_power*). This example is based on the demonstration file *$installdir/demo/hspice/cchar/load1.sp*.

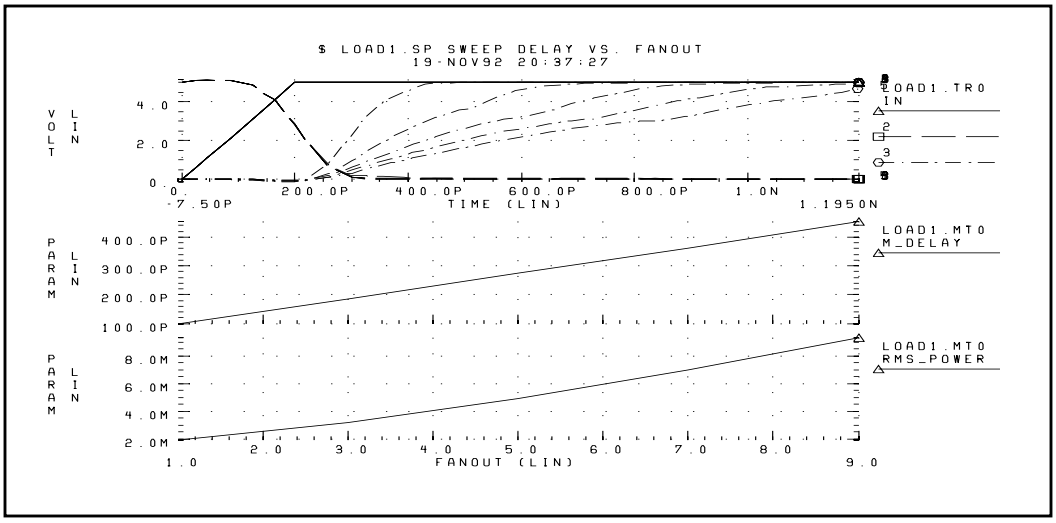
Input File Example

```
tran 100p 2.0n sweep fanout 1 10 2
.param vref=2.5
.meas m_delay trig v(2) val=vref fall=1
+ targ v(3) val=vref rise=1
.meas rms_power rms power

x1 in 2 inv
x2 2 3 inv
x3 3 4 inv m=fanout
```

Output Statistical 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 15-4: Inverter Delay and Power versus Fanout



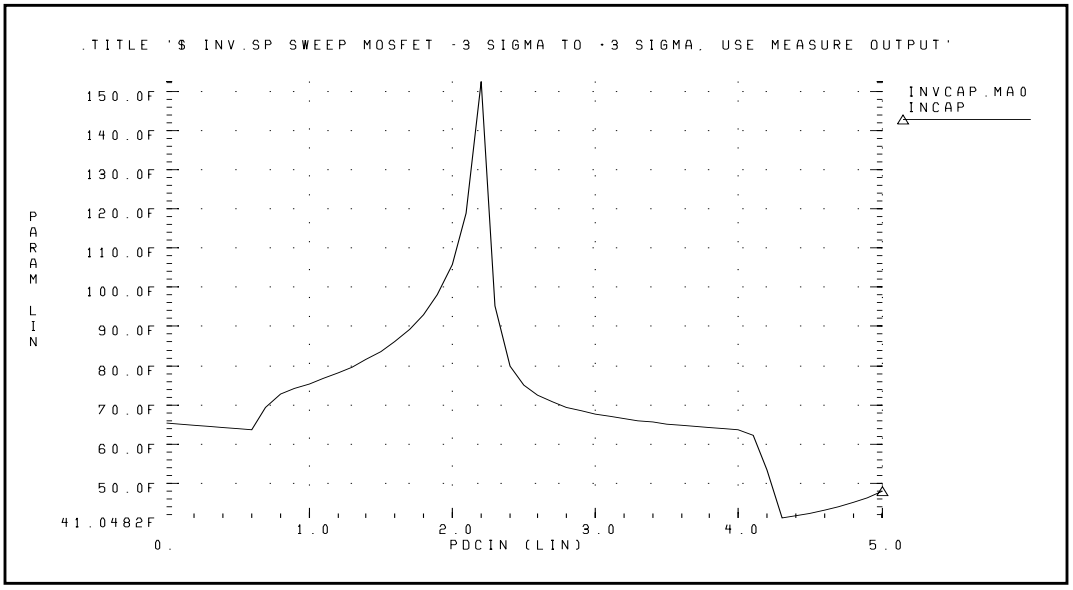
Pin Capacitance Measurement

This example shows the effect of dynamic capacitance at the switch point. It sweeps the DC input voltage (*pdcin*) to the inverter and performs an AC analysis each 0.1 volt. The measure parameter *incap* is calculated from the imaginary current through the voltage source at the 10 kilohertz point in the AC curve (not shown). The peak capacitance at the switch point occurs when the voltage at the output side is changing in the opposite direction from the input side of the Miller capacitor, adding the Miller capacitance times the inverter gain to the total effective capacitance.

Example

```
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 15-5: Graph of Pin Capacitance versus Inverter Input Voltage



Op-amp Characterization of ALM124

This example analyzes op-amps with .MEASURE statements to present a very complete data sheet. It references op-amp circuit output node out0 in the four .MEASURE statements using output variable operators for decibels vdb(out0), voltage magnitude vm(out0), and phase vp(out0). The example is taken from the demonstration file *demo/apps/alm124.sp.*

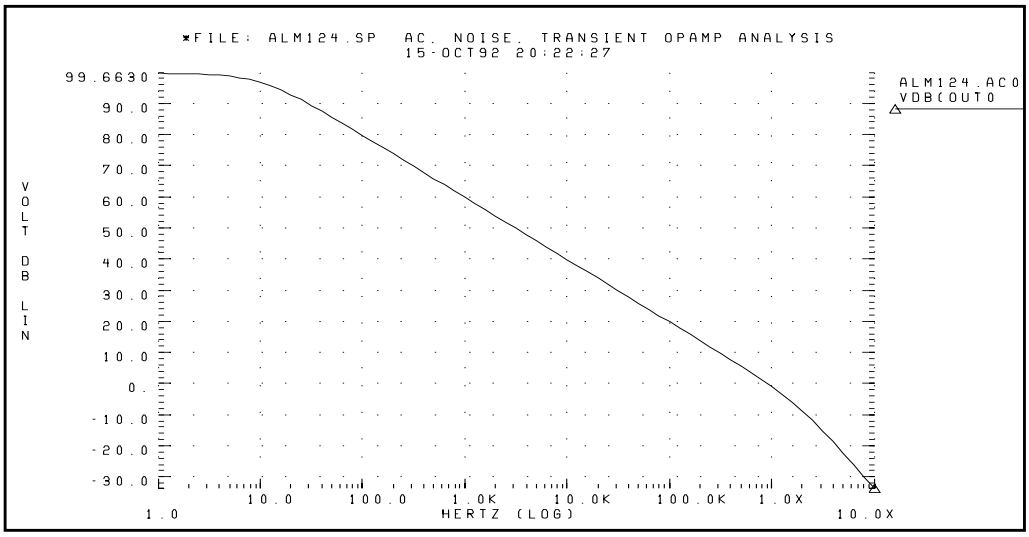
Input File Example

```
.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)
```

Measure 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 15-6: Magnitude Plot of Op-Amp Gain



Cell Characterization Using Data Driven Analysis

This section provides example input files that perform cell characterization of an inverter based on 3-micron MOSFET technology. The program finds the propagation delay and rise and fall times for the inverter for best, worst, and typical cases for different fanouts. This data then can be used as library data for digital-based simulators such as those found in the simulation of gate arrays and standard cells.

The example, taken from the demonstration file *$installdir/demo/hspice/apps/cellchar.sp*, demonstrates the use of the .MEASURE statement, the .DATA statement, and the AUTOSTOP option in the characterization of a CMOS inverter. Figure 15-7 and Figure 15-8 are identical except that their input signals are complementary. The circuit in Figure 15-7 calculates the rise time and the low-to-high propagation delay time. The circuit in Figure 15-8 calculates the fall time and the high-to-low propagation delay time. When only one circuit is used, CPU time increases because the analysis time increases to calculate both rise and fall times.

Figure 15-7: Cell Characterization Circuit 1

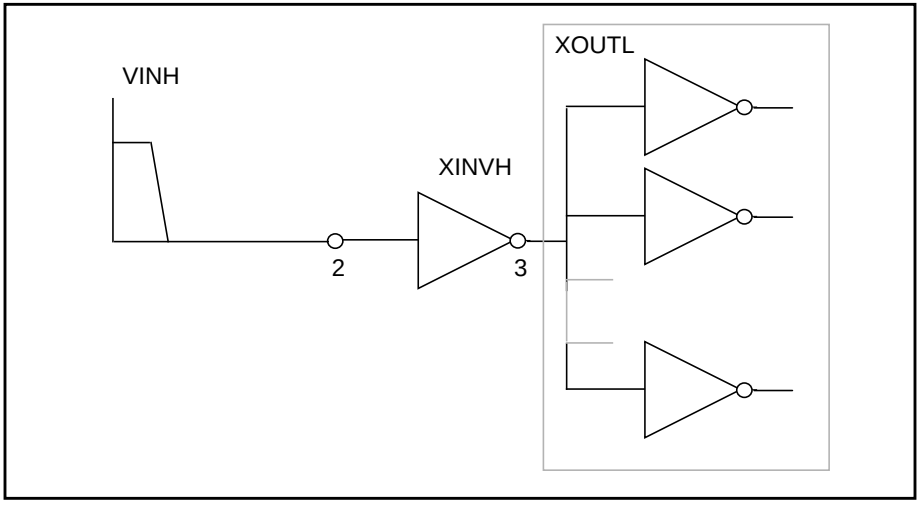
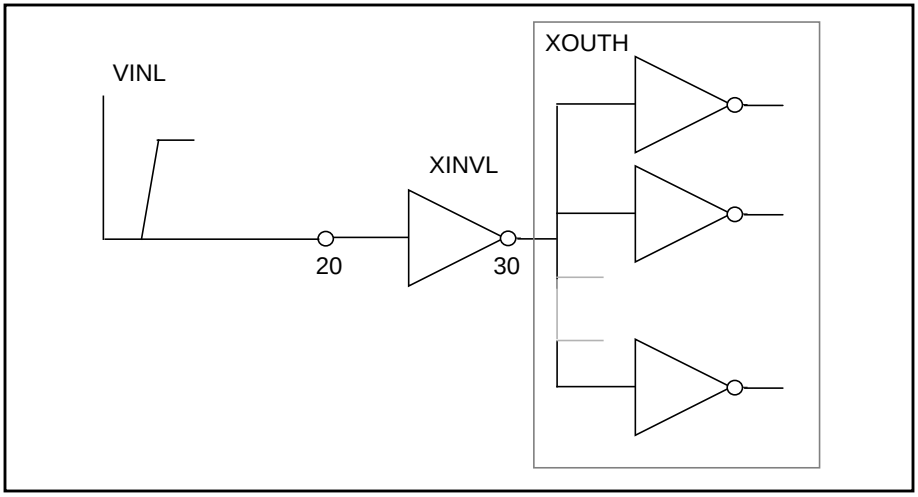


Figure 15-8: Cell Characterization Circuit 2



The sub-circuit XOUTL or XOUTH represents the fanout of the cell (inverter). Star-Hspice modifies fanout by specifying different multipliers (m) in the sub-circuit calls.

Star-Hspice also provides local and global temperature specifications. This example characterizes the cell at global temperature 27, while devices M1 and M2 are at temperature (27+DTEMP). The .DATA statement specifies the DTEMP value.

The example uses a transient parameterized sweep with the .DATA and .MEASURE statements to determine the timing of the inverter for best, typical and worst cases. The parameters varied include power supply, input rise and fall time, fanout, MOSFET temperature, n-channel and p-channel threshold, and both the drawn width and length of the MOSFET. Use the AUTOSTOP option to speed simulation time and work with the .MEASURE statement. Once the .MEASURE statement determines the parameter to be measured, the AUTOSTOP option terminates the transient sweep, even though it has not completely swept the transient sweep range specified.

The .MEASURE statement uses quoted string parameter variables to measure the rise and fall times, as well as the propagation delays. 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) and 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 a time delay, a simulation cycle specifying delay time in the .MEASURE statement, or in the input pulse statement.

The following example features:

- AUTOSTOP and .MEASURE statements
- Mean, variance, sigma, and avgdev calculations
- Circuit and element temperature
- Algebraic equation handling
- PAR( ) as output variable in the .MEASURE statement
- Sub-circuit parameter passing and sub-circuit multiplier
- .DATA statement

Example Input Files

```
FILE: CELLCHAR.SP
*
.OPTIONS SPICE NOMOD AUTOSTOP
.PARAM TD=10N PW=50N TRR=5N TRF=5N VDD=5 LDEL=0 WDEL=0
+ NVT=0.8 PVT=-0.8 DTEMP=0 FANOUT=1
.GLOBAL VDD
* – global supply name
.TEMP 27
```

SUBCKT Definition

```
.SUBCKT INV IN OUT
M1 OUT IN VDD VDD P L=3U W=15U DTEMP=DTEMP
M2 OUT IN 0 0 N L=3U W=8U DTEMP=DTEMP
CL OUT 0 200E-15 .001
CI IN 0 50E-15 .001
.ENDS
```

SUBCKT Calls

```
XINVH 2 3 INV      $–- INPUT START HIGH
XOUTL 3 4 INV M=FANOUT
XINVL 2030 INV      $–- INPUT START LOW
XOUTH 30 40INV M=FANOUT
* – INPUT VOLTAGE SOURCES
VDD VDD 0 VDD
VINH 2 0 PULSE(VDD,0,TD,TRR,TRF,PW,200NS)
```

```
VINL 20 0 PULSE(0,VDD,TD,TRR,TRF,PW,200NS)
* – MEASURE STATEMENTS FOR RISE, FALL, AND PROPAGATION
+ DELAYS
.MEAS RISETIME  TRIG PAR('V(3) -0.1*VDD')  VAL=0  RISE=1
+               TARG PAR('V(3) -0.9*VDD')  VAL=0  RISE=1
.MEAS FALLTIME  TRIG PAR('V(30)-0.9*VDD')  VAL=0  FALL=1
+              TARG PAR('V(30)-0.1*VDD')  VAL=0  FALL=1
.MEAS TPLH      TRIG PAR('V(2) -0.5*VDD')  VAL=0  FALL=1
+               TARG PAR('V(3) -0.5*VDD')  VAL=0  RISE=1
.MEAS TPHL      TRIG PAR('V(20)-0.5*VDD')  VAL=0  RISE=1
+               TARG PAR('V(30)-0.5*VDD')  VAL=0  FALL=1
* – ANALYSIS SPECIFICATION
.TRAN 1N 500N SWEEP DATA=DATNM
* – DATA STATEMENT SPECIFICATION
.DATA DATNM
VDD   TRR   TRF  FAN   DTEMP  NVT   PVT    LDEL     WDEL
                 OUT
5.0   2N    2N   2     0      0.8   -0.8   0        0     $ TYPICAL
5.5   1N    1N   1     -80    0.6   -0.6   -0.2U    0.2U  $ BEST
4.5   3N    3N   10    100    1.0   -1.0   +0.2U -0.2U    $ WORST
5.0   2N    2N   2     0      1.0   -0.6   0        0     $ STRONG P
                                                         $ WEAK N
5.0   2N    2N   2     0      0.6   -1.0   0        0     $ WEAK P
                                                         $ STRONG N
5.0   2N    2N   4     0      0.8   -0.8   0        0    $ FANOUT=4
5.0   2N    2N   8     0      0.8   -0.8   0        0    $ FANOUT=8
.ENDDATA
```

Models

```
.MODEL N NMOS LEVEL=2 LDEL=LDEL WDEL=WDEL
+ VTO=NVT    TOX =300     NSUB=1.34E16  UO=600
+ LD=0.4U    WD  =0.6U    UCRIT=4.876E4 UEXP=.15
+ VMAX=10E4  NEFF=15      PHI=.71       PB=.7
+ RS=10      RD  =10      GAMMA=0.897   LAMBDA=0.004
+ DELTA=2.31 NFS =6.1E11  CAPOP=4
+ CJ=3.77E-4 CJSW=1.9E-10 MJ=.42 MJSW=.128
*
.MODEL P PMOS  LEVEL=2 LDEL=LDEL WDEL=WDEL
+ VTO=PVT     TOX=300      NSUB=0.965E15 UO=250
+ LD=0.5U     WD=0.65U     UCRIT=4.65E4  UEXP=.25
+ VMAX=1E5    NEFF=10      PHI=.574      PB=.7
+ RS=15       RD=15        GAMMA=0.2     LAMBDA=.01
+ DELTA=2.486 NFS=5.2E11   CAPOP=4
+ CJ=1.75E-4  CJSW=2.3E-10 MJ=.42        MJSW=.128
.END
```

A sample of measure statements is printed:

```
***  MEASURE STATEMENT RESULTS FROM THE FIRST ITERATION
*** ($ TYPICAL)

RISETIME = 3.3551E-09 TARG= 1.5027E-08 TRIG= 1.1672E-08
FALLTIME = 2.8802E-09 TARG= 1.4583E-08 TRIG= 1.1702E-08
TPLH     = 1.8537E-09 TARG= 1.2854E-08 TRIG= 1.1000E-08
TPHL     = 1.8137E-09 TARG= 1.2814E-08 TRIG= 1.1000E-08

***  MEASURE STATEMENT RESULTS FROM THE LAST ITERATION
*** ($ FANOUT=8)

RISETIME = 8.7909E-09 TARG= 2.0947E-08 TRIG= 1.2156E-08
FALLTIME = 7.6526E-09 TARG= 1.9810E-08 TRIG= 1.2157E-08
TPLH     = 3.9922E-09 TARG= 1.4992E-08 TRIG= 1.1000E-08
TPHL     = 3.7995E-09 TARG= 1.4800E-08 TRIG= 1.1000E-08

MEAS_VARIABLE = RISETIME
MEAN  =  6.5425E-09  VARIAN =  4.3017E-17
SIGMA =  6.5588E-09  AVGDEV =  4.6096E-09

MEAS_VARIABLE = FALLTIME
MEAN  =  5.7100E-09  VARIAN =  3.4152E-17
SIGMA =  5.8440E-09  AVGDEV =  4.0983E-09

MEAS_VARIABLE = TPLH
MEAN  =  3.1559E-09  VARIAN =  8.2933E-18
SIGMA =  2.8798E-09  AVGDEV =  1.9913E-09

MEAS_VARIABLE = TPHL
MEAN  =  3.0382E-09  VARIAN =  7.3110E-18
SIGMA =  2.7039E-09  AVGDEV =  1.8651E-0
```

Figure 15-9: Plotting the Simulation Outputs

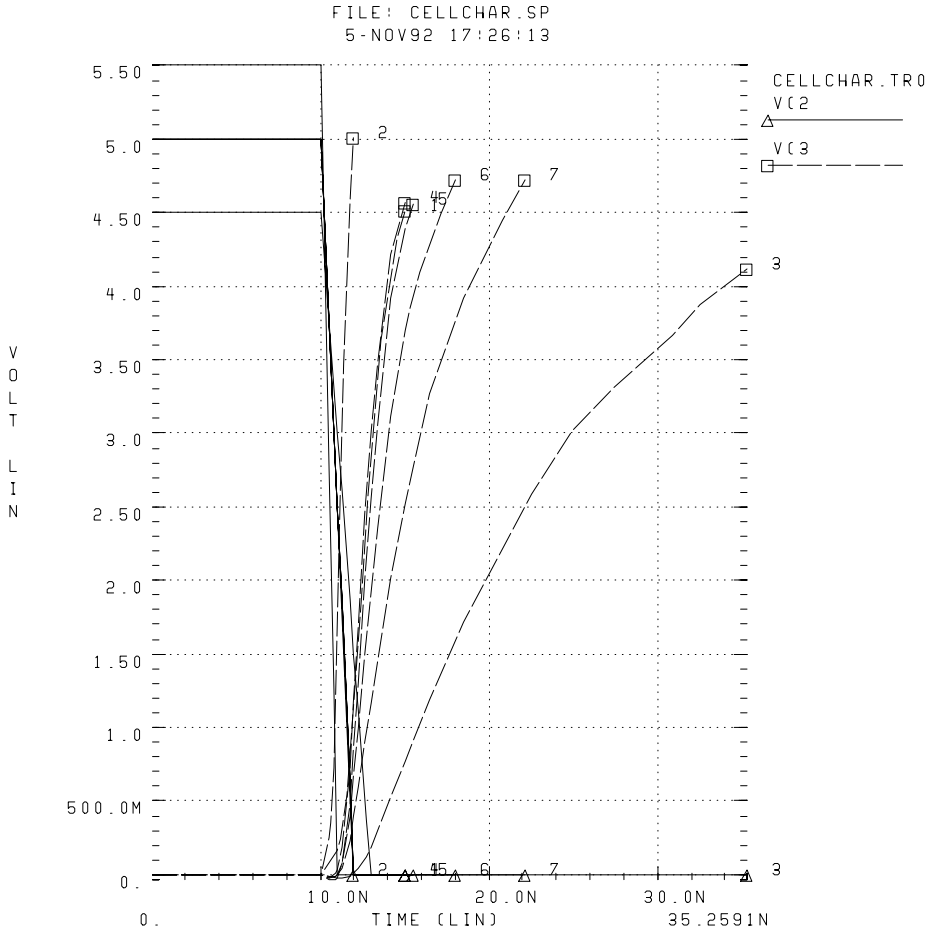
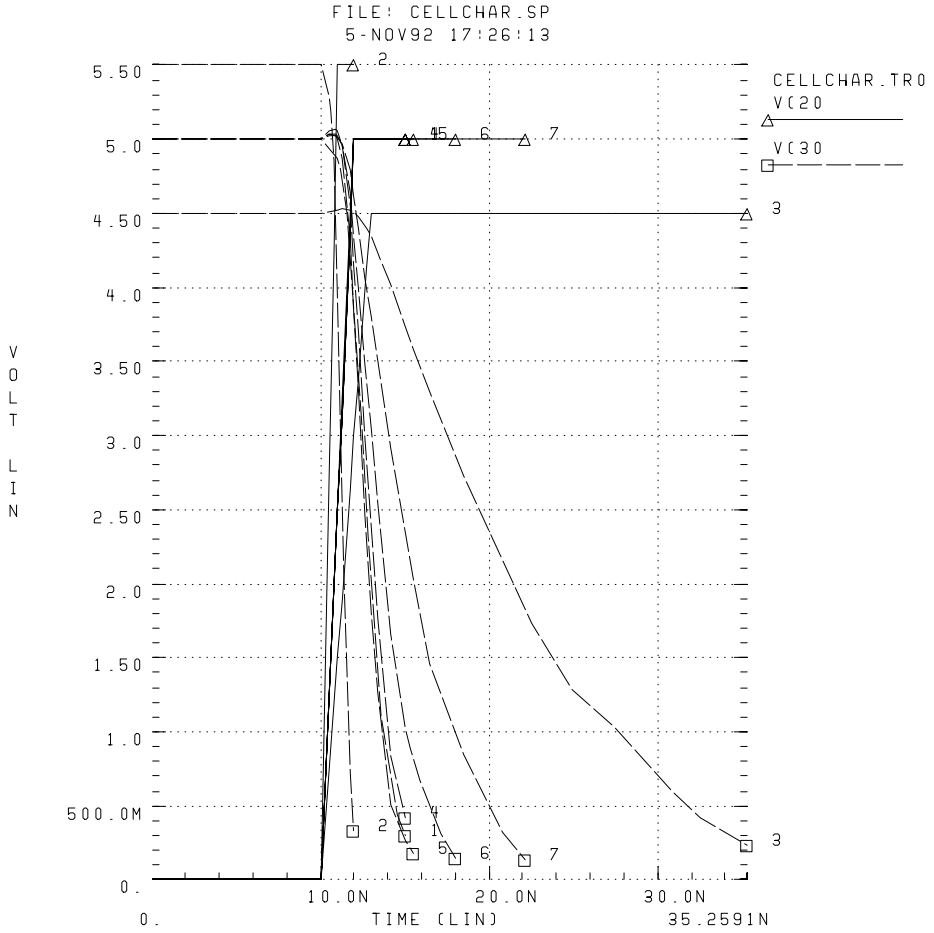


Figure 15-10: Verifying the Measure Statement Results by the Plots



Avant!

Chapter 16
Signal Integrity

The performance of an IC design is no longer limited to how many million transistors a vendor fits on a single chip. With tighter packaging space and increasing clock frequencies, packaging and system-level performance issues such as crosstalk and transmission lines are becoming increasingly significant.

At the same time, with the popularity of multi-chip packages and increased I/O counts, package design itself is becoming more and more like chip design.

This chapter describes how to maintain signal integrity for your design, and covers the following topics:

Preparing for Simulation

To simulate a PC board or backplane with Star-Hspice, you must consider models for:

- A driver cell, including the parasitic pin capacitances and package lead inductances
- Transmission lines
- A receiver cell with its parasitic pin capacitances and package lead inductances
- Terminations or other electrical elements on the line

It is important to model the transmission line as closely as possible— that is, to include all electrical elements exactly as they are laid out on the backplane or printed circuit board, to maintain the integrity of the simulation.

With readily available I/O drivers from ASIC vendors and Star-Hspice's advanced lossy transmission lines, you can simulate the electrical behavior of the board interconnect, bus, or backplane to analyze the transmission line behavior under various conditions.

Simulation is possible because the critical models and simulation technology exist.

- Many manufacturers of high-speed components use Star-Hspice already.
- The complexity can be hidden from the system level.
- The necessary electrical characteristics are preserved with full transistor level library circuits.

Star-Hspice has been enhanced for systems simulation with:

- Systems level behavior, such as local component temperature and independent models, to allow accurate prediction of electrical behavior
- Automatic inclusion of library components via the SEARCH option
- Lossy transmission line models that:
  - Support common mode simulation
  - Include ground plane reactance
  - Include resistive loss of conductor and ground plane
  - Allow multiple signal conductors
  - Require minimum CPU computation time

Signal Integrity Problems

Some signal integrity problems that can cause failures in high-speed designs are listed in Table 16-1.

Table 16-1: High-Speed Design Problems and Solutions

Signal Integrity Problem	Causes	Solution
Noise: delta I (current)	Multiple simultaneously switching drivers; high-speed devices create larger delta I	Adjust or evaluate location, size, and value of decoupling capacitors.
Noise: coupled (crosstalk)	Closely spaced parallel traces	Establish parallel line length design rules.
Noise: reflective	Impedance mismatch	Reduce the number of connectors and select proper impedance connectors.
Delay: path length	Poor placement and routing; too many or too few layers; chip pitch	Choose MCM or other high-density packaging technology.
Propagation speed	Dielectric medium	Choose dielectric with lowest dielectric constant.
Delay: rise time degradation	Resistive loss and impedance mismatch	Adjust width, thickness and length of line.

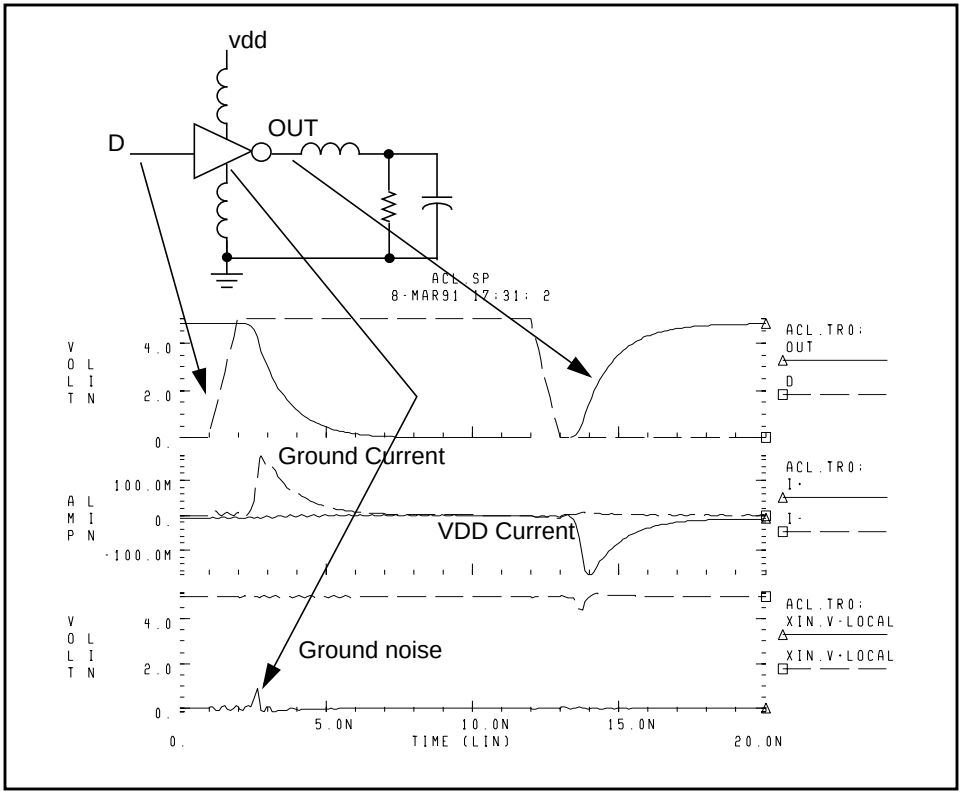
Analog Side of Digital Logic

Circuit simulation of a digital system only becomes necessary when the analog characteristics of the digital signals become electrically important. Is the digital circuit a new design, or simply a fast version of the old design? Many new digital products are really faster versions of existing designs. The transition from a 100

MHz to a 150 MHz Pentium PC may not require extensive logic simulations. However, the integrity of the digital quality of the signals may require very careful circuit analysis.

The source of a signal integrity problem is the digital output driver. A high-speed digital output driver can only drive a few inches before the noise and delay due to the wiring become a problem. To speed up circuit simulation and modeling, you can create analog behavioral models that mimic the full analog characteristics at a fraction of the traditional evaluation time. The simulation of the output buffer in Figure 16-1 demonstrates the analog behavior of a digital gate simulated in the Star-Hspice circuit simulator.

Figure 16-1: Simulation of Output Buffer with 2 ns Delay and 1.8 ns Rise and Fall Times



The roadblocks to successful high-speed digital designs are noise and signal delays. Digital noise can come from several sources. The fundamental digital noise sources are:

- Line termination noise
- Ground bounce noise
- Coupled line noise

Line termination noise is the additional voltage that is reflected from the load back to the driver because of impedance mismatch. Digital output buffers are not designed to have accurately controlled output impedance and most buffers have different rising and falling edge impedances.

Ground bounce noise is generated where leadframes or other circuit wires cannot be formed into transmission lines. The resulting inductance creates an induced voltage in the ground circuit, the supply circuit, and the output driver circuit. The ground bounce noise lowers the noise margins for the rest of the system. Coupled line noise is the noise induced from lines that are physically adjacent. This noise is generally most severe for data lines that are next to clock lines.

Circuit delays become critical as timing requirements become tighter. The key circuit delays are:

- Gate delays
- Line turnaround delays for tristate buffers
- Line length delays (clock skew)

Logic analysis only addresses gate delays. You can compute the variation in the gate delay from circuit simulation only if you understand the best case and worst case manufacturing conditions. The line turnaround delays add to the gate delays because extra margin must be added so that multiple tristate buffer drivers do not simultaneously turn on. The line length delay affects the clock skew most directly in most systems. As system cycle times approach the speed of electromagnetic signal propagation for the printed circuit board, consideration of the line length becomes critical. The system noises and line delays interact with the electrical characteristics of the gates and may require circuit level simulation.

Analog details find digital systems problems. Exceeding the noise quota may not cause a system to fail. Only when a digital input is being accepted does the maximum noise become a problem. If a digital systems engineer can decouple the system, much higher noise can be accepted.

Common decoupling methods are:

- Multiple ground and power planes on the PCB, MCM, PGA
- Separating signal traces with ground traces
- Decoupling capacitors
- Series resistors on output buffer drivers
- Twisted pair line driving

In present systems designs, you must select the best packaging methods at the printed circuit board level, the multi-chip module level, and the pin grid array level. Extra ground and power planes are often necessary to lower the supply inductance and provide decoupling. Decoupling capacitors must have very low internal inductance in order to be effective for high speed designs. Newer designs frequently use series resistance in the output drivers to lower circuit ringing. Finally, in critical high speed driver applications, twisted differential pair transmission lines are used.

The systems engineer must determine how to partition the logic. The propagation speed of signals on a printed circuit board is about 6 in/ns. As digital designs become faster, the wiring interconnect becomes a factor in deciding how to partition the logic. The critical wiring systems are:

- IC level wiring
- Package wiring for SIPs, DIPs, PGAs, MCMs
- Printed circuit board wiring
- Backplane and connector wiring
- Long lines – power, coax, twisted pair

Systems designers who use ASIC or custom integrated circuits as part of their system logic partitioning strategy find that they must make decisions about integrated circuit level wiring. The more familiar decisions involve the selection of packages and the arrangement of packages on a printed circuit board. Large systems generally have a central backplane that becomes the primary challenge at the system partition level.

Use the following equation to estimate wire length when transmission line effects become noticeable:

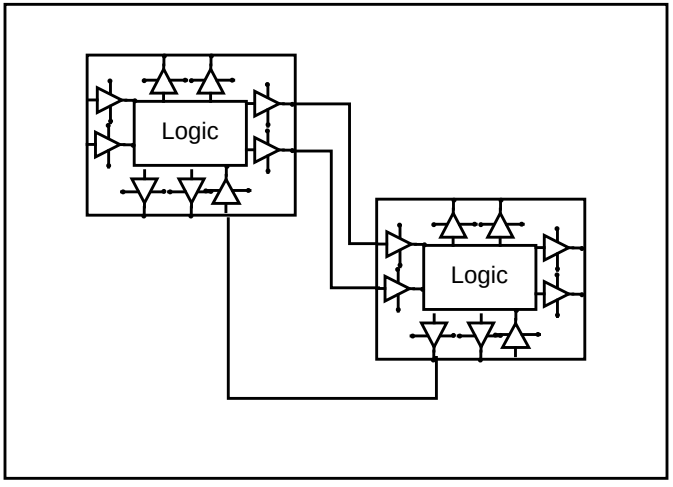
```
critical length=(rise time)*velocity/8
```

For example, if rise time is 1 ns and board velocity is 6 in/ns, distortion becomes noticeable at a wire length of 3/4 in. The Star-Hspice circuit simulator automatically generates models for each type of wire to define full loss transmission line effects.

ECL logic design engineers typically partitioned the system by calculating the noise quota for each line. Now, most high-speed digital logic must be designed with respect to the noise quota so that the engineer knows how much noise and delay can be accepted before the timing and logic levels fail.

To solve the noise quota problem, you must calculate the noise associated with the wiring. Large integrated circuits can be separated into two parts: the internal logic and the external input and output amplifiers.

Figure 16-2: Analog Drivers and Wires



Using mixed digital and analog tools such as Avant! Star-Hspice and Viewlogic Viewsim A/D, you can merge a complete system together with full analog quality timing constraints and full digital representation. You can simultaneously evaluate noise quota calculation subject to system timing.

Optimizing TDR Packaging

Packaging plays an important role in determining the overall speed, cost, and reliability of a system. With today's small feature sizes and high levels of integration, a significant portion of the total delay comes from the time required for a signal to travel between chips.

Multi-layer ceramic technology has proven to be well suited for high-speed GaAs IC packages.

The multi-chip module minimizes the chip-to-chip spacing and reduces the inductive and capacitive discontinuity between the chips mounted on the substrate with a more direct path (die-bump-interconnect-bump-die), thus eliminating wirebonding. In addition, narrower and shorter wires on the ceramic substrate have much less capacitance and inductance than the PC board interconnections.

Time domain reflectometry (TDR) is the closest measurement to actual digital component function. It provides a transient display of the impedance versus time for pulse behavior.

With a digitized TDR file, you can automatically select design components using the Star-Hspice optimizer. You can extract critical points from digitized TDR files using the Star-Hspice .MEASURE statement, and use the results as electrical specifications for optimization. This process eliminates recurring design cycles to find component values to curve-fit the TDR files.

Figure 16-3: Optimization Process

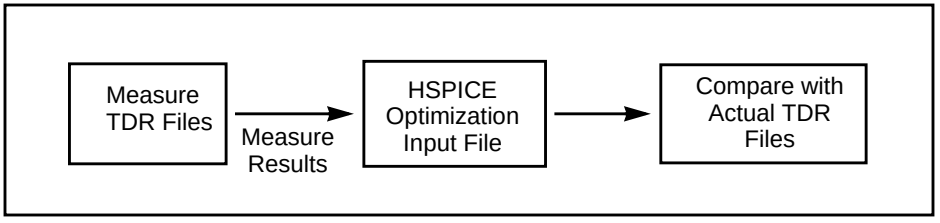
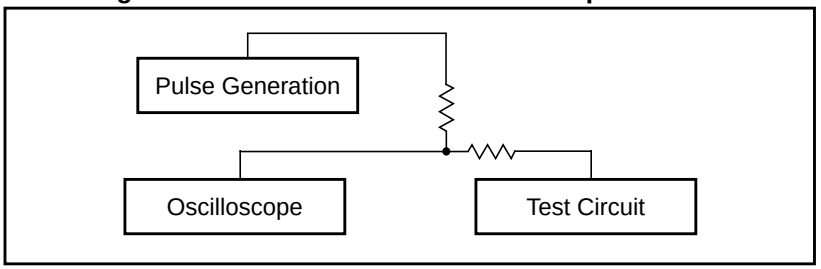


Figure 16-4: General Method for TDR Optimization



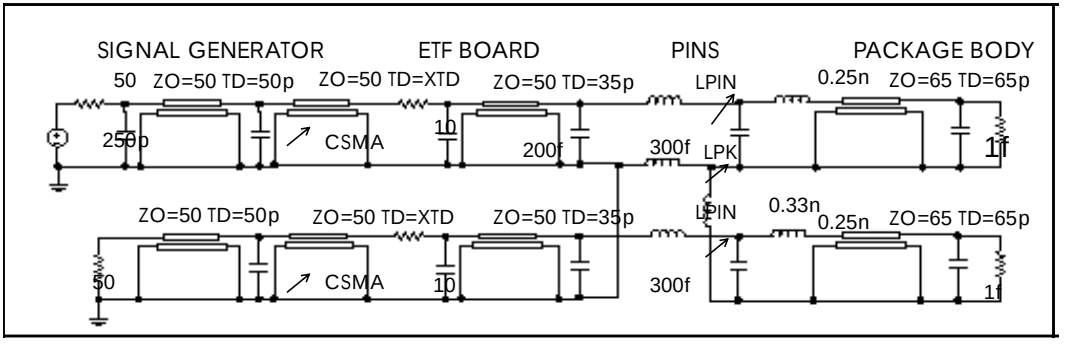
The following is a method used for realistic high-speed testing of packaging.

Test fixtures were designed that closely emulate a high-speed system environment. A Star-Hspice model was constructed for measurements using ideal transmission lines and discrete components.

The circuit tested contained the following components:

- Signal generator
- Coax connecting the signal generator to ETF (engineering test fixture) board
- ETF board
- Package pins
- Package body

Figure 16-5: SPICE Model for Package-Plus-Test Fixture Optimized Parameters: XTD, CSMA, LPIN, and LPK



The package tests performed traditional time domain measurements using a digital sampling oscilloscope. Tests were designed to observe the reflected and transmitted signals derived from the built-in high-speed pulse generator and translated output signals into digitized time domain reflectometer files (voltage vs. time).

A fully developed SPICE model was used to simulate the package-plus-test fixture. The simulated and measured reflected/transmitted signals were compared.

The input netlist file for this experiment is shown on the following pages. Output plots are shown in Figure 16-6 through Figure 16-9.

You can further investigate this experiment using Star-Hspice's advanced lossy transmission lines to include attenuation and dispersion.

TDR Optimization Procedure

Measure Critical Points in the TDR Files

```
Vin 1 0 PWL(TIME,VOLT)
.DATA D_TDR
      TIME    VOLT
      0       0.5003mV
      0.1n    0.6900mV
       ...
      2.0n    6.4758mV
.ENDDATA
  .TRAN DATA=D_TDR
  .MEAS .....
.END
```

Set Up an Input Optimization File

```
$ SPICE MODEL FOR PACKAGE-PLUS-TEST FIXTURE
$ AUTHOR: DAVID H. SMITH & RAJ M. SAVARA
.OPTION POST RELV=1E-4 RELVAR=1E-2
$ DEFINE PARAMETERS

.PARAM LV=-0.05 HV=0.01 TD=1P TR=25P TF=50P TPW=10N
+ TPER=15N
```

```
$ PARAMETERS TO BE OPTIMIZED
.PARAM CSMA=OPT1(500f,90f,900f,5f)
+      XTD=OPT1(150p,100p,200p)
+      LPIN=OPT1(0.65n,0.10n,0.90n,0.2n)
+      LPK=OPT1(1.5n,0.75n,3.0n,0.2n)
+      LPKCL=0.33n
+      LPKV=0.25n
```

Signal Generator

```
VIN   S1 GND PULSE LV HV TD TR TF TPW TPER
RIN   S1 S2 50
CIN1 S2 GND 250f
TCOAX S2 GND SIG_OUT GND ZO=50 TD=50p
```

ETF Board

```
CSNAL SIG_OUT GND CSMA
TEFT2 SIG_OUT GND E3 GND ZO=50 TD=XTD
RLOSS1 E3 E4 10
CRPAD1 E4 GND 200f
TLIN2 E4 GND ETF_OUT GND ZO=50 TD=35p
CPAD2 ETF_OUT GND 300f
TLIN1 E5 GND E6 GND ZO=50 TD=35p
CPAD1 E5 GND 300f
CRPAD2 E6 GND 200f
RLOS1 E6 E7 10
TEFT1 E7 GND E8 GND ZO=50 TD=XTD
CSMA2 E8 GND CSMA
TCOAX2 E8 GND VREF1 GND ZO=50 TD=50p
RIN1 VREF1 GND 50
```

Package Body

```
LPIN1 ETF_OUT P1 LPIN
LPK1 GND P5 LPK
LPKGCL P5 NVOUT2 LPKCL
CPKG1 P1 P5 250f
LPKV1 P1 P2 LPKV
TPKGL P2 NVOUT2 VOUT NVOUT2 ZO=65 TD=65P
CBPL VOUT NVOUT 1f
ROUT1 VOUT NVOUT 50meg
LPIN2 E5 P3 LPIN
CPKG2 P3 NVOUT2 250f
LPKV2 P3 P4 LPKV
TPKG2 P4 NVOUT2 VOUT2 NVOUT2 ZO=65 TD=65p
```

```
CBPD1 VOUT2 NVOUT2 1f
ROUT2 VOUT2 NVOUT2 50meg

$ BEFORE OPTIMIZATION
.TRAN .004NS 2NS

$ OPTIMIZATION SETUP
.MODEL OPTMOD OPT ITROPT=30

.TRAN .05NS 2NS SWEEP OPTIMIZE=OPT1
+            RESULTS=MAXV,MINV,MAX_2,COMP1,PT1,PT2,PT3
+            MODEL=OPTMOD

$ MEASURE CRITICAL POINTS IN THE REFLECTED SIGNAL
$ GOALS ARE SELECTED FROM MEASURED TDR FILES

.MEAS TRAN COMP1 MIN V(S2) FROM=100p TO=500p
+ GOAL=-27.753
.MEAS TRAN PT1 FIND V(S2) AT=750p GOAL=-3.9345E-3
+ WEIGHT=5
.MEAS TRAN PT2 FIND V(S2) AT=775p GOAL=2.1743E-3
+ WEIGHT=5
.MEAS TRAN PT3 FIND V(S2) AT=800p GOAL=5.0630E-3
+ WEIGHT=5

$ MEASURE CRITICAL POINTS IN THE TRANSMITTED SIGNAL
$ GOALS ARE SELECTED FROM MEASURED TDR FILES

.MEAS TRAN MAXV FIND V(VREF1)AT=5.88E-10 GOAL=6.3171E-
+ WEIGHT=7
.MEAS TRAN MINV FIND V(VREF1) AT=7.60E-10 GOAL=-
+ 9.9181E-3
.MEAS TRAN MAX_2 FIND V(VREF1) AT=9.68E-10
+ GOAL=4.9994E-3

$ COMPARE SIMULATED RESULTS WITH MEASURED TDR VALUES
.TRAN .004NS 2NS
.PRINT C_REF=V(S2) C_TRAN=V(VREF1)

.END
```

Figure 16-6: Reflected Signals Before Optimization

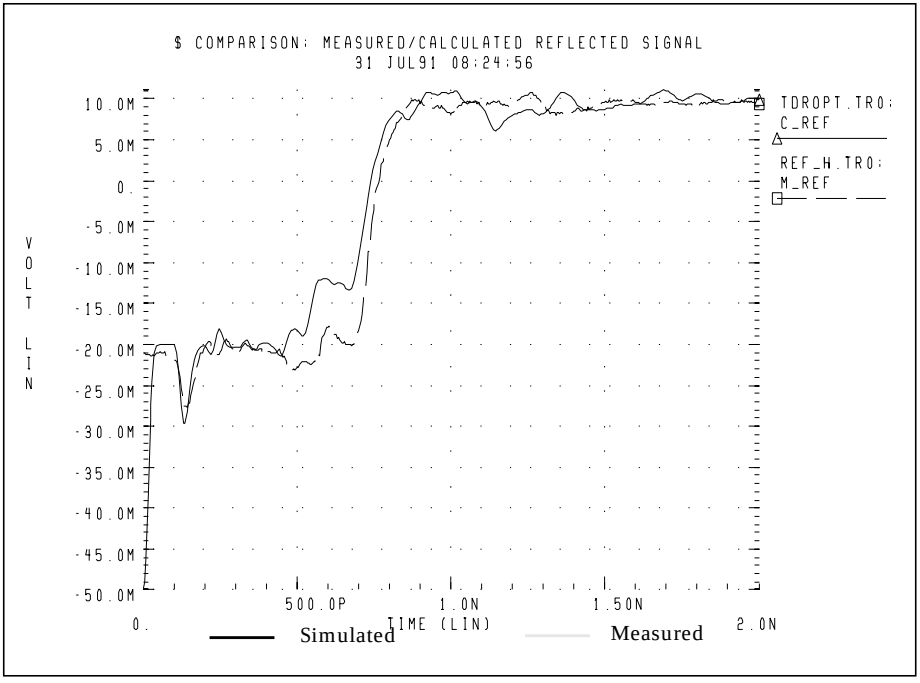


Figure 16-7: Reflected Signals After Optimization

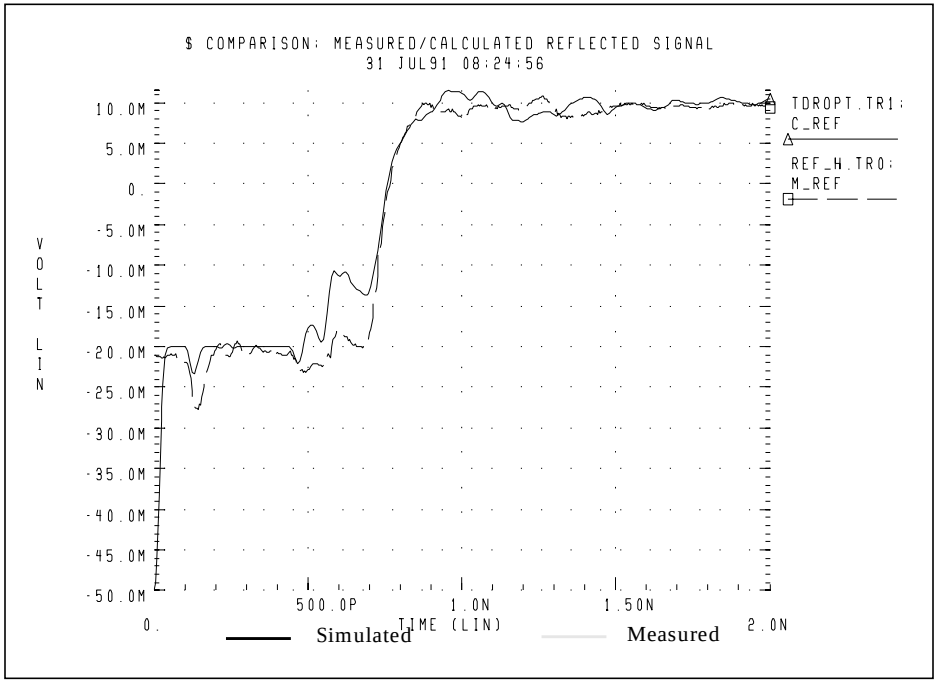


Figure 16-8: Transmitted Signals Before Optimization

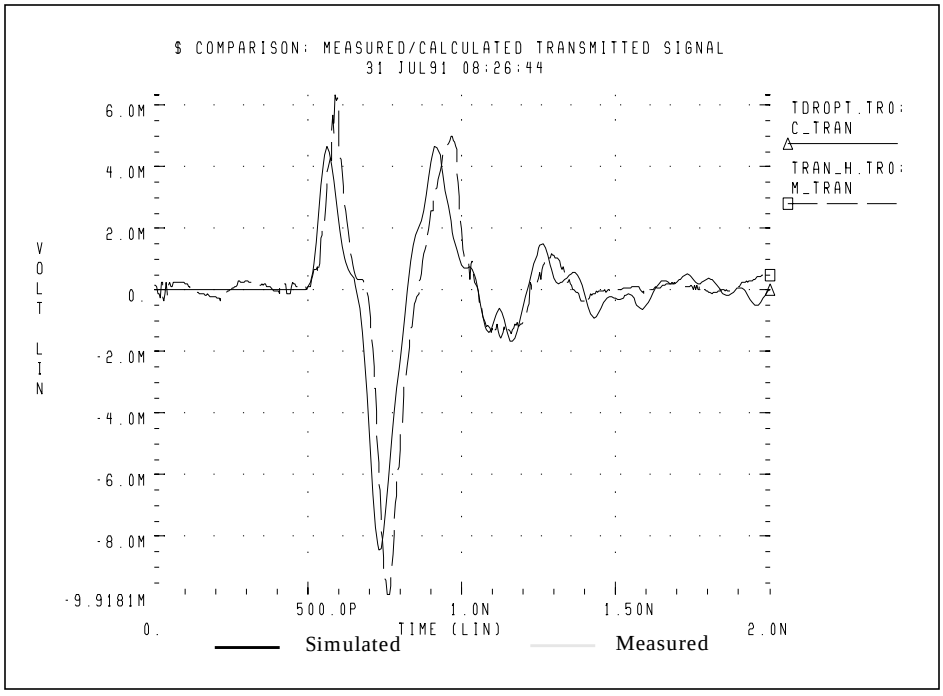
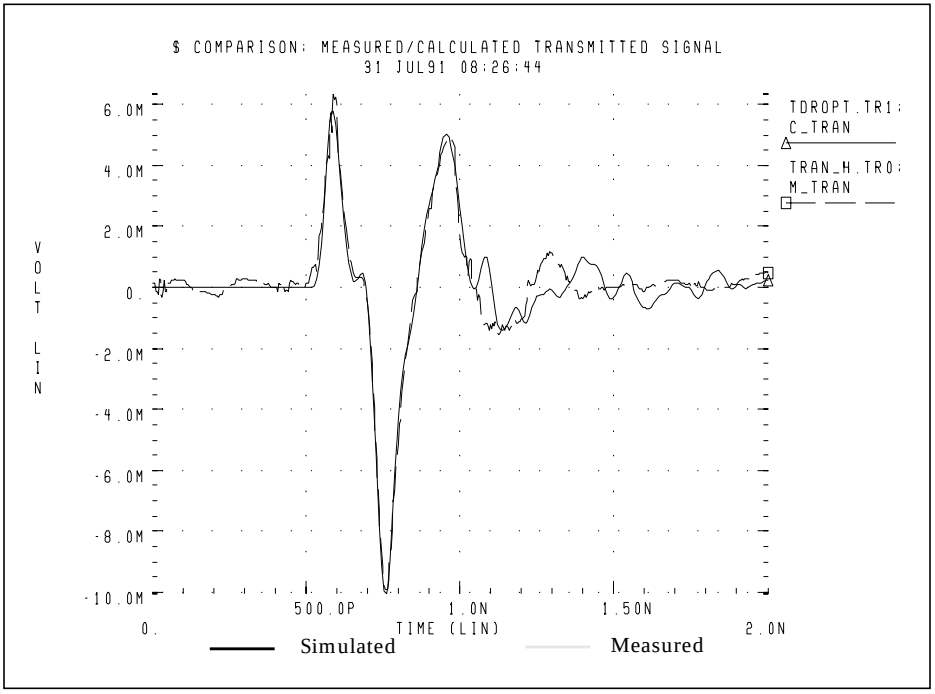


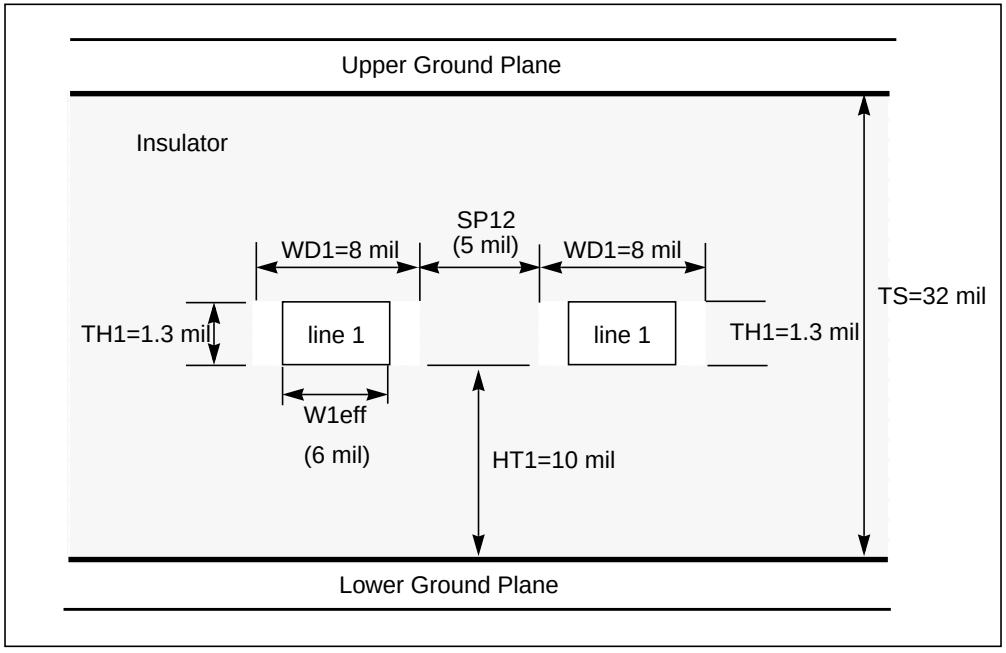
Figure 16-9: Transmitted Signals after Optimization



Simulating Circuits with Signetics Drivers

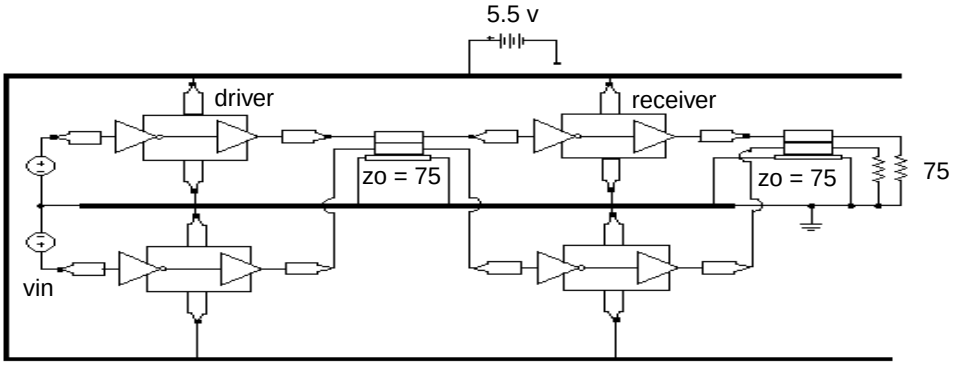
The Signetics I/O buffer library is distributed with Star-Hspice in the *$installdir/parts/signet* directory. These are high-performance parts used in backplane design. The transmission line model used describes two conductors.

Figure 16-10: Planar Transmission Line DLEV=2: Microstrip Sea of Dielectric



In the following application, a pair of drivers are driving about 2.5 inches of adjacent lines to a pair of receivers that drive about four inches of line.

Figure 16-11: I/O Drivers/Receivers with Package Lead Inductance, Parallel 4" Lossy Microstrip Connectors



Example

This is an example of connecting I/O chips with transmission lines.

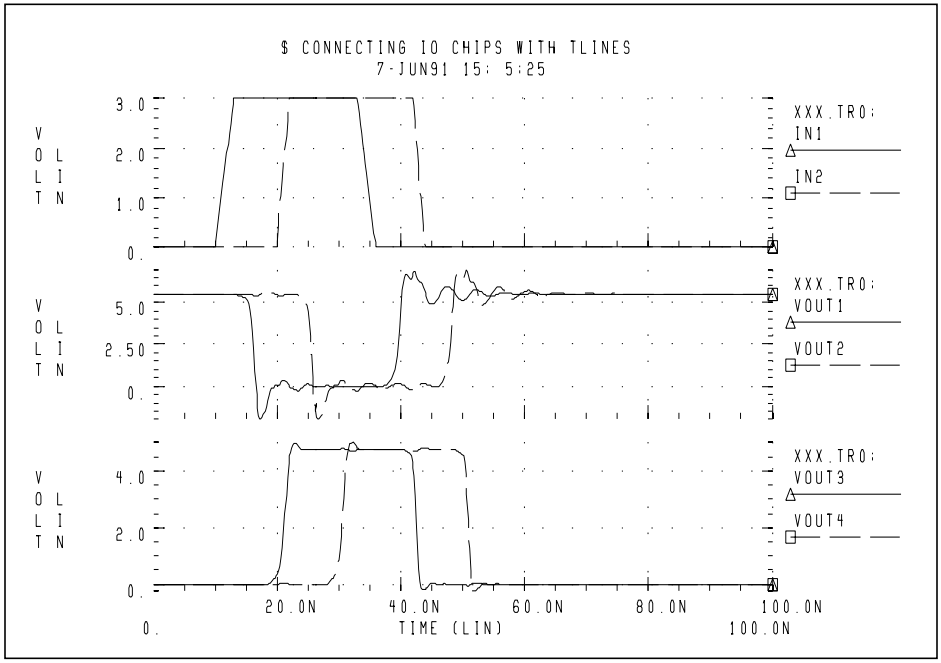
```
.OPTIONS SEARCH='$installdir/parts/signet'
.OPTIONS POST=2 TNOM=27 NOMOD LIST METHOD=GEAR
.TEMP 27

$ DEFINE PARAMETER VALUES
.PARAM LV=0 HV=3 TD1=10n TR1=3n TF1=3n TPW=20n
+ TPER=100n TD2=20n TR2=2n TF2=2n LNGTH=101.6m

$ POWER SUPPLY
VCC VCC 0 DC 5.5

$ INPUT SOURCES
VIN1 STIM1 0 PULSE LV HV TD1 TR1 TF1 TPW TPER
VIN2 STIM2 0 PULSE LV HV TD2 TR2 TF2 TPW TPER

$ FIRST STAGE: DRIVER WITH TLINE
X1ST_TOP STIM1 OUTPIN1 VCC GND IO_CHIP PIN_IN=2.6n
+ PIN_OUT=4.6n
X1ST_DN STIM2 OUTPIN2 VCC GND IO_CHIP PIN_IN=2.9n
+ PIN_OUT=5.6n
U_1ST OUTPIN1 OUTPIN2 GND TLOUT1 TLOUT2 GND USTRIP
+ L=LNGTH
```

```
$ SECOND STAGE: RECEIVER WITH TLINE
X2ST_TOP TLOUT1 OUTPIN3 VCC GND IO_CHIP PIN_IN=4.0n
+ PIN_OUT=2.5n
X2ST_DN TLOUT2 OUTPIN4 VCC GND IO_CHIP PIN_IN=3.6n
+ PIN_OUT=5.1n
U_2ST OUTPIN3 OUTPIN4 GND TLOUT3 TLOUT4 GND USTRIP
+ L=LNGTH

$ TERMINATING RESISTORS
R1 TLOUT3 GND 75
R2 TLOUT4 GND 75

$ IO CHIP MODEL - SIGNETICS
.SUBCKT IO_CHIP IN OUT VCC XGND PIN_VCC=7n PIN_GND=1.8n

X1 IN1    INVOUT VCC1 XGND1 ACTINPUT
X2 INVOUT OUT1   VCC1 XGND1 AC109EQ
```

Package Inductance

```
LIN_PIN IN IN1 PIN_IN
LOUT_PIN OUT1 OUT PIN_OUT
LVCC VCC VCC1 PIN_VCC
LGND XGND1 XGND PIN_GND
.ENDS

$ TLINE MODEL - 2 SIGNAL CONDUCTORS WITH GND
$ PLANE
.MODEL USTRIP U LEVEL=3 ELEV=1 PLEV=1
+ TH1=1.3mil HT1=10mil TS=32mil KD1=4.5 DLEV=0 WD1=8mil
+ XW=-2mil KD2=4.5 NL=2 SP12=5mil

$ ANALYSIS / PRINTS
.TRAN .1NS 100NS
.GRAPH IN1=V(STIM1) IN2=V(STIM2) VOUT1=V(TLOUT1)
+ VOUT2=V(TLOUT2)
.GRAPH VOUT3=V(TLOUT3) VOUT4=V(TLOUT4)

.END
```

Figure 16-12: Connecting I/O Chips with Transmission Lines



Simulating Circuits with Xilinx FPGAs

Avant!, in conjunction with Xilinx, maintains a library of Star-Hspice transistor level sub-circuits for the 3000 and 4000 series Field Programmable Gate Arrays (FPGAs). These sub-circuits model the input and output buffer.

The following simulations use the Xilinx input/output buffer (*xil_iob.inc*) to simulate the ground bounce effects for the 1.08µm process at room temperature and nominal model conditions. The IOB and IOB4 are parameterized Star-Hspice sub-circuits that allow you to specify:

- Local temperature
- Fast/slow/typical speed selections
- Technology 1.2µ/1.08µ

These choices allow the system designer to perform a variety of simulations to measure:

- Ground bounce as a function of package, temperature, part speed and technology
- Coupled noise, both on-chip and chip-to-chip
- Full transmission line effects at the package and printed circuit board levels
- Peak current and instantaneous power consumption for power supply bussing considerations and chip capacitor placement

Syntax for IOB (xil_iob) and IOB4 (xil_iob4)

```
* EXAMPLE OF CALL FOR 1.2U PART:
* X1 I O PAD TS FAST PPUB TTL VDD GND XIL_IOB
*+ XIL_SIG=0 XIL_DTEMP=0 XIL_SHRINK=0

* EXAMPLE OF CALL FOR 1.08U PART:
* X1 I O PAD TS FAST PPUB TTL VDD GND XIL_IOB
*+ XIL_SIG=0 XIL_DTEMP=0 XIL_SHRINK=1
```

Nodes	Description
I (IOB only)	output of the TTL/CMOS receiver
O (IOB only)	input pad driver stage

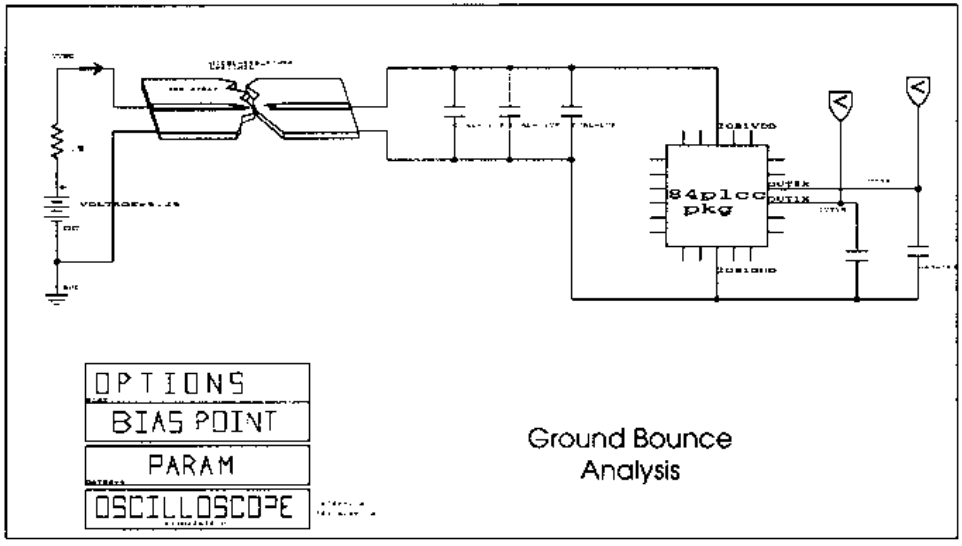
Nodes	Description
I1 (IOB4 only)	input data 1
I2 (IOB4 only)	input data 2
DRIV_IN (IOB4 only)	
PAD	bonding pad connection
TS	three-state control input (5 V disables)
FAST	slew rate control (5 V fast)
PPUB (IOB only)	pad pull-up enable (0 V enables)
PUP (IOB4 only)	pad pull-up enable (0 V enables)
PDOWN (IOB4 only)	pad pull-up enable (5 V enables)
TTL (IOB only)	CMOS/TTL input threshold (5 V selects TTL)
VDD	5 volt supply
GND	ground
Parameters	Description
XIL_SIG	model distribution: (default 0) -3==> slow 0==> typical +3==> fast
XIL_DTEMP	Buffer temperature difference from ambient The default = 0 degrees if ambient is 25 degrees and the buffer is 10 degrees hotter than XIL_DTEMP=10.
XIL_SHRINK	Old or new part; (default is new) ■ 0==>old ■ 1==>new

All grounds and supplies are common to the external nodes for ground and VDD. In Star-Hspice, you can redefine grounds, to add package models.

Ground Bounce Simulation

The ground bounce simulation presented duplicates Xilinx internal measurements methods; 8 to 32 outputs are simultaneously toggled. Each output is loaded with a 56-pF capacitance. The simulation uses an 84-pin package mode and an output buffer held at chip ground to measure the internal ground bounce.

Figure 16-13: Ground Bounce Simulation



The simulation model is adjusted for the oscilloscope recordings for the two-bond wire ground.

Star-Hspice Input File for Ground Bounce

```
qabounce.sp test of xilinx   i/o buffers

* The following is the netlist for the above schematic
* (Figure 16-13)

.op
.option post list
.tran 1ns 50ns sweep gates 8 32 4
.measure bounce max v(out1x)
*.tran .1ns 7ns
.param gates=8
.print v(out1x) v(out8x) i(vdd) power

$.param xil_dtemp=-65 $ -40 degrees c
$ (65 degrees from +25 degrees)
vdd vdd gnd 5.25
vgnd return gnd 0

upower1 vdd return   iob1vdd iob1gnd   pcb_power
+ L=600mil

* local power supply capacitors
xc1a iob1vdd iob1gnd cap_mod cval=.1u
xc1b iob1vdd iob1gnd cap_mod cval=.1u
xc1c iob1vdd iob1gnd cap_mod cval=1u
xgnd_b   iob1vdd iob1gnd out8x out1x xil_gnd_test
xcout8x out8x iob1gnd cap_mod m=gates
xcout1x out1x iob1gnd cap_mod m=1

.model pcb_power u LEVEL=3 elev=1 plev=1 nl=1 llev=1
+ th=1.3mil ht=10mil kd=4.5 dlev=1 wd=500mil xw=-2mil

.macro cap_mod node1 node2 cval=56p
Lr1 node1 node1x L=2nh R=0.05
cap node1x node2x c=cval
Lr2 node2x node2 L=2nh R=0.05
.eom

.macro xil_gnd_test vdd gnd outx outref
+ gates=8
* example of 8 iobuffers simultaneously switching
* through approx. 4nh lead inductance
* 1 iob is active low for ground bounce measurements

vout drive chipgnd pwl 0ns 5v, 10ns 5v, 10.5ns 0v,
$+ 20ns 0v, 20.5ns 5v, 40ns 5v R
```

```
x8 I8 drive PAD8x TS FAST PPUB TTL chipvdd chipgnd
+ xil_iob xil_sig=0 xil_dtemp=0 xil_shrink=1 M=gates
x1 I1 gnd PAD1x TS FAST PPUB TTL chipvdd chipgnd
+ xil_iob xil_sig=0 xil_dtemp=0 xil_shrink=1 m=1
```

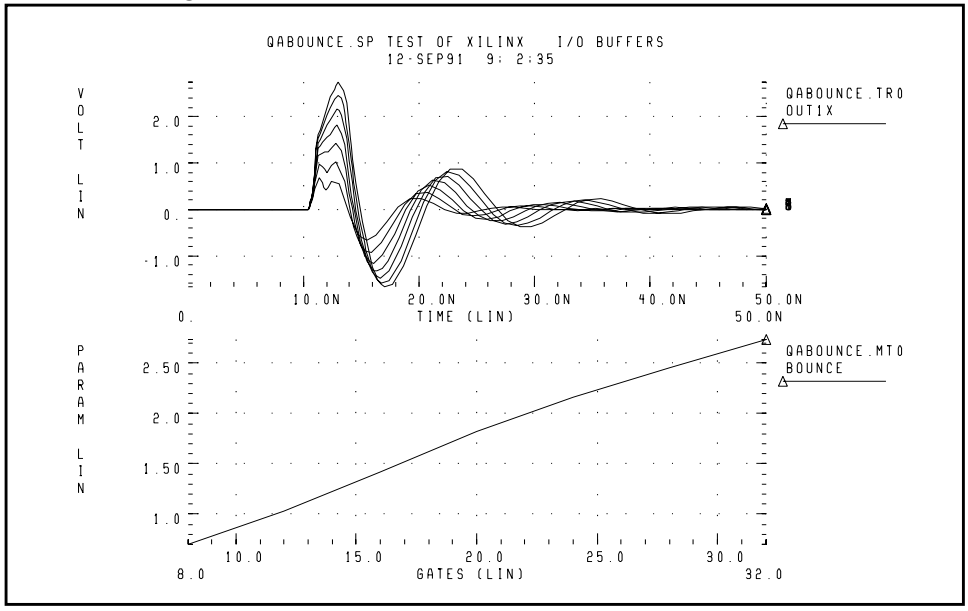
Control Settings

```
rts    ts    chipgnd 1
rfast  fast  chipvdd 1
rppub  ppub  chipgnd 1
rttl   ttl   chipvdd 1
```

```
* pad model plcc84 rough estimates
lvdd  vdd chipvdd L=3.0nh r=.02
lgnd  gnd chipgnd L=3.0nh r=.02
lout8x  outx pad8x L='5n/gates' r='0.05/gates'
lout1x  outref pad1x L=5nh r=0.05
c_vdd_gnd  chipvdd chipgnd 100n
```

```
.eom
.end
```

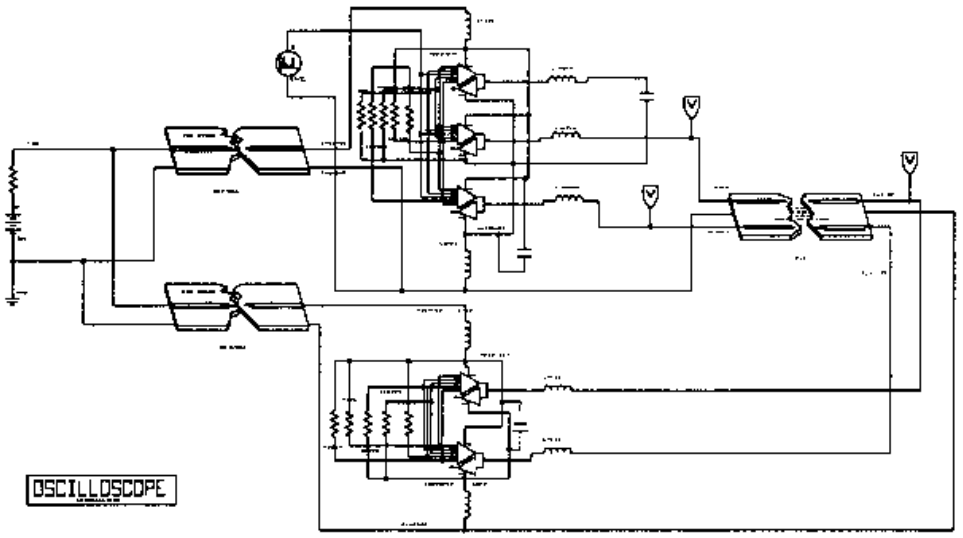
Figure 16-14: Results of Ground Bounce Simulation



Coupled Line Noise

This example uses coupled noise to separate IOB parts. The output of one part drives the input of the other part through 0.6 inch of PCB. The example also monitors an adjacent quiet line.

Figure 16-15: Coupled Noise Simulation



Coupled Noise

Star-Hspice Input File for

```
qa8.sp test of xilinx 0.8u i/o buffers

* The following is the netlist for the above schematic
* (Figure 16-15).

.op
.option nomod post=2

*.tran .1ns 5ns sweep xil_sig -3 3 3
.tran .1ns 15ns
.print v(out1x) v(out3x) i(vdd) v(irec)

vdd vdd gnd 5
vgnd return gnd 0
```

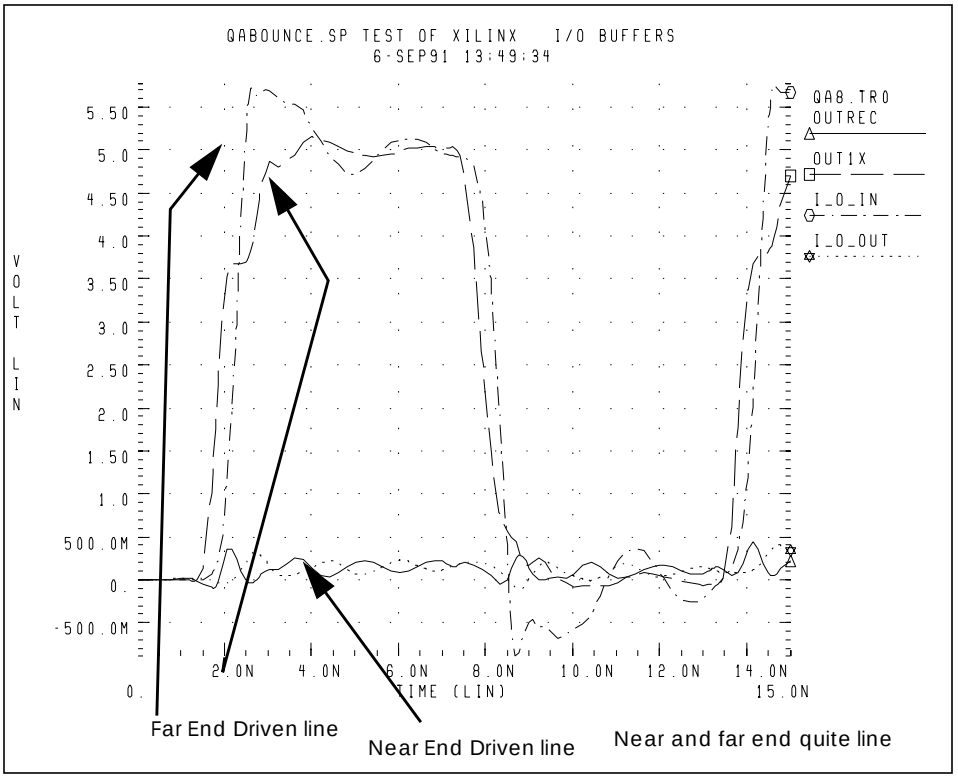
```
upower1 vdd return   iob1vdd iob1gnd   pcb_power
+ L=600mil
upower2 vdd return   iob2vdd iob2gnd   pcb_power
+ L=600mil

x4io   iob1vdd iob1gnd out3x out1x outrec irec xil_iob4
cout3x out3x iob1gnd 9pf
u1x out1x outrec iob1gnd i_o_in i_o_out iob2gnd
+ pcb_top L=2000mil
xrec iob2vdd iob2gnd i_o_in i_o_out xil_rec

.ic i_o_out 0v
.model pcb_top u LEVEL=3 elev=1 plev=1 nl=2 llev=1
+ th=1.3mil ht=10mil sp=5mil kd=4.5 dlev=1 wd=8mil
+ xw=-2mil

.model pcb_power u LEVEL=3 elev=1 plev=1 nl=1 llev=1
+ th=1.3mil ht=10mil kd=4.5 dlev=1 wd=500mil xw=-2mil
.macro xil_rec vdd gnd tri1 tri2
* example of 2 iobuffers in tristate

xtri1 Irec O pad_tri1 TSrec FAST PPUB TTL
+ chipvdd chipgnd xil_iob xil_sig=0 xil_dtemp=0
+ xil_shrink=1 m=1
xtri2 Irec O pad_tri2 TSrec FAST PPUB TTL
+ chipvdd chipgnd xil_iob xil_sig=0 xil_dtemp=0
+ xil_shrink=1 m=1
```

Control Setting

```
rin_output    O    chipgnd 1
rtsrec tsrec chipvdd 1
rfast  fast  chipvdd 1
rppub  ppub  chipgnd 1
rttl   ttl   chipvdd 1

* pad model plcc84 rough estimates
lvdd  vdd chipvdd L=1nh r=.01
lgnd  gnd chipgnd L=1nh r=.01
ltri1 tri1 pad_tri1 L=3nh r=0.01
ltri2 tri2 pad_tri2 L=3nh r=.01
c_vdd_gnd chipvdd chipgnd 100n
.eom

.macro xil_iob4 vdd gnd out3x out1x outrec Irec
* example of 4 iobuffers simultaneously switching
* through approx. 3nh lead inductance
* 1 iob is a receiver (tristate)
```

```
vout O chipgnd pwl 0ns 0v, 1ns 0v, 1.25ns 4v, 7ns 4v,
+ 7.25ns 0v, 12ns 0v R

x3 I3 O PAD3x TS FAST PPUB TTL
+ chipvdd chipgnd xil_iob
+ xil_sig=0 xil_dtemp=0 xil_shrink=1 m=3

x1 I1 O PAD1x TS FAST PPUB TTL
+ chipvdd chipgnd xil_iob
+ xil_sig=0 xil_dtemp=0 xil_shrink=1 m=1

xrec Irec O PADrec TSrec FAST PPUB TTL
+ chipvdd chipgnd xil_iob
+ xil_sig=0 xil_dtemp=0 xil_shrink=1 m=1

* control settings
rts    ts    chipgnd 1
rtsrec tsrec chipvdd 1
rfast  fast  chipvdd 1
rppub  ppub  chipgnd 1
rttl   ttl   chipvdd 1

* pad model plcc84 rough estimates
lvdd  vdd chipvdd L=1nh r=.01
lgnd  gnd chipgnd L=1nh r=.01
lout3x  out3x pad3x L=1nh r=.0033
lout1x  out1x pad1x L=4nh r=0.01
loutrec outrec padrec L=4nh r=.01
c_vdd_gnd chipvdd chipgnd 100n

.eom
.end
```

Figure 16-16: Results of Coupled Noise Simulation



IOB Buffer Module

```
* INPUT/OUTPUT BLOCK MODEL
* PINS:
*       I     OUTPUT OF THE TTL/CMOS INPUT RECEIVER.
*       O     INPUT TO THE PAD DRIVER STAGE.
*       PAD   BONDING PAD CONNECTION.
*       TS    THREE-STATE CONTROL INPUT. HIGH LEVEL
*             DISABLES PAD DRIVER.
*       FAST SLEW RATE CONTROL. HIGH LEVEL SELECTS
*             FAST SLEW RATE.
*       PPUB  PAD PULLL-UP ENABLE. ACTIVE LOW.
```

```
*       TTL   CMOS/TTL INPUT THRESHOLD SELECT. HIGH
*             SELECTS TTL.
*       VDD   POSITIVE SUPPLY CONNECTION FOR INTERNAL
*             CIRCUITRY.
*   ALL THE ABOVE SIGNALS ARE REFERENCED TO NODE 0.
*   THIS MODEL DOES CAUSE SOME DC CURRENT TO FLOW
*   INTO NODE 0 THAT IS AN ARTIFACT OF THE MODEL.
*       GND   CIRCUIT GROUND
```

Description

```
* THIS SUBCIRCUIT MODELS THE INTERFACE BETWEEN XILINX
* 3000 SERIES PARTS AND THE BONDING PAD. IT IS NOT
* USEFUL FOR PREDICTING DELAY TIMES FROM THE OUTSIDE
* WORLD TO INTERNAL LOGIC IN THE XILINX CHIP. RATHER,
* IT CAN BE USED TO PREDICT THE SHAPE OF WAVEFORMS
* GENERATED AT THE BONDING PAD AS WELL AS THE RESPONSE
* OF THE INPUT RECEIVERS TO APPLIED WAVEFORMS.
* THIS MODEL IS INTENDED FOR USE BY SYSTEM DESIGNERS
* WHO ARE CONCERNED ABOUT TRANSMISSION EFFECTS IN
* CIRCUIT BOARDS CONTAINING XILINX 3000 SERIES PARTS.
* THE PIN CAPACITANCE AND BONDING WIRE INDUCTANCE,
* RESISTANCE ARE NOT CONTAINED IN THIS MODEL. THESE
* ARE A FUNCTION OF THE CHOSEN PACKAGE AND MUST BE
* INCLUDED EXPLICITLY IN A CIRCUIT BUILT WITH THIS
* SUBCIRCUIT.
* NON-IDEALITIES SUCH AS GROUND BOUNCE ARE ALSO A
* FUNCTION OF THE SPECIFIC CONFIGURATION OF THE
* XILINX PART, SUCH AS THE NUMBER OF DRIVERS WHICH
* SHARE POWER PINS SWITCHING SIMULTANEOUSLY. ANY
* SIMULATION TO EXAMINE THESE EFFECTS MUST ADDRESS
* THE CONFIGURATION-SPECIFIC ASPECTS OF THE DESIGN.
*
.SUBCKT XIL_IOB I O PAD_IO TS FAST PPUB TTL VDD GND
+ XIL_SIG=0 XIL_DTEMP=0 XIL_SHRINK=1
.prot FREELIB
;]= $.[;qW.261DW3Eu0
VO\;:n[ $.[;qW.2’4%S+%X;:0[(3’1:67*8-:1:\[
kp39H2J9#Yo%XpVY#O!rDI$UqhmE%:\7%(3e%:\7\5O)1-5i# ;
.ENDS XIL_IOB
```

Avant!

Chapter 17
Performing Behavioral Modeling

Behavioral modeling refers to the substitution of 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, leading to substantial reduction in the actual simulation time per circuit. This reduction in elapsed time per simulation, when considering the whole of the design and simulation cycle, can lead to a tremendous increase in design efficiency, as well as possible reduction in the time necessary to take a design from a concept to a marketable product.

This chapter describes how to create behavioral models in the following topics:

Understanding the Behavioral Design Process

Star-Hspice provides specific modeling elements that promote the use of behavioral and mixed signal techniques. These models include controllable sources that may be configured to emulate op-amps, single- or multi-input logic gates, or any system with a continuous algebraic transfer function. These functions may be in algebraic form or in the form of coordinate pairs. Digital stimulus files are useful features that allow you to enter a number of logic waveforms into the simulation deck without resorting to the awkward procedure of entering digital waveforms using piecewise linear sources. You can define clock rise times, fall times, periods, and voltage levels.

The typical design cycle of a circuit or system using Star-Hspice behavioral models is described below.

- Perform full simulation of a subcircuit with pertinent inputs, characterizing its transfer functions.
- Determine which of the Star-Hspice elements, singularly or in combination, accurately describe the transfer function.
- Reconfigure the subcircuit appropriately.
- After the behavioral model is verified, substitute the model into the larger system in place of the lower level subcircuit.

Using Behavioral Elements

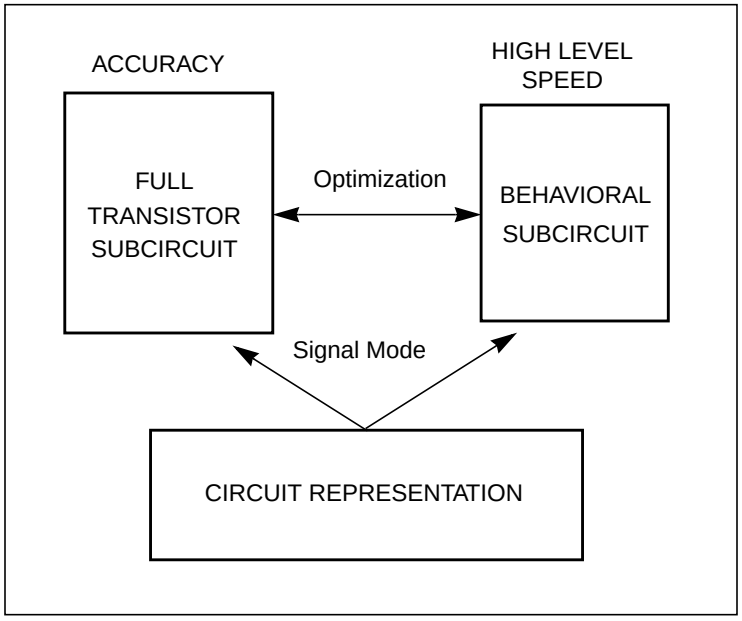
Behavioral elements offer a higher level of abstraction and a faster processing time over the lower level description of an analog function. For system-level designers, function libraries of sub-circuits containing these elements are used to describe parts such as op-amps, vendor specific output buffer drivers, TTL drivers, logic-to-analog and analog-to-logic simulator converters. For the integrated circuit designer, these elements offer a fast representation that is particularly useful in filter and signal processing design.

Behavioral elements are based on using an arbitrary algebraic equation as a transfer function to a voltage (E) or current (G) source. This function can include nodal voltages, element currents, time, or user defined parameters. 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)'
```

Sub-circuits provide a way to encapsulate a function. If you split the function definition from the use, you create a 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 deal with mixed signal functions within Star-Hspice. You can calibrate the behavioral elements from a full transistor circuit using the built-in OPTIMIZE function.

Figure 17-1: Netlisting by Signal Mode



Controlled Sources

Controlled sources model both analog and digital circuits at the behavioral level, allowing for fast mixed-signal simulation times and providing a means to model system level operations. Controlled sources model gate switching action for the behavioral modeling of digital circuits. For analog behavioral modeling, the controlled sources can be programmed as mathematical functions that are either linear or nonlinear, dependent on other nodal voltages and branch currents.

Digital Stimulus Files

Complex transition files are difficult to process using the piecewise linear sources. You can use the U Element A2D and D2A conversion functions to simplify processing of transition files. The A2D function converts analog output to digital data, and the D2A function converts digital input data to analog. You can export output to logic or VHDL simulators as well.

Behavioral Examples

The examples of analog and digital elements in this chapter give some insight into how the behavioral elements operate.

Op-Amp Subcircuit Generators

Operational amplifiers are automatically designed using the subcircuit generator to meet given electrical specifications, such as PSRR, CMRR, and Vos. The generator produces component values for each of the elements in the design. The sub-circuits produced by combining these values offer faster simulation times than conventional circuit level implementations.

Libraries

Use the Discrete Device Library of standard industry IC components to model board level designs that contain transistors, diodes, opamps, comparators, converters, IC pins, printed circuit board traces and coaxial cables. You can model drivers and receivers to analyze transmission line effects and power and signal line noise.

Using Voltage and Current Controlled Elements

Star-Hspice uses four voltage and current controlled elements, known as G, E, H and F Elements. Use these controlled elements to model the following:

- MOS and bipolar transistors
- Tunnel diodes
- SCRs

and analog functions such as

- Operational amplifiers
- Summers
- Comparators
- Voltage controlled oscillators
- Modulators
- Switched capacitor circuits

The controlled elements can be either linear or nonlinear functions of controlling node voltages or branch currents, depending on whether you used the polynomial or piecewise linear functions.

The functions of the G, E, F, and H controlled elements are different.

- The G Element can be a voltage or current controlled current source, a voltage controlled resistor, a piecewise linear voltage controlled capacitor, an ideal delay element, or a piecewise linear multi-input AND, NAND, OR, or NOR gate.
- The E Elements can be a voltage or current controlled voltage source, an ideal op-amp, an ideal transformer, an ideal delay element, or a piecewise linear voltage controlled multi-input AND, NAND, OR, or NOR gate.
- The H Element can be a current controlled voltage source, an ideal delay element, or a piecewise linear current controlled multi-input AND, NAND, OR, or NOR gate.
- The F Element can be a current controlled current source, an ideal delay element, or a piecewise linear current controlled multi-input AND, NAND, OR, or NOR gate.

The next section describes polynomial and piecewise linear functions. Element statements for linear or nonlinear functions are described in later sections.

Polynomial Functions

Use the controlled element statement to define the controlled output variable (current, resistance, or voltage) as a polynomial function of one or more voltages or branch currents. You can select three polynomial equations, using the POLY(ndim) parameter in the E, F, G, or H Element statement.

- POLY(1) one-dimensional equation
- POLY(2) two-dimensional equation
- POLY(3) three-dimensional equation

The POLY(1) polynomial equation specifies a polynomial equation as a function of one controlling variable, POLY(2) as a function of two controlling variables, and POLY(3) as a function of three controlling variables.

Along with each polynomial equation are polynomial coefficient parameters (P0, P1 … Pn) that can be set to explicitly define the equation.

One-Dimensional Function

If the function is one-dimensional, that is, a function of one branch current or node voltage, the function value FV is determined by the following expression:

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

FV	the controlled voltage or current from the controlled source
P0. . .Pn	coefficients of polynomial equation
FA	the controlling branch current or nodal voltage

Note: If the polynomial is one-dimensional and exactly one coefficient is specified, Star-Hspice assumes it to be P1 (P0 = 0.0), in order to facilitate the input of linear controlled sources.

The following controlled source statement is an example of a one-dimensional function:

```
E1 5 0 POLY(1) 3 2 1 2.5
```

The above voltage controlled voltage source is connected to nodes 5 and 0. The single dimension polynomial function parameter, POLY(1), informs Star-Hspice that E1 is a function of the difference of one nodal voltage pair, in this case, the voltage difference between nodes 3 and 2, hence 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 E1 is:

$$E1 = 1 + 2.5 \cdot V(3,2)$$

Two-Dimensional Function

Where the function is two-dimensional, that is, a function of two node voltages or two branch currents, FV is determined by the following expression:

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

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 gives the controlled voltage, E1, as:

$$E1 = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2$$

To implement this function, use the following controlled source element statement:

```
E1 1 0 POLY(2) 3 2 7 6 0 3 0 0 0 4
```

This specifies a controlled voltage source connected between nodes 1 and 0 that is controlled by two differential voltages: the voltage difference between nodes 3 and 2 and the 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, and P5=4.

Three-Dimensional Function

For a three-dimensional polynomial function with arguments FA, FB, and FC, the function value FV is determined by the following expression:

$$\begin{aligned} FV = {} & P0 + (P1 \cdot FA) + (P2 \cdot FB) + (P3 \cdot FC) + (P4 \cdot FA^2) \\ & + (P5 \cdot FA \cdot FB) + (P6 \cdot FA \cdot FC) + (P7 \cdot FB^2) + (P8 \cdot FB \cdot FC) \\ & + (P9 \cdot FC^2) + (P10 \cdot FA^3) + (P11 \cdot FA^2 \cdot FB) + (P12 \cdot FA^2 \cdot FC) \\ & + (P13 \cdot FA \cdot FB^2) + (P14 \cdot FA \cdot FB \cdot FC) + (P15 \cdot FA \cdot FC^2) \\ & + (P16 \cdot FB^3) + (P17 \cdot FB^2 \cdot FC) + (P18 \cdot FB \cdot FC^2) \\ & + (P19 \cdot FC^3) + (P20 \cdot FA^4) + \ldots \end{aligned}$$

For example, generate a voltage controlled source that gives the voltage as:

$$E1 = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2 + 5 \cdot V(9,8)^3$$

From the above equation and the three-dimensional polynomial equation:

$$FA = V(3,2)$$

$$FB = V(7,6)$$

$$FC = V(9,8)$$

$$P1 = 3$$

$$P7 = 4$$

$$P19 = 5$$

Substituting 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 above specifies a voltage source connected between nodes 1 and 0, and controlled by three differential voltages: the voltage differences between nodes 3 and 2, between nodes 7 and 6, and between nodes 9 and 8—that is, FA=V(3,2), FB=V(7,6), and FC=V(9,8). The statement gives the polynomial coefficients as P1=3, P7=4, P19=5, and the rest are zero.

Piecewise Linear (PWL) Function

The one-dimensional piecewise linear (PWL) function allows designers to model some special element characteristics, such as those of tunnel diodes, silicon controlled rectifiers. and diode breakdown regions. You can describe the piecewise linear function by specifying measured data points. Although the device characteristic is described by data points, Star-Hspice automatically smooths the corners to ensure derivative continuity and, as a result, better convergence.

A parameter DELTA is provided to control the curvature of the characteristic at the corners. The smaller the DELTA, the sharper the corners are. The maximum value allowed for DELTA is half the smallest of the distances between breakpoints. Specify a DELTA that provides satisfactory sharpness of the function corners. You can specify up to 100 breakpoint pairs. You must specify at least two point pairs (with each point consisting of an x and a y coefficient).

The functions NPWL and PPWL can be used for modeling bidirectional switch or transfer gates using G Elements. The NPWL and PPWL functions behave like NMOS and PMOS transistors.

The piecewise linear function also is used to model multi-input AND, NAND,OR, and NOR gates. In this case only one input determines the state of the output. In AND and NAND gates, the input with the smallest value is used in the piecewise linear function to determine the corresponding output of the gates. In the OR and NOR gates, the input with the largest value is used to determine the corresponding output of the gates.

Voltage-Dependent Voltage Sources — E Elements

Voltage Controlled Voltage Source (VCVS)

The syntax is:

Linear

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

Polynomial

```
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=vals>
```

Piecewise Linear

```
Exxx n+ n- <VCVS> PWL(1) in+ in-    <DELTA=val>
+ <SCALE=val> <TC1=val> <TC2=val> x1,y1 x2,y2  ...
+ x100,y100 <IC=val>
```

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>
```

Delay Element

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

Behavioral Voltage Source

The syntax is:

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

Ideal Op-Amp

The syntax is:

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

Ideal Transformer

The syntax is:

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

E Element Parameters

ABS	Output is absolute value if ABS=1.
DELAY	Keyword for the delay element. The delay element is the same as voltage controlled voltage source, except it is associated by a propagation delay TD.This element facilitates the adjustment of propagation delay in the subcircuit modeling process. DELAY is a Star-Hspice keyword and should not be used as a node name.
DELTA	Used to control the curvature of the piecewise linear corners. Defaults to 1/4 of the smallest of the distances between breakpoints. The maximum is 1/2 of the smallest of the distances between breakpoints.
Exxx	Voltage controlled element name. Must begin with “E”, which may be followed by up to 15 alphanumeric characters.
gain	Voltage gain.
gatetype(k)	May be AND, NAND, OR, or NOR. The value of k is the number of inputs of the gate. The x and y terms represent the piecewise linear variation of output as a function of input. In the multi-input gates only one input determines the state of the output.
IC	Initial condition: the initial estimate of the value(s) of the controlling voltage(s). Default=0.0.

in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
j	Ideal transformer turn ratio: V(in+,in-) = j · V(n+,n-)
MAX	Maximum output voltage value. The default is undefined, and sets no maximum value.
MIN	Minimum output voltage value. The default is undefined, and sets no minimum value.
n+/-	Positive or negative node of controlled element.
NPDELAY	Sets the number of data points to be used in delay simulations. The default value is the larger of 10 or the smaller of TD/tstep and tstop/tstep. That is, $\mathrm{NPDELAY}_{\mathrm{default}} = \max\left[\frac{\min\langle \mathrm{TD}, \mathrm{tstop}\rangle}{\mathrm{tstep}}, 10\right]$ The values of tstep and tstop are specified in the .TRAN statement.
OPAMP	Keyword for ideal op-amp element. OPAMP is a Star-Hspice keyword and should not be used as a node name.
p0, p1 …	Polynomial coefficients. When one coefficient is specified, Star-Hspice assumes it to be p1, with p0=0.0, and the element is linear. When more than one polynomial coefficient is specified by p0, p1, p2, ..., the element is nonlinear. See “Polynomial Functions” on page Chapter 177.
POLY	Polynomial dimension. If POLY(ndim) is not specified, a one-dimensional polynomial is assumed. Ndim must be a positive number.
PWL	Piecewise linear function keyword.
SCALE	Element value multiplier.

TC*1, TC2*	First and second order temperature coefficients. The SCALE is updated by temperature: $SCALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Time delay keyword.
TRANSFORMER	Keyword for ideal transformer. TRANSFORMER is a Star-Hspice keyword and should not be used as a node name.
VCVS	Keyword for voltage controlled voltage source. VCVS is a Star-Hspice keyword and should not be used as a node name.
x1,...	Controlling voltage across nodes in+ and in-. The x values must be in increasing order.
y1,...	Corresponding element values of x.

Example

Ideal Op-Amp

A voltage amplifier with supply limits can be built with the voltage controlled voltage source. The output voltage across nodes 2,3 = v(14,1) * 2. The voltage gain parameter, 2, is also given. The MAX and MIN parameters specify a maximum E1 voltage of 5V and a minimum E1 voltage output of -5V. If, for instance, V(14,1) = -4V, E1 would be set to -5V and not -8V as the equation would produce.

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

A user-defined parameter may be used in the following format to specify a value for polynomial coefficient parameters:

```
.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) and V(17,0). It describes a voltage source with the value:

$$V(13,0) + V(15,0) + V(17,0)$$

This example represents an ideal voltage summer. The three controlling voltages are initialized for a DC operating point analysis to 1.5, 2.0, and 17.25 V, respectively.

```
EX 17 0 POLY(3) 13 0 15 0 17 0 0 1 1 1 IC=1.5,2.0,17.25
```

Polynomial Function

The voltage controlled source can also output a nonlinear function using the one-dimensional polynomial. Since the POLY parameter is not specified, a one-dimensional polynomial is assumed, i.e., a function of one controlling voltage. The equation corresponds to the element syntax. Behavioral equations replace this older method.

```
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

A simple inverter with no delay can be built with a piecewise linear transfer function.

```
Einv out 0 PWL(1) in 0 .7v,5v 1v,0v
```

Ideal Transformer

With the turn ratio 10 to 1, the voltage relationship is V(out)=V(in)/10.

```
Etrans out 0 TRANSFORMER in 0 10
```

Voltage Controlled Oscillator (VCO)

The keyword VOL is used to define a single-ended input that controls the output of a VCO.

In the following example, the frequency of the sinusoidal output voltage at node “out” is controlled by the voltage at node “control”. Parameter “v0” is the DC offset voltage and “gain” is the amplitude. The output is a sinusoidal voltage with a frequency of 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). This equation does not apply if you sweep the control voltage.

Voltage-Dependent Current Sources — G Elements

Voltage Controlled Current Source (VCCS)

The syntax is:

Linear

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

Polynomial

```
Gxxx n+ n- <VCCS> POLY(ndim) in1+ in1- ...
+ <inndim+ inndim-> MAX=val> <MIN=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> <ABS=1>
+ p0 <p1…> <IC=vals>
```

Piecewise Linear

```
Gxxx n+ n- <VCCS> PWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
```

```
Gxxx n+ n- <VCCS> NPWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
```

```
Gxxx n+ n- <VCCS> PPWL(1) in+ in- <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
```

Multi-Input Gates

```
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>
```

Delay Element

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

Behavioral Current Source

The syntax is:

```
Gxxx n+ n- CUR='equation' <MAX>=val> <MIN=val>
```

Voltage Controlled Resistor (VCR)

The syntax is:

Linear

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

Polynomial

```
Gxxx n+ n- VCR POLY(ndim) in1+ in1- ...
+ <inndim+ inndim-> <MAX=val> <MIN=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> p0 <p1…> <IC=vals>
```

Piecewise Linear

```
Gxxx n+ n- VCR PWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>

Gxxx n+ n- VCR NPWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>

Gxxx n+ n- VCR PPWL(1) in+ in- <DELTA=val> <SCALE=val>
+ <M=val> <TC1=val> <TC2=val> x1,y1 x2,y2 ... x100,y100
+ <IC=val> <SMOOTH=val>
```

Multi-Input Gates

```
Gxxx n+ n- VCR gatetype(k) in1+ in1- ... ink+ ink-
+ <DELTA=val> <TC1=val> <TC2=val> <SCALE=val> <M=val>
+ x1,y1 ... x100,y100 <IC=val>
```

Voltage Controlled Capacitor (VCCAP)

The syntax is:

```
Gxxx n+ n- VCCAP PWL(1) in+ in-   <DELTA=val>
+ <SCALE=val> <M=val> <TC1=val> <TC2=val>
+ x1,y1 x2,y2 ... x100,y100 <IC=val> <SMOOTH=val>
```

The two functions NPWL and PPWL allow the interchange of the ‘n+’ and ‘n-’ nodes while keeping the same transfer function. This action can be summarized as follows:

NPWL Function

For node ‘in-’ 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 node ‘in-’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 node ‘in-’connected to ‘n-’;

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

For node ‘in-’connected to ‘n+’;

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

G Element Parameters

ABS	Output is absolute value if ABS=1.
CUR=equation	Current output which flows from n+ to n-. The “equation”, which you define, can be a function of node voltages, branch currents, TIME, temperature (TEMPER), and frequency (HERTZ).

DELAY	Keyword for the delay element. The delay element is the same as voltage controlled current source except it is associated by a propagation delay TD. This element facilitates the adjustment of propagation delay in the subcircuit model process. DELAY is a Star-Hspice keyword and should not be used as a node name.
DELTA	Used to control the curvature of the piecewise linear corners. Defaults to 1/4 of the smallest of the distances between breakpoints. The maximum is 1/2 of the smallest of the distances between breakpoints.
Gxxx	Voltage controlled element name. Must begin with “G”, which may be followed by up to 15 alphanumeric characters.
gatetype(k)	May be AND, NAND, OR, or NOR. The value of k is the number of inputs of the gate. The x and y terms represent the piecewise linear variation of output as a function of input. In the multi-input gates only one input determines the state of the output.
IC	Initial condition. The initial estimate of the value(s) of the controlling voltage(s). If IC is not specified, Default=0.0.
in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
M	Number of replications of the element in parallel
MAX	Maximum current or resistance value. The default is undefined, and sets no maximum value.
MIN	Minimum current or resistance value. The default is undefined, and sets no minimum value.
n+/-	Positive or negative node of controlled element

NPDELAY Sets the number of data points to be used in delay simulations. The default value is the larger of 10 or the smaller of TD/tstep and tstop/tstep. That is,

$$\mathrm{NPDELAY}_{\mathrm{default}} = \max\left[\frac{\min\langle \mathrm{TD}, \mathrm{tstop}\rangle}{\mathrm{tstep}}, 10\right]$$

The values of tstep and tstop are specified in the .TRAN statement.

NPWL Models the symmetrical bidirectional switch or transfer gate, NMOS

p0, p1 … Polynomial coefficients. When one coefficient is specified, Star-Hspice assumes it to be p1, with p0=0.0, and the element is linear. When more than one polynomial coefficient is specified by p0, p1, p2, ..., the element is nonlinear. See “Polynomial Functions” on page Chapter 177.

POLY Polynomial dimension. If POLY(ndim) is not specified, a one-dimensional polynomial is assumed. Ndim must be a positive number.

PWL Piecewise linear function keyword

PPWL Models the symmetrical bidirectional switch or transfer gate, PMOS

SCALE Element value multiplier

SMOOTH For piecewise linear dependent source elements, SMOOTH selects the curve smoothing method.

A curve smoothing method simulates exact data points you provide. This method can be used to make Star-Hspice simulate specific data points that correspond to measured data or data sheets.

	Choices for SMOOTH are 1 or 2. Specifying *1* selects the smoothing method prior to Release H93A. Specifying *2* selects the smoothing method that uses data points you provide. This is the default method starting with release H93A.
TC1,TC2	First- and second-order temperature coefficients. The SCALE is updated by temperature: $SCALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Time delay keyword
transconductance	Voltage to current conversion factor
transfactor	Voltage to resistance conversion factor
VCCAP	Keyword for voltage controlled capacitance element. VCCAP is a Star-Hspice keyword and should not be used as a node name.
VCCS	Keyword for voltage controlled current source. VCCS is a Star-Hspice keyword and should not be used as a node name.
VCR	Keyword for voltage controlled resistor element. VCR is a Star-Hspice keyword and should not be used as a node name.
x1,...	Controlling voltage across nodes in+ and in-. The x values must be in increasing order.
y1,...	Corresponding element values of x

Example

Switch

A voltage controlled resistor represents a basic switch characteristic. The resistance between nodes 2 and 0 varies linearly from 10meg to 1m ohms when voltage across nodes 1 and 0 varies between 0 and 1 volt. Beyond the voltage limits, the resistance remains at 10meg and 1m ohms respectively.

```
Gswitch 2 0 VCR PWL(1) 1 0 0v,10meg 1v,1m
```

Switch-level MOSFET

A switch level n-channel MOSFET can be modelled by the N-piecewise linear resistance switch. The resistance value does not change when the node d and s positions are switched.

```
Gnmos d s VCR NPWL(1) g s LEVEL=1 0.4v,150g 1v,10meg
+ 2v,50k 3v,4k 5v,2k
```

Voltage Controlled Capacitor

The capacitance value across nodes (out,0) varies linearly from 1p to 5p when voltage across nodes (ctrl,0) varies between 2v and 2.5v. Beyond the voltage limits, the capacitance value remains constant at 1 picofarad and 5 picofarads respectively.

```
Gcap out 0 VCCAP PWL(1) ctrl 0 2v,1p 2.5v,5p
```

Zero Delay Gate

A two-input AND gate can be implemented using an expression and a piecewise linear table. The inputs are voltages at nodes a and b, and the output is the current flow from node out to 0. The current is multiplied by the SCALE value, which, in this example, is specified as the inverse of the load resistance connected across the nodes (out,0).

```
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. A transmission line, on the other hand, has an infinite frequency response. A glitch input to a G delay is attenuated similarly to a buffer circuit. In this example, the output of the delay element is the current flow from node *out* to node *1* with a value equal to the voltage across nodes (*in*, *0*) multiplied by SCALE value and delayed by TD value.

```
Gdel out 0 DELAY in 0 TD=5ns SCALE=2 NPDELAY=25
```

Diode Equation

A forward bias diode characteristic from node 5 to ground can be modelled with a run time expression. The saturation current is 1e-14 amp, and the thermal voltage is 0.025v.

```
Gdio 5 0 CUR='1e-14*(EXP(V(5)/0.025)-1.0)'
```

Diode Breakdown

Diode breakdown region to forward region can be modelled. When voltage across diode goes beyond 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 the following voltage controlled current sources implement a basic triode. The first uses the poly(2) operator to multiply the anode and grid voltages together and scale by .02. The next example uses the explicit behavioral algebraic description.

```
gt i_anode cathode poly(2) anode,cathode grid,cathode
+ 0 0 0 0 .02 gt i_anode cathode
+ cur='20m*v(anode,cathode) v(grid,cathode)'
```

Current-Dependent Voltage Sources – H Elements

Current Controlled Voltage Source (CCVS)

The syntax is:

Linear

```
Hxxx n+ n- <CCVS> vn1 transresistance <MAX=val>
+ <MIN=val> <SCALE=val> <TC1=val> <TC2=val> <ABS=1>
+ <IC=val>
```

Polynomial

```
Hxxx n+ n- <CCVS> POLY(ndim) vn1 <... vnndim>
+ <MAX=val>MIN=val> <TC1=val> <TC2=val> <SCALE=val>
+ <ABS=1> p0 <p1…> <IC=vals>
```

Piecewise Linear

```
Hxxx n+ n- <CCVS> PWL(1) vn1 <DELTA=val> <SCALE=val>
+ <TC1=val> <TC2=val> x1,y1  ...   x100,y100 <IC=val>
```

Multi-Input Gates

```
Hxxx n+ n- gatetype(k) vn1, ... vnk <DELTA=val>
+ <SCALE=val> <TC1=val> <TC2=val> x1,y1  ...
+ x100,y100 <IC=val>
```

Delay Element

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

Note: E Elements with algebraics make CCVS elements obsolete. However, CCVS elements may still be used for the sake of backward compatibility.

H Element Parameters

ABS	Output is absolute value if ABS=1.
CCVS	Keyword for current controlled voltage source. CCVS is a Star-Hspice keyword and should not be used as a node name.
DELAY	Keyword for the delay element. The delay element is the same as a current controlled voltage source except it is associated by a propagation delay TD.This element facilitates the adjustment of propagation delay in the subcircuit model process. DELAY is a Star-Hspice keyword and should not be used as a node name.
DELTA	Used to control the curvature of the piecewise linear corners. Defaults to 1/4 of the smallest of the distances between breakpoints. The maximum is 1/2 of the smallest of the distances between breakpoints.
gatetype(k)	May be AND, NAND, OR, or NOR. The value of k is the number of inputs of the gate. The x and y terms represent the piecewise linear variation of output as a function of input. In the multi-input gates only one input determines the state of the output.
Hxxx	Current controlled voltage source element name. Must begin with “H”, which may be followed by up to 15 alphanumeric characters.
IC	Initial condition. This is the initial estimate of the value(s) of the controlling current(s) in amps. Default=0.0.
MAX	Maximum voltage value. The default is undefined, which sets no maximum value.
MIN	Minimum voltage value. The default is undefined, which sets no minimum value.
n+/-	Positive or negative controlled source connecting nodes.

NPDELAY Sets the number of data points to be used in delay simulations. The default value is the larger of 10 or the smaller of TD/tstep and tstop/tstep. That is,

$$\mathrm{NPDELAY}_{\mathrm{default}} = \max\left[\frac{\min\langle \mathrm{TD}, \mathrm{tstop}\rangle}{\mathrm{tstep}}, 10\right]$$

The values of tstep and tstop are specified in the .TRAN statement.

p0, p1 . . . The polynomial coefficients. When one coefficient is specified, Star-Hspice assumes it to be p1, with p0=0.0, and the element is linear. When more than one polynomial coefficient is specified by p0, p1, p2, ..., the element is nonlinear. See “Polynomial Functions” on page Chapter 177.

POLY Polynomial dimension. If POLY(ndim) is not specified, a one-dimensional polynomial is assumed. Ndim must be a positive number.

PWL Piecewise linear function keyword.

SCALE Element value multiplier.

TC1, TC2 First and second order temperature coefficients. The SCALE is updated by temperature:

$$\mathrm{SCALEeff} = \mathrm{SCALE} \cdot (1 + \mathrm{TC1} \cdot \Delta t + \mathrm{TC2} \cdot \Delta t^2)$$

TD Time delay keyword.

transresistance Current to voltage conversion factor.

vn1… Names of voltage sources through which controlling current flows. One name must be specified for each dimension.

x1,... Controlling current through vn1 source. The x values must be in increasing order.

y1,... Corresponding output voltage values of x.

Example

```
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. The defining equation of the CCVS is:

$$HX = 1000 \cdot VCUR$$

The defining equation states that the voltage output of HX is 1000 times the value of current flowing through CUR. If the equation produces a value of HX greater than +10V or less than -10V, HX, because of the MAX= and MIN= parameters, would be set to either 10V or -10V, respectively. CUR is the name of the independent voltage source that the controlling current flows through. If the controlling current does not flow through an independent voltage source, a dummy independent voltage source must be inserted.

```
.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, and P4=1. The controlling current for flowing through VIN1 is initialized at.5mA. For VIN2, the initial current is 1.3mA.

The direction of positive controlling current flow is from the positive node, through the source, to the negative node of vnam (linear). The polynomial (nonlinear) specifies the source voltage as a function of the controlling current(s).

Current-Dependent Current Sources — F Elements

Current Controlled Current Source (CCCS)

The syntax is:

Linear

```
Fxxx n+ n- <CCCS> vn1 gain <MAX=val> <MIN=val>
+ <SCALE=val> <TC1=val> <TC2=val> <M=val> <ABS=1>
+ <IC=val>
```

Polynomial

```
Fxxx n+ n- <CCCS> POLY(ndim) vn1 <... vnndim> <MAX=val>
+ <MIN=val> <TC1=val> <TC2=val> <SCALE=vals> <M=val>
+ <ABS=1> p0 <p1…> <IC=vals>
```

Piecewise Linear

```
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>
```

Delay Element

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

Note: G Elements with algebraics make CCCS elements obsolete. However, CCCS elements may still be used for backward compatibility with existing designs.

F Element Parameters

ABS	Output is absolute value if ABS=1.
CCCS	Keyword for current controlled current source. CCCS is a Star-Hspice keyword and should not be used as a node name
DELAY	Keyword for the delay element. The delay element is the same as a current controlled current source except it is associated by a propagation delay TD.This element facilitates the adjustment of propagation delay in the subcircuit model process. DELAY is a Star-Hspice keyword and should not be used as a node name.
DELTA	Used to control the curvature of the piecewise linear corners. Defaults to 1/4 of the smallest of the distances between breakpoints. The maximum is 1/2 of the smallest of the distances between breakpoints.
Fxxx	Current controlled current source element name. Must begin with “F”, followed by up to 15 alphanumeric characters.
gain	Current gain.
gatetype(k)	Can be AND, NAND, OR, or NOR. The value of k is the number of inputs of the gate. The x and y terms represent the piecewise linear variation of output, as a function of input. In the multi-input gates, only one input determines the state of the output. Do not use the above keywords as node names.
IC	Initial condition: the initial estimate of the value(s) of the controlling current(s) in amps. Default=0.0.
M	Number of replications of the element in parallel.
MAX	Maximum output current value. The default is undefined, and sets no maximum value.
MIN	Minimum output current value. The default is undefined, and sets no minimum value.
n+/-	Positive or negative controlled source connecting nodes.

NPDELAY Sets the number of data points to be used in delay simulations. The default value is the larger of 10 or the smaller of TD/tstep and tstop/tstep. That is,

$$\mathrm{NPDELAY}_{\mathrm{default}} = \max\left[\frac{\min\langle \mathrm{TD}, \mathrm{tstop}\rangle}{\mathrm{tstep}}, 10\right]$$

The values of tstep and tstop are specified in the .TRAN statement.

p0, p1 … The polynomial coefficients. When one coefficient is specified, Star-Hspice assumes it to be p1, with p0=0.0, and the element is linear. When more than one polynomial coefficient is specified by p0, p1, p2, ..., the element is nonlinear. See “Polynomial Functions” on page Chapter 177.

POLY Polynomial dimension. If POLY(ndim) is not specified, a one-dimensional polynomial is assumed. Ndim must be a positive number.

PWL Piecewise linear function keyword.

SCALE Element value multiplier.

TC1, TC2 First and second order temperature coefficients. The SCALE is updated by temperature:

$$SCALEeff = SCALE \cdot (1 + \mathrm{TC1} \cdot \Delta t + \mathrm{TC2} \cdot \Delta t^2)$$

TD Time delay keyword.

vn1… Names of voltage sources through which the controlling current flows. Specify one name for each dimension.

x1,... Controlling current through vn1 source. The x values must be in increasing order.

y1,... Corresponding output current values of x.

Example

```
$ Current controlled current sources - F Elements,
F1 13 5 VSENS MAX=+3 MIN=-3 5
F2 12 10 POLY VCC 1MA 1.3M
Fd 1 0 DELAY vin TD=7ns SCALE=5
Filim 0 out PWL(1) vsrc -1a,-1a 1a,1a
```

The first example describes a current controlled current source connected between nodes 13 and 5. The current that controls the value of the controlled source flows through the voltage source named VSENS (to use a current controlled current source, a dummy independent voltage source is often placed into the path of the controlling current). The defining equation is:

$$I(F1) = 5 \cdot I(VSENS)$$

The current gain is 5, the maximum current flow through F1 is 3 A, and the minimum current flow is -3 A. If I(VSENS) = 2 A, I(F1) would be set to 3 amps and not 10 amps as would be suggested by the equation. A user-defined parameter may be specified for the polynomial coefficient(s), as shown below.

```
.PARAM VU = 5
F1 13 5 VSENS MAX=+3 MIN=-3 VU
```

The second example describes a current controlled current source with the value:

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

Current flow is from the positive node through the source to the negative node. The direction of positive controlling current flow is from the positive node through the source to the negative node of vnam (linear), or to the negative node of each voltage source (nonlinear).

The third example is a delayed current controlled current source.

The fourth example is a piecewise linear current controlled current source.

Modeling with Digital Behavioral Components

This section shows how to model using digital behavioral components.

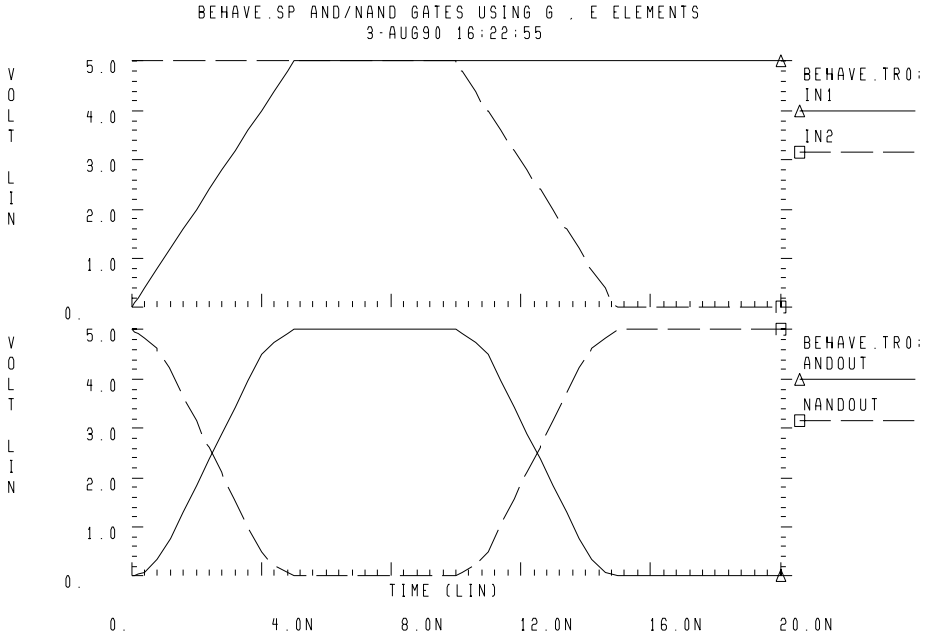
Behavioral AND and NAND Gates

In this example, a two-input AND gate is modeled by a G Element. A two-input NAND gate is modeled by an E Element.

Example

```
behave.sp and/nand gates using g, e Elements
.options post=2
.op
.tran .5n 20n
.probe v(in1) v(in2) v(andout) v(in1) v(in2) v(nandout)
g 0 andout and(2) in1 0 in2 0
+ 0.0 0.0ma
+ 0.5 0.1ma
+ 1.0 0.5ma
+ 4.0 4.5ma
+ 4.5 4.8ma
+ 5.0 5.0ma
*
e nandout 0 nand(2) in1 0 in2 0
+ 0.0 5.0v
+ 0.5 4.8v
+ 1.0 4.5v
+ 4.0 0.5v
+ 4.5 0.2v
+ 5.0 0.0v
*
vin1 in1 0 0 pwl(0,0 5ns,5)
vin2 in2 0 5 pwl(0,5 10ns,5 15ns,0)
rin1 in1 0 1k
rin2 in2 0 1k
rand andout 0 1k
rnand nandout 0 1k
.end
```

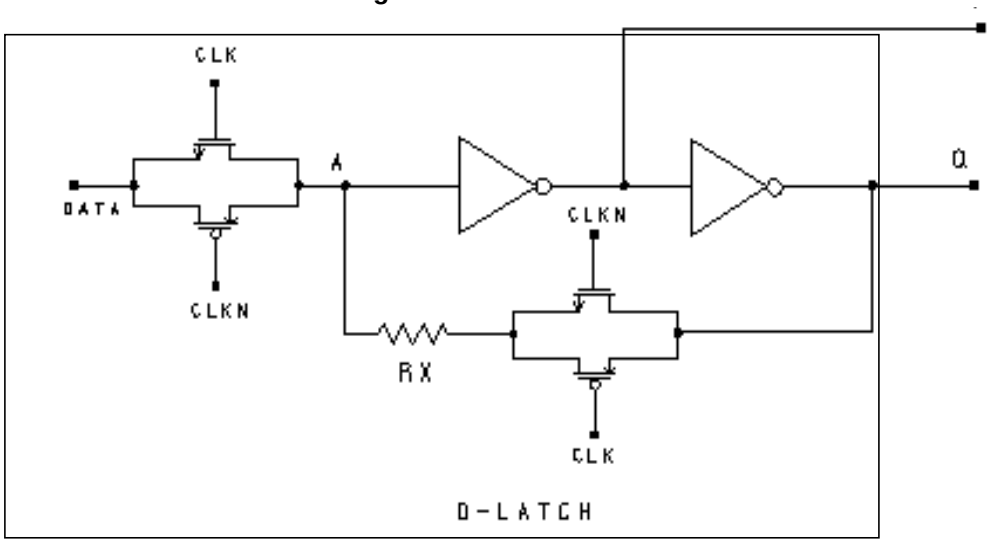
Figure 17-2: NAND/AND Gates



Behavioral D-Latch

In this example, a D flip-flop is modeled by one input NAND gates and NPWL/ PPWL functions.

Figure 17-3: D-Latch Circuit



Example

```
dlatch.sp--- cmos d-latch
.option post
.tran .2n 60ns
.probe tran clock=v(clck)data=v(d) q=v(q)
.ic v(q)=0
```

Waveforms

```
vdata d 0 pulse(0,5 2n,1n,1n 19n,40n)
vclk clck 0 pulse(0,5 7n,1n,1n 10n,20n)
vclkn clckn 0 pulse(5,0 7n,1n,1n 10n,20n)
xdlatch d clck clckn q qb dlatch cinv=.2p
```

Subcircuit Definitions for Behavioral N-Channel MOSFET

```
* DRAIN GATE SOURCE
.SUBCKT nmos 1 2 3 capm=.5p
cd 1 0 capm
cs 3 0 capm
gn 3 1 VCR NPWL(1) 2 3
+ 0. 495.8840G
+ 200.00000M 456.0938G
+ 400.00000M 141.6902G
+ 600.00000M 7.0624G
+ 800.00000M 258.9313X
+ 1.00000 6.4866X
+ 1.20000 842.9467K
+ 1.40000 321.6882K
+ 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
+ 3.20000 3.8407K
+ 3.40000 3.4950K
+ 3.60000 3.2441K
+ 3.80000 3.0534K
+ 4.00000 2.9042K
+ 4.20000 2.7852K
+ 4.40000 2.6822K
+ 4.60000 2.5k
+ 5.0 2.3k
.ENDS nmos
```

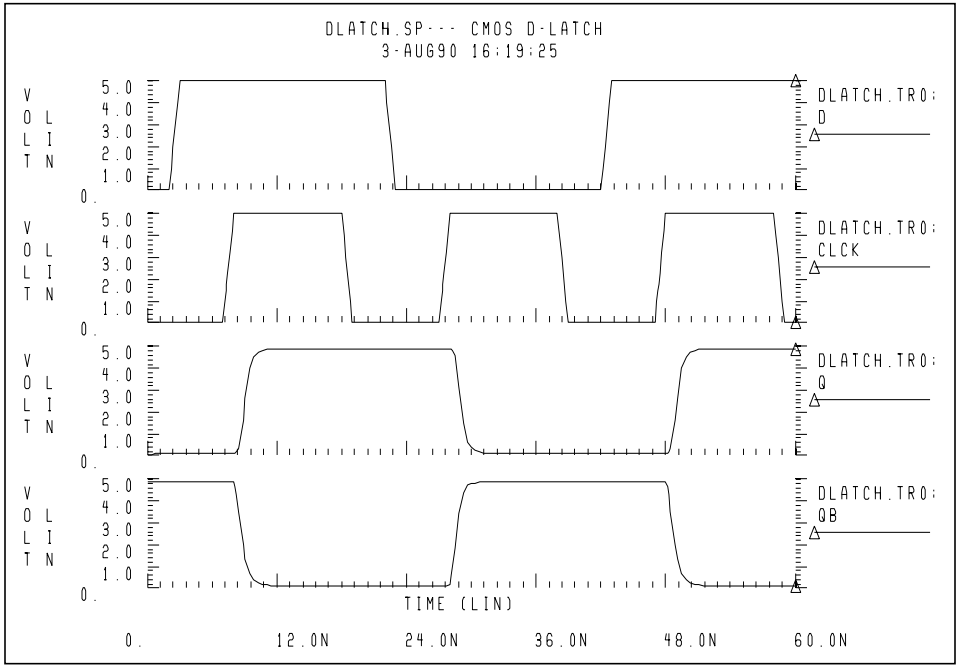
Behavioral P-Channel MOSFET

```
* DRAIN GATE SOURCE
.SUBCKT pmos 1 2 3 capm=.5p
cd 1 0 capm
cs 3 0 capm
gp 1 3 VCR PPWL(1) 2 3
+ -5.0000 2.3845K
+ -4.8000 2.4733K
+ -4.6000 2.5719K
+ -4.4000 2.6813K
```

```
+ -4.2000 2.8035K
+ -4.0000 2.9415K
+ -3.8000 3.1116K
+ -3.6000 3.3221K
+ -3.4000 3.5895K
+ -3.2000 3.9410K
+ -3.0000 4.4288K
+ -2.8000 5.1745K
+ -2.6000 6.6041K
+ -2.4000 29.6203K
+ -2.2000 42.4517K
+ -2.0000 58.3239K
+ -1.8000 83.4296K
+ -1.6000 128.1517K
+ -1.4000 221.2640K
+ -1.2000 471.8433K
+ -1.0000 1.6359X
+ -800.00M 41.7023X
+ -600.00M 1.3394G
+ -400.00M 38.3449G
+ -200.00M 267.7325G
+ 0. 328.7122G
.ENDS pmos
*
.subckt tgate in out clk clkn ctg=.5p
xmn in clk out nmos capm=ctg
xmp in clkn out pmos capm=ctg
.ends tgate

.SUBCKT inv in out capout=1p
cout out 0 capout
rout out 0 1.0k
gn 0 out nand(1) in 0 scale=1
+ 0. 4.90ma
+ 0.25 4.88ma
+ 0.5 4.85ma
+ 1.0 4.75ma
+ 1.5 4.42ma
+ 3.5 1.00ma
+ 4.000 0.50ma
+ 4.5 0.2ma
+ 5.0 0.1ma
.ENDS inv
```

```
.subckt dlatch data clck clckn q qb cinv=1p
xtg1 data a clck clckn tgate ctg='cinv/2'
xtg2 q ax clckn clck tgate ctg='cinv/2'
rx ax a 5
xinv1 a qb inv capout=cinv
xinv2 qb q inv capout=cinv
.ends dlatch

.end
```

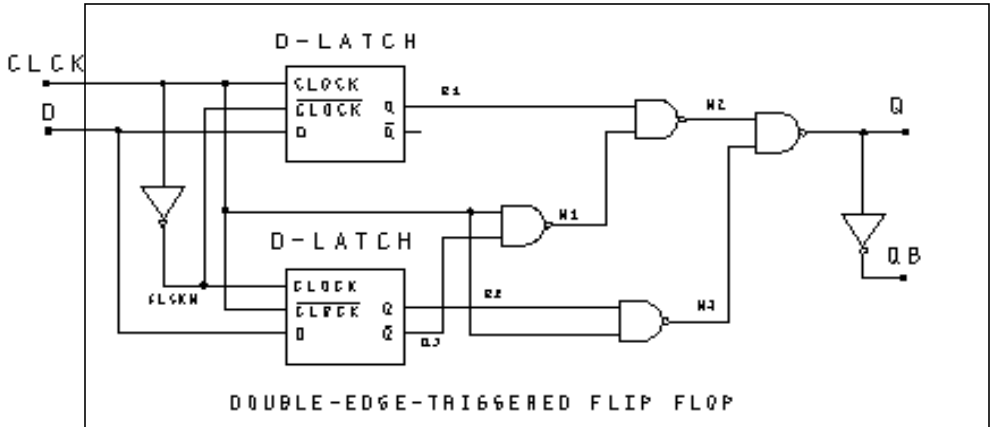
Figure 17-4: D-Latch Response



Behavioral Double-Edge Triggered Flip-Flop

In this example a double edged triggered flip-flop is modeled by using the D_LATCH subcircuit from previous example and several NAND gates.

Figure 17-5: Double-Edge Triggered Flip-Flop Schematic



Example

```
det_dff.sp--- double edge triggered flip-flop
.option post=2
.tran .2n 100ns
.probe tran clock=v(clck) data=v(d) q=v(q)
```

Waveforms

```
vdata d 0 pulse(0,5 2n,1n,1n 28n,50n)
vclk clck 0 pulse(0,5 7n,1n,1n 10n,20n)
```

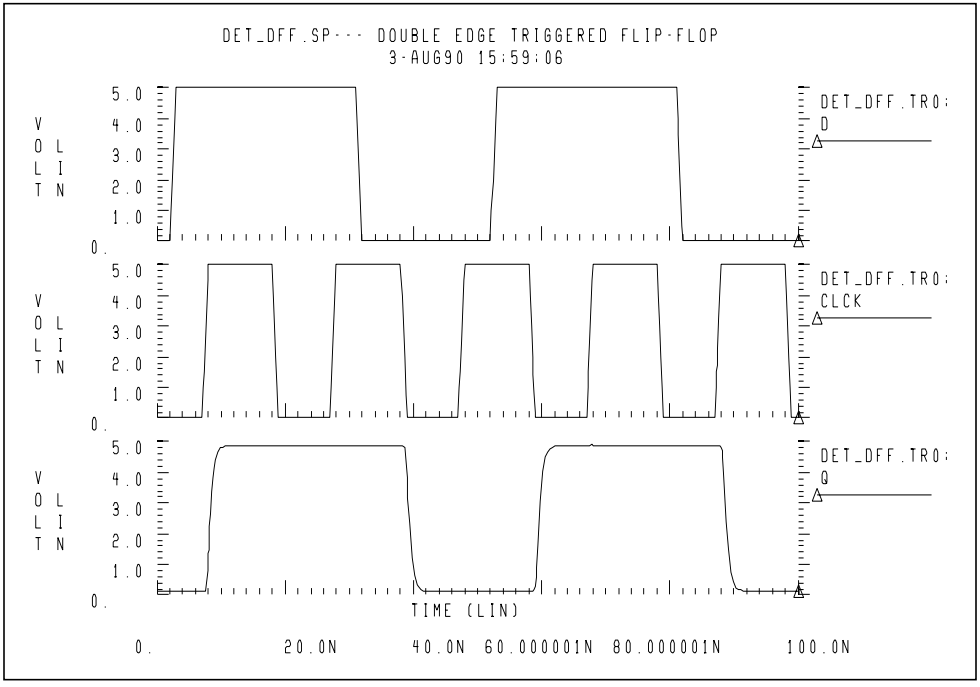
Main Circuit

```
xclkn clck clckn inv cinv=.1p
xd1 d clck clckn q1 qb1 dlatch cinv=.2p
xd2 d clckn clck q2 qb2 dlatch cinv=.2p
xnand1 clck qb2 n1 nand2 capout=.5p
xnand2 q1 n1 n2 nand2 capout=.5p
xnand3 q2 clck n3 nand2 capout=.5p
xnand4 n2 n3 q nand2 capout=.5p
xinv q qb inv capout=.5p
```

Subcircuit Definitions

```
* Note: Subcircuit definitions for NMOS, PMOS, and INV
* are given in the D-Latch examples; therefore they are
*not repeated here.
*

.SUBCKT nand2 in1 in2 out capout=2p
cout out 0 capout
rout out 0 1.0k
gn 0 out nand(2) in1 0 in2 0 scale=1
+ 0. 4.90ma
+ 0.25 4.88ma
+ 0.5 4.85ma
+ 1.0 4.75ma
+ 1.5 4.42ma
+ 3.5 1.00ma
+ 4.000 0.50ma
+ 4.5 0.2ma
+ 5.0 0.1ma
.ENDS nand2
*
.subckt dlatch data clck clckn q qb cinv=1p
xtg1 data a clck clckn tgate ctg='cinv/2'
xtg2 q ax clckn clck tgate ctg='cinv/2'
rx ax a 10
xinv1 a qb inv capout=cinv
xinv2 qb q inv capout=cinv
.ends dlatch

.end
```

Figure 17-6: Double Edge Triggered Flip-Flop Response



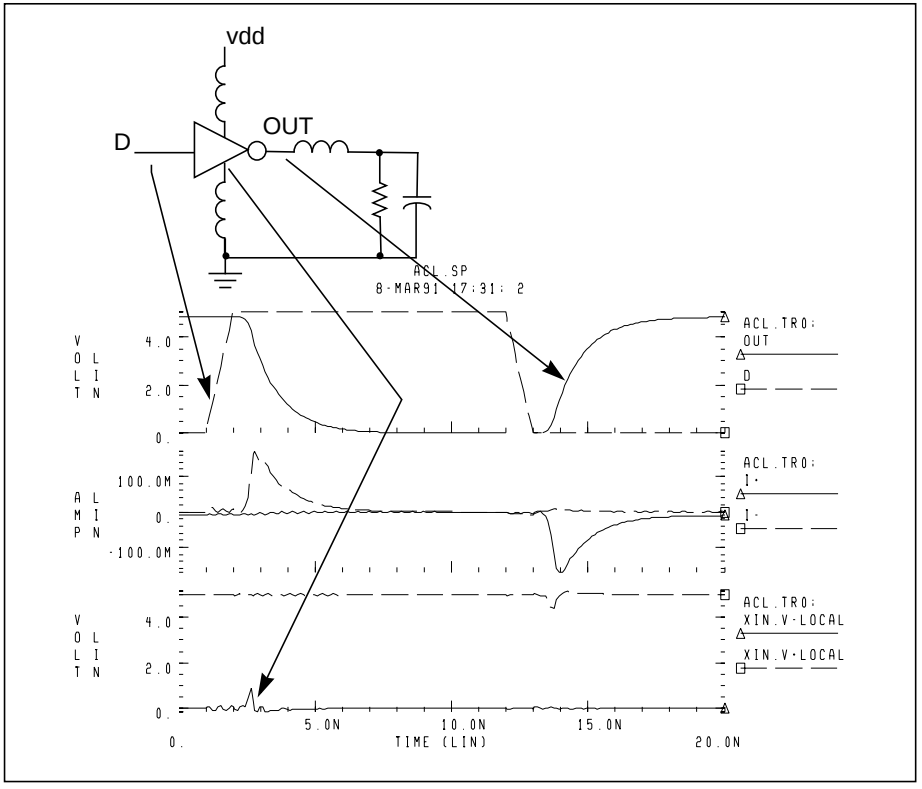
Calibrating Digital Behavioral Components

This section describes how to calibrate with digital behavioral components.

Building Behavioral Lookup Tables

The following simulation demonstrates an ACL family output buffer with 2 ns delay and 1.8 ns rise and fall time. The ground and VDD supply currents and the internal ground bounce due to the package are also shown.

Figure 17-7: ACL Family Output Buffer

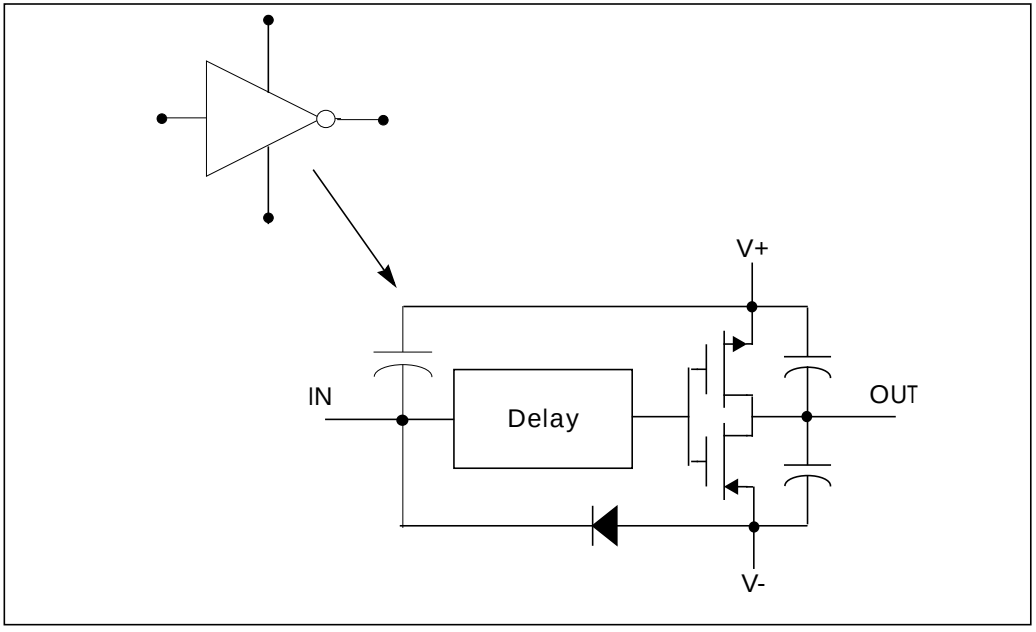


Star-Hspice can automatically measure the datasheet quantities such as TPHL, risetime, maximum power dissipation, and ground bounce using the following commands.

```
.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 is composed of capacitors, diodes, one-dimensional lookup table MOSFETs, and a special low-pass delay element. The low-pass delay element has the property that attenuates pulses that are narrower than the delay value.

Figure 17-8: 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
```

The behavioral MOSFETs are represented by one dimensional lookup tables. The equivalent n-channel lookup table is shown below.

Behavioral N-Channel MOSFET

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 table above describes a voltage versus resistance table. It shows, for example, that the resistance at 5 volts is 2.3 kohm.

Creating a Behavioral Inverter Lookup Table

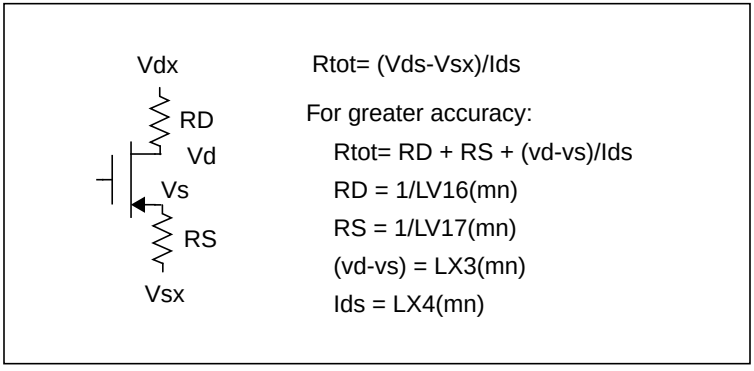
You can create an inverter lookup table in two simple steps. First simulate an actual transistor level inverter using a DC sweep of the input and print the output V/I for the output pullup and pulldown transistors. Next, copy the printed output into the volt controlled resistor lookup table element.

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.

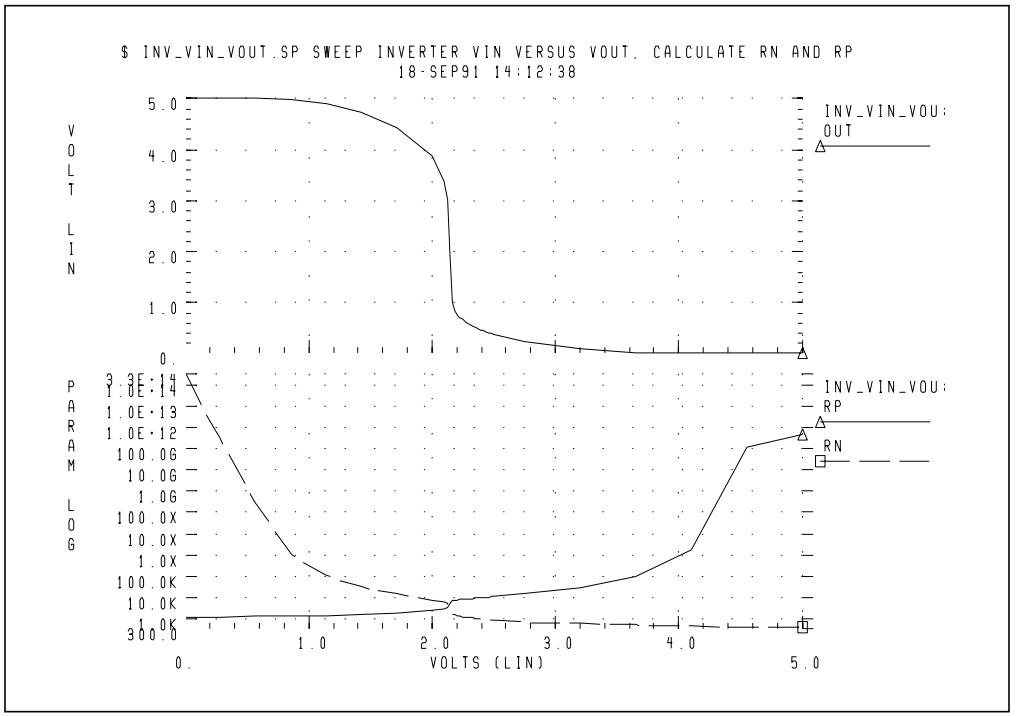
The actual calculation uses a more accurate way of obtaining the transistor series resistance as follows:

Figure 17-9: VIN versus VOUT



The first graph below shows VIN versus VOUT and the second graph shows the computed transfer resistances RP and RN as a function of VIN.

Figure 17-10: RP and RN as a Function of VIN



The Star-Hspice file used to calculate RP and RN is

```
$ inv_vin_vout.sp sweep inverter vin versus vout,
$ calculate rn and rp
```

The triple range DC sweep allows coarse grid before and after:

```
* use dc sweep with 3 ranges; 0-1.5v, 1.6-2.5, 2.6 5
.dc vin lin 8 0 2.0 lin 20 2.1 2.5 lin 6 2.75 5
$$ rn=par('v(out)/i(x1.mn)')
.print rn=
+ par('1/lv16(x1.mn)+1/lv17(x1.mn)+abs(lx3(x1.mn)/
+ lx4(x1.mn))')
.print rp=par('(-vcc+v(out))/i(x1.mp)')
.param sigma=0 vcc=5
.global vcc
vcc vcc 0 vcc
vin in 0 pwl 0,0 0.2n,5
```

```
x1 in out inv
.macro inv in out
mn out in 0 0 nch w=10u l=1u
mp out in vcc vcc pch w=10u l=1u
.eom
```

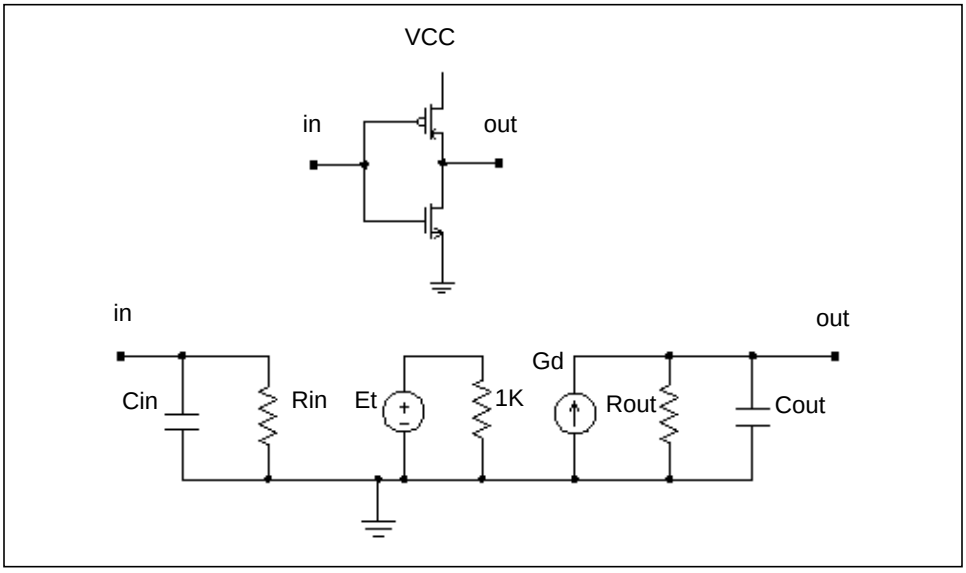
The tabular listing produced by Star-Hspice is

```
****** dc transfer curves tnom= 25.000 temp= 25.000
******

volt            rn
0.              3.312e+14
285.71429m      317.3503g
571.42857m      304.0682x
857.14286m      1.1222x
1.14286         107.6844k
1.42857         32.1373k
1.71429         14.6984k
2.00000         7.7108k
2.10000         5.8210k
2.12105         5.1059k
2.14211         3.2036k
2.16316         1.6906k
2.18421         1.4421k
2.20526         1.3255k
2.22632         1.2470k
2.24737         1.1860k
2.26842         1.1360k
2.28947         1.0935k
2.31053         1.0565k
2.33158         1.0238k
2.35263         994.3804
2.37368         967.7559
2.39474         943.4266
2.41579         921.0413
2.43684         900.3251
2.45789         881.0585
2.47895         863.0632
2.50000         846.1922
2.75000         701.5119
3.20000         560.6908
3.65000         479.8893
4.10000         426.4486
4.55000         387.7524
5.00000         357.4228
```

Optimizing Behavioral CMOS Inverter Performance

Calibrate behavioral models by running Star-Hspice on the full transistor version of a cell and then optimizing the behavioral model to this data.

Figure 17-11: CMOS Inverter and its Equivalent Circuit



In this example, Star-Hspice simulates the CMOS inverter using the LEVEL 3 MOSFET model. The input and output resistances are obtained by performing a .TF transfer function analysis (.TF V(out) Vin). The transfer function table of the inverter is obtained by performing the .DC analysis sweeping input voltage (.DC Vin 0 5 .1). This table is then used in the PWL element to represent the transfer function of the inverter. The rise and fall time of the inverter in the equivalent circuit is adjusted by a voltage controlled PWL capacitance across the output resistance. The propagation delay is obtained by the delay element across the output rc circuit. The input capacitance is adjusted by using the inverter in a ring oscillator. All the adjustment in this example is done using the Star-Hspice optimization analysis. The data file and the results are shown below.

Example

```
INVB_OP.SP---OPTIMIZATION OF CMOS MACROMODEL INVERTER
.OPTIONS POST PROBE NOMOD METHOD=GEAR
.GLOBAL VCC VCCM
.PARAM VCC=5 ROUT=2.5K CAPIN=.5P
+ TDELAY=OPTINV(1.0N,.5N,3N)
+ CAPL=OPTINV(.2P,.1P,.6P)
+ CAPH=OPTINV(.2P,.1P,.6P)
.TRAN .25N 120NS
+ SWEEP OPTIMIZE=OPTINV
RESULTS=RISEX,FALLX,PROPFX,PROPRX
+ MODEL=OPT1
.MODEL OPT1 OPT  ITROPT=30 RELIN=1.0E-5  RELOUT=1E-4
.MEAS TRAN PROPFM  TRIG V(INM)  VAL='.5*VCC' RISE=2
+                  TARG V(OUTM) VAL='.5*VCC' FALL=2
.MEAS TRAN PROPFX  TRIG V(IN)   VAL='.5*VCC' RISE=2
+                  TARG V(OUT)  VAL='.5*VCC' FALL=2
+                  GOAL='PROPFM' WEIGHT=0.8
.MEAS TRAN PROPRM  TRIG V(INM)  VAL='.5*VCC' FALL=2
+                  TARG V(OUTM) VAL='.5*VCC' RISE=2
.MEAS TRAN PROPRX  TRIG V(IN)   VAL='.5*VCC' FALL=2
+                  TARG V(OUT)  VAL='.5*VCC' RISE=2
+                  GOAL='PROPRM' WEIGHT=0.8
.MEAS TRAN FALLM  TRIG V(OUTM) VAL='.9*VCC' FALL=2
+                 TARG V(OUTM) VAL='.1*VCC' FALL=2
.MEAS TRAN FALLX  TRIG V(OUT)  VAL='.9*VCC' FALL=2
+                 TARG V(OUT)  VAL='.1*VCC' FALL=2
+                 GOAL='FALLM'
.MEAS TRAN RISEM  TRIG V(OUTM) VAL='.1*VCC' RISE=2
+                 TARG V(OUTM) VAL='.9*VCC' RISE=2
.MEAS TRAN RISEX  TRIG V(OUT)  VAL='.1*VCC' RISE=2
+                 TARG V(OUT)  VAL='.9*VCC' RISE=2
+                 GOAL='RISEM'
.TRAN 0.5N 120N
.PROBE V(out) V(outm)
VC VCC 0 VCC
VCCM VCCM 0 VCC
X1  IN OUT INV
X1M INM OUTM INVM
VIN   IN   GND PULSE(0,5 1N,5N,5N 20N,50N)
VINM INM GND PULSE(0,5 1N,5N,5N 20N,50N)
```

Subcircuit Definition

```
.SUBCKT INV IN OUT
RIN IN 0 1E12
CIN IN 0 CAPIN
ET   1 0 PWL(1)    IN 0
+    1.00000       5.0
+    1.50000       4.93
+    2.00000       4.72
+    2.40000       4.21
+    2.50000       3.77
+    2.60000       0.90
+    2.70000       0.65
+    3.00000       0.30
+    3.50000       0.092
+    4.00000       0.006
+    4.60000       0.

RT    1 0    1K
GD   0 OUT DELAY 1 0 TD=TDELAY   SCALE='1/ROUT'
GCOUT OUT 0 VCCAP PWL(1) IN 0   1V,CAPL 2V,CAPH
ROUT OUT 0   ROUT
.ENDS
```

Inverter Using Model

```
.SUBCKT INVM IN OUT
        XP1 OUT IN VCCM VCCM MP
        XN1 OUT IN GND  GND   MN
.ENDS

.MODEL N NMOS LEVEL=3 TOX=850E-10 LD=.85U NSUB=2E16
+ VTO=1 GAMMA=1.4 PHI=.9 UO=823 VMAX=2.7E5 XJ=0.9U
+ KAPPA=1.6 ETA=.1 THETA=.18 NFS=1.6E11 RSH=25
+ CJ=1.85E-4 MJ=.42 PB=.7 CJSW=6.2E-10 MJSW=.34
+ CGSO=5.3E-10 CGDO=5.3E-10 CGBO=1.75E-9

.MODEL P PMOS LEVEL=3 TOX=850E-10 LD=.6U
+ NSUB=1.4E16 VTO=-.86 GAMMA=.65 PHI=.76 UO=266
+ VMAX=.8E5 XJ=0.7U KAPPA=4 ETA=.25 THETA=.08
+ NFS=2.3E11 RSH=85 CJ=1.78E-4 MJ=.4 PB=.6 CJSW=5E-10
+ MJSW=.22 CGSO=5.3E-10 CGDO=5.3E-10 CGBO=.98E-9

SUBCKT MP   1     2     3       4
M1 1 2 3 4 P W=45U L=5U AD=615P AS=615P
+ PD=65U PS=65U NRD=.4 NRS=.4
.ENDS MP
```

```
.SUBCKT MN   1     2    3        4
M1 1 2 3 4 N W=17U L=5U AD=440P AS=440P
+ PD=80U PS=80U NRD=.85 NRS=.85
.ENDS MN

.END
```

Result

```
OPTIMIZATION RESULTS
  RESIDUAL SUM OF SQUARES     = 4.589123E-03
  NORM OF THE GRADIENT        = 1.155285E-04
  MARQUARDT SCALING PARAMETER =   130.602
  NO. OF FUNCTION EVALUATIONS =       51
  NO. OF ITERATIONS           =       15
OPTIMIZATION COMPLETED

MEASURED RESULTS < RELOUT= 1.0000E-04 ON LAST
+ ITERATIONS
```

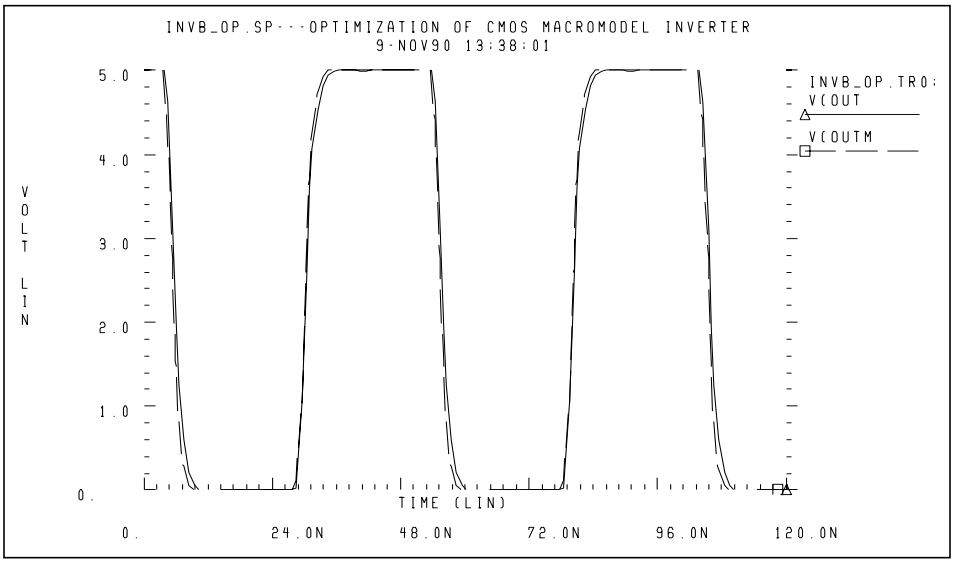
Optimized Parameters OPTINV

```
*                                  %NORM-SEN   %CHANGE
.PARAM TDELAY       =   1.3251N $  37.6164   -48.6429U
.PARAM CAPL         = 390.2613F $  37.2396    60.2596U
.PARAM CAPH         = 364.2716F $  25.1440    62.1922U
```

Optimize Results Measure Names and Values

```
* RISEX           =   2.7018N
* FALLX           =   2.5388N
* PROPFX          =   2.0738N
* PROPRX          =   2.1107N
```

Figure 17-12: CMOS Inverter Response



Optimizing Behavioral Ring Oscillator Performance

To optimize behavioral ring oscillator performance, review the examples in this section.

Example Five-Stage Ring Oscillator

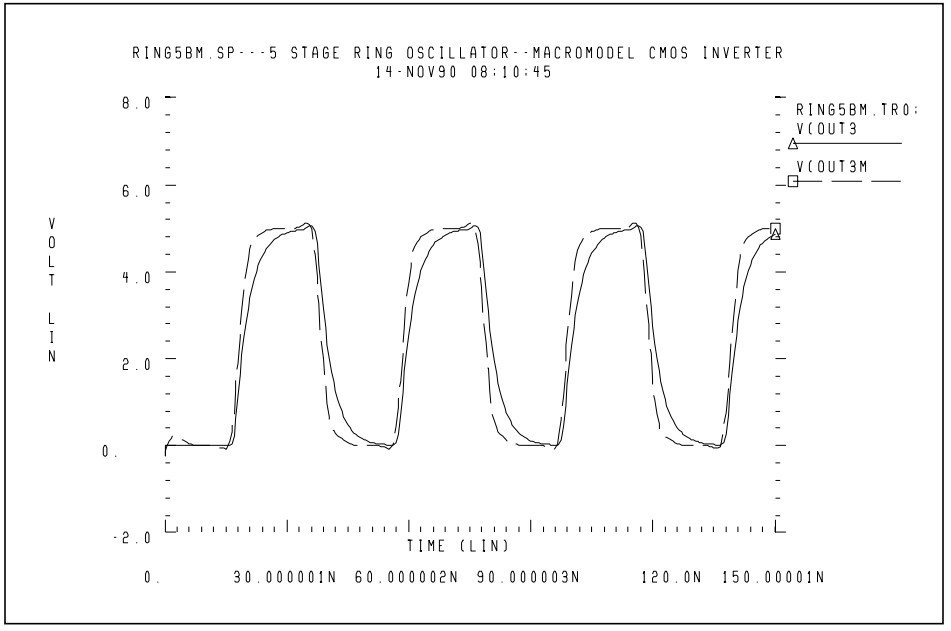
```
RING5BM.SP-5 STAGE RING OSCILLATOR--MACROMODEL CMOS
+ INVERTER

.IC V(IN)=5 V(OUT1)=0 V(OUT2)=5 V(OUT3)=0
.IC V(INM)=5 V(OUT1M)=0 V(OUT2M)=5 V(OUT3M)=0
.GLOBAL VCCM
.OPTIONS NOMOD POST=2 PROBE METHOD=GEAR DELMAX=0.5N
.PARAM VCC=5 $ CAPIN=0.92137P
.PARAM TDELAY=1.32N CAPL=390.26F CAPH=364.27F ROUT=2.5K
+ CAPIN=OPTOSC(0.8P,0.1P,1.0P)
.TRAN 1NS 150NS UIC
+ SWEEP OPTIMIZE=OPTOSC RESULTS=PERIODX MODEL=OPT1
.MODEL OPT1 OPT RELIN=1E-5 RELOUT=1E-4 DIFSIZ=.02
+ ITROPT=25
```

```
.MEAS TRAN PERIODM  TRIG V(OUT3M) VAL='.8*VCC' RISE=2
+                   TARG V(OUT3M) VAL='.8*VCC' RISE=3
.MEAS TRAN PERIODX  TRIG V(OUT3) VAL='.8*VCC' RISE=2
+                   TARG V(OUT3) VAL='.8*VCC' RISE=3
+                   GOAL='PERIODM'

.TRAN 1NS 150NS UIC
.PROBE V(OUT3) V(OUT3M)
X1 IN   OUT1 INV
X2 OUT1 OUT2 INV
X3 OUT2 OUT3 INV
X4 OUT3 OUT4 INV
X5 OUT4 IN    INV
CL IN   0     1P
VCCM VCCM 0 VCC
X1M INM OUT1M INVM
X2M OUT1M OUT2M INVM
X3M OUT2M OUT3M INVM
X4M OUT3M OUT4M INVM
X5M OUT4M INM INVM
CLM INM 0 1P

*Subcircuit definitions given in the previous example
*are not repeated here.

.END
```

Result

```
Optimization Results
  RESIDUAL SUM OF SQUARES     = 4.704516E-10
  NORM OF THE GRADIENT        = 2.887249E-04
  MARQUARDT SCALING PARAMETER =   32.0000
  NO. OF FUNCTION EVALUATIONS =       52
  NO. OF ITERATIONS           =       20
OTIMIZATION COMPLETED
MEASURED RESULTS < RELOUT= 1.0000E-04 ON LAST
+ ITERATIONS

**** OPTIMIZED PARAMETERS OPTOSC
*                                  %NORM-SEN  %CHANGE
.PARAM CAPIN        = 921.4155F $  100.0000    8.5740U

*** OPTIMIZE RESULTS MEASURE NAMES AND VALUES
* PERIODX         = 40.3180N
```

Figure 17-13: Ring Oscillator Response



Using Analog Behavioral Elements

The following components are examples of analog behavioral building blocks. Each demonstrates a basic Star-Hspice feature:

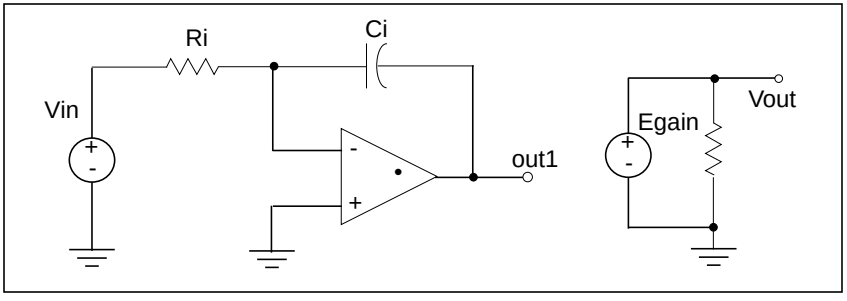
- integrator — ideal op-amp E Element source
- differentiator — ideal op-amp E Element source
- ideal transformer — ideal transformer E Element source
- tunnel diode — lookup table G Element source
- silicon-controlled rectifier — lookup table H Element source
- triode vacuum tube — algebraic G Element source
- AM modulator — algebraic G Element source
- data sampler — algebraic E Element source

Behavioral Integrator

The integrator circuit is modelled by an ideal op-amp and uses a VCVS to adjust the output voltage. The output of integrator is given by:

$$Vout = -\frac{gain}{Ri \cdot Ci} \cdot \int_0^t Vin \cdot dt + Vout(0)$$

Figure 17-14: Integrator Circuit



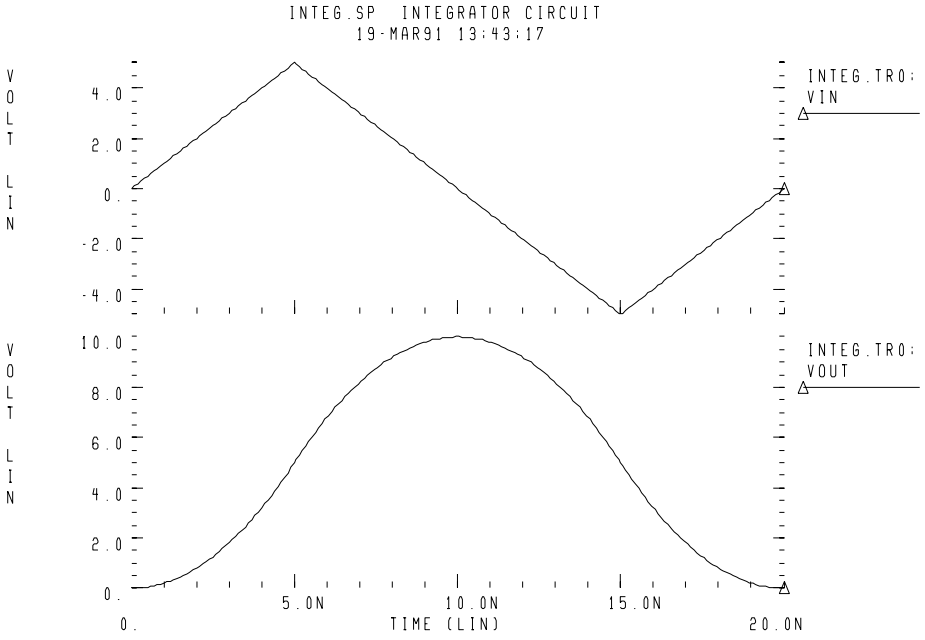
Example

```
Integ.sp integrator circuit
```

Control and Options

```
.TRAN 1n 20n
.OPTIONS POST PROBE DELMAX =.1n
.PROBE Vin=V(in) Vout=V(out)
```

Figure 17-15: Response of Integrator to a Triangle Waveform



Subcircuit Definition

```
.SUBCKT integ in out gain=-1 rval=1k cval=1p
EOP out1 0 OPAMP   in- 0
Ri in in- rval
Ci in- out1 cval
Egain out 0 out1 0 gain
Rout out 0 1e12
.ENDS
```

Circuit

```
Xint in out integ gain=-0.4
Vin in 0 PWL(0,0 5n,5v 15n,-5v 20n,0)
.END
```

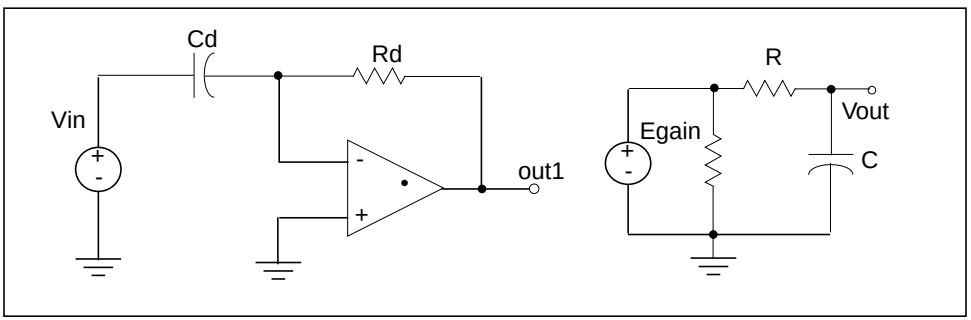
Behavioral Differentiator

A differentiator is modelled by an ideal op-amp and a VCVS for adjusting the magnitude and polarity of the output. The differentiator response is given by:

$$\mathrm{Vout} = -\mathrm{gain} \cdot \mathrm{Rd} \cdot \mathrm{Cd} \cdot \frac{d}{dt}\mathrm{Vin}$$

For a high-frequency signal, the output of a differentiator can have overshoot at the edges. You can smooth this out using a simple RC filter.

Figure 17-16: Differentiator Circuit



Example

```
Diff.sp differentiator circuit
* V(out)=Rval * Cval * gain * (dV(in)/dt)
```

Control and Options

```
.TRAN 1n 20n
.PROBE Vin=V(in) Vout=V(out)
.OPTIONS PROBE POST
```

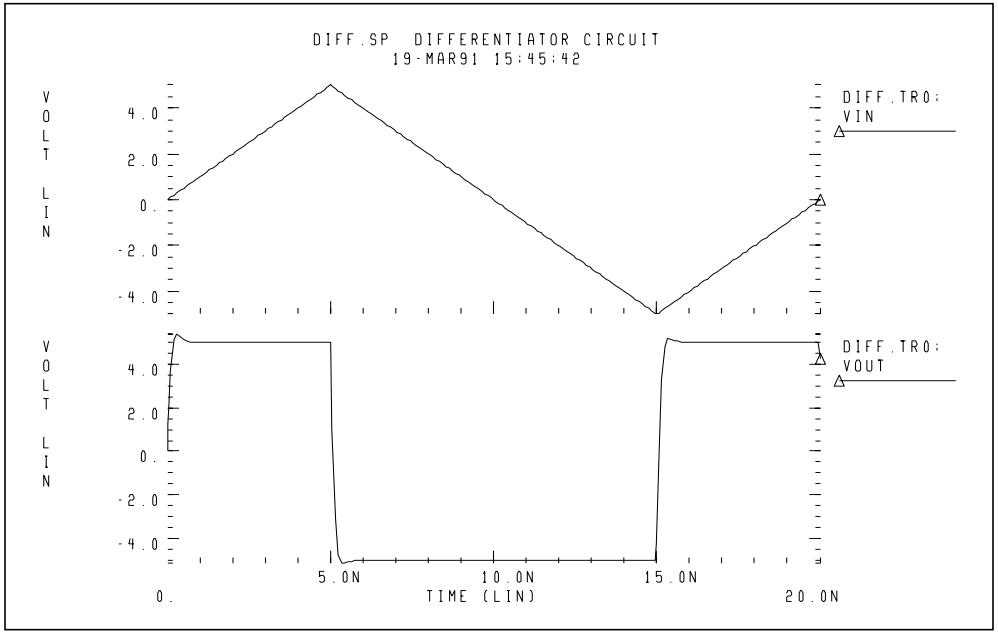
Differentiator Subcircuit Definition

```
.SUBCKT diff in out gain=-1 rval=1k cval=1pf
EOP out1 0 OPAMP   in- 0
Cd in in- cval
Rd in- out1 rval
Egain out2 0 out1 0 gain
Rout out2 0 1e12
*rc filter to smooth the output
R out2 out 75
C out 0   1pf
.ENDS
```

Circuit

```
Xdiff in out diff rval=5k
Vin in 0 PWL(0,0 5n,5v 15n,-5v 20n,0)
.END
```

Figure 17-17: Response Of a Differentiator to a Triangle Waveform

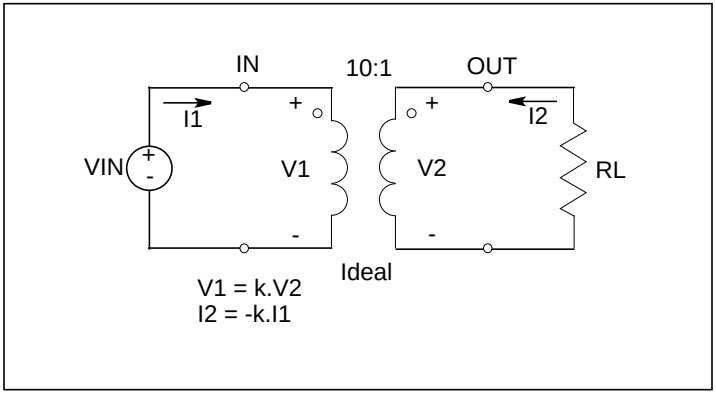


Ideal Transformer

The following example uses the ideal transformer to convert 8 ohms impedance of a loudspeaker to 800 ohms impedance, which is a proper load value for a power amplifier, $Rin=n^2 \cdot RL$.

```
MATCHING IMPEDANCE BY USING IDEAL TRANSFORMER
E OUT 0 TRANSFORMER IN 0 10
RL OUT 0 8
VIN IN 0 1
.OP
.END
```

Figure 17-18: Ideal Transformer Example



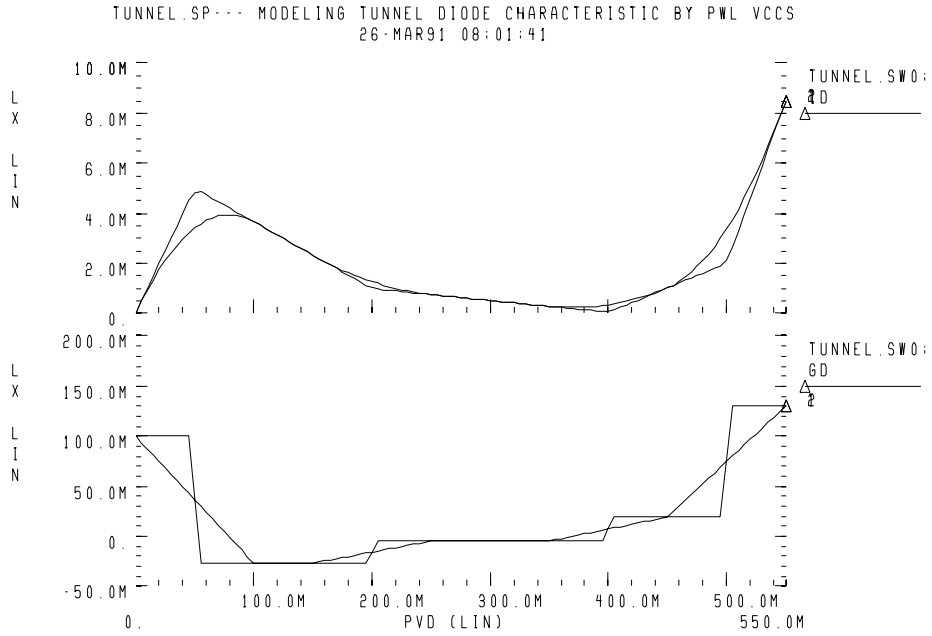
Behavioral Tunnel Diode

In the following example, a tunnel diode is modeled by a PWL VCCS. The current characteristics are obtained for two DELTA values (50 μv and 10 μv). The IV characteristics corresponding to DELTA=10 μv have sharper corners. The derivative of current with respect to voltage (GD) is also displayed. The GD value around breakpoints changes in a linear fashion.

Example

```
tunnel.sp-- modeling tunnel diode characteristic
+ by pwl vccs
* pwl function is tested for two different delta values.
+ The smaller delta will create the sharper corners.

.options post=2
vin 1 0 pvd
.dc pvd 0 550m 5m sweep delta poi 2 50mv 5mv
.probe dc id=lx0(g) gd=lx2(g)
g 1 0 pwl(1) 1 0 delta=delta
+ -50mv,-5ma 50mv,5ma 200mv,1ma 400mv,.05ma
+ 500mv,2ma 600mv,15ma
.end
```

Figure 17-19: Tunnel Diode Characteristic



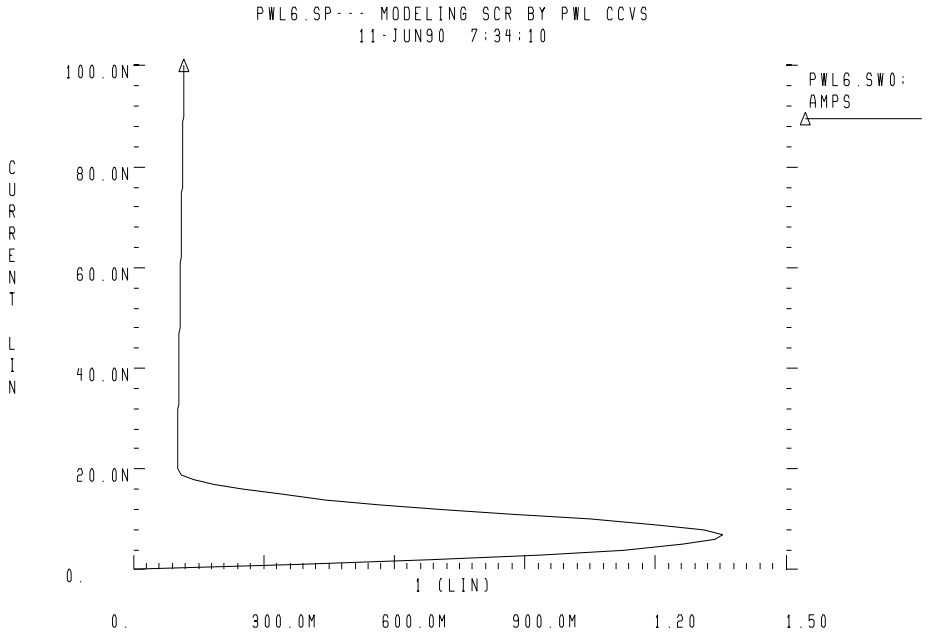
Behavioral Silicon Controlled Rectifier (SCR)

The silicon controlled rectifier (SCR) characteristic can be easily modeled using a PWL CCVS because there is a unique voltage value for any current value.

Example

```
pwl6.sp--- modeling SCR by pwl ccvs
*The silicon controlled rectifier (SCR) characteristic
*is modelled by a piecewise linear current controlled
*voltage source (PWL_CCVS), because for any current
*value there is a unique voltage value.
*
*use iscr as y-axis and v(1) as x-axis
*
.options post=2
iscr 0 2 0
vdum 2 1 0
.dc iscr 0 1u 1n
.probe vscr=lx0(h) transr=lx3(h)
h 1 0 pwl(1) vdum -5na,-2v 5na,2v 15na,.1v 1ua,.3v
+ 10ua,.5 delta=5na
.end
```

Figure 17-20: 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

```
triode.sp triode model family of curves using
+ behavioral elements
```

Control and Options

```
.options post acct
.dc va 20v 60v 1v vg 1v 10v 1v
.probe ianode=i(xt.ra) v(anode) v(grid) eqn=lv6(xt.gt)
.print v(xt.int_anode) v(xt.i_anode) inode=i(xt.ra)
+ eqn=lv6(xt.gt)
```

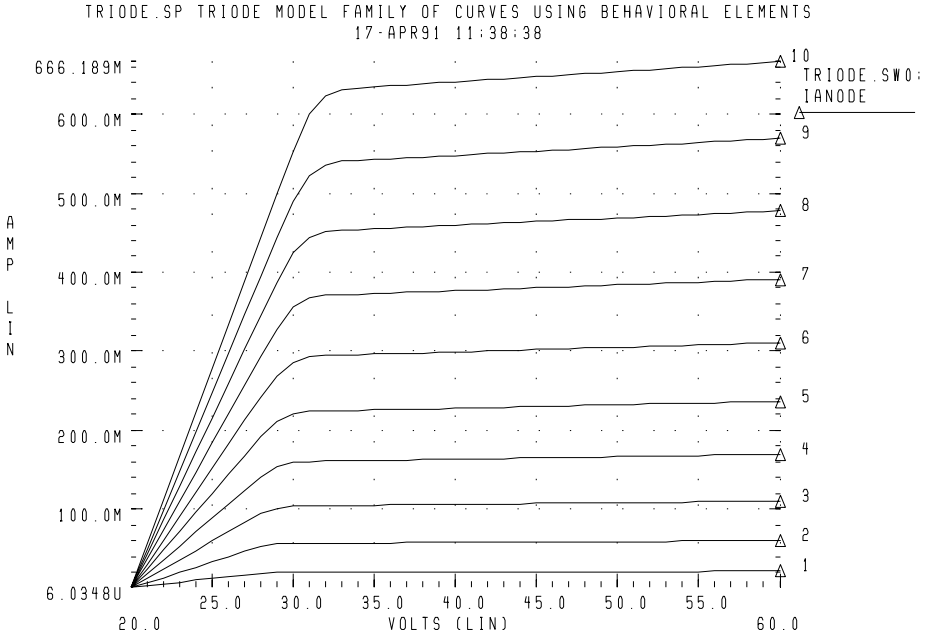
Circuit

```
vg grid 0 1v
va anode 0 20v
vc cathode 0 0v
xt anode grid cathode triode
```

Subcircuit Definition

```
.subckt triode anode grid cathode
* 5 ohm anode resistance
* ea creates saturation region conductance
ra anode i_anode 5
ea int_anode cathode pwl(1) i_anode cathode delta=.01
+ 20,0 27,.85 28,.95 29,.99 30,1 130,1.2
gt i_anode cathode
+ cur='20m*v(int_anode,cathode)*pwr
+ (max(v(grid,cathode),0),1.5)'
cga grid i_anode 30p
cgc grid cathode 20p
cac i_anode cathode 5p
.ends
*
.end
```

Figure 17-21: Triode Family of Curves



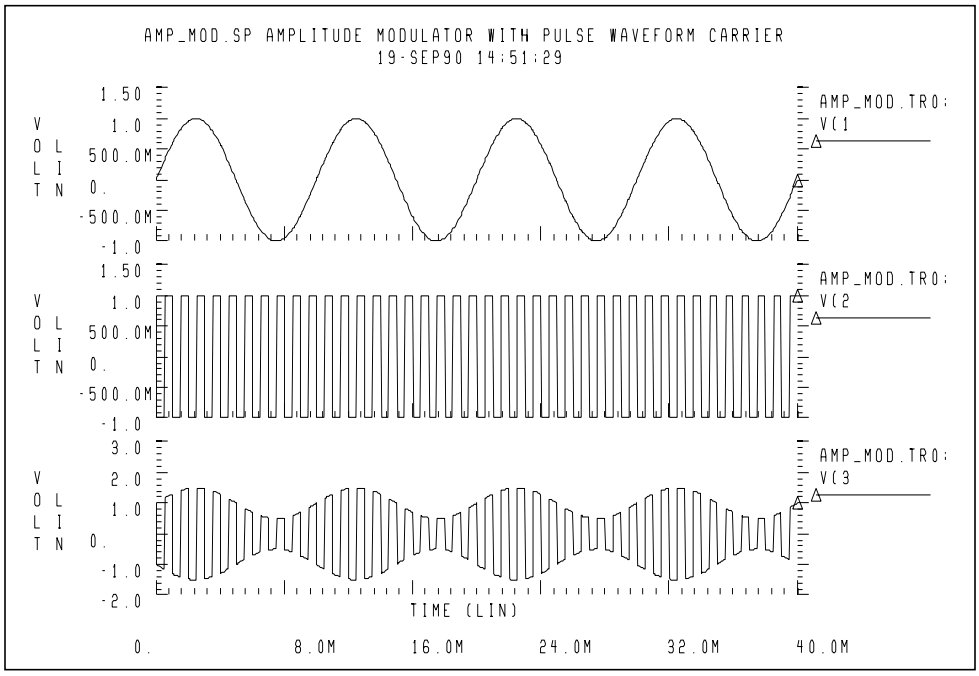
Behavioral Amplitude Modulator

This example uses a G Element as an amplitude modulator with pulse waveform carrier.

Example

```
amp_mod.sp amplitude modulator with pulse waveform
+ carrier
.OPTIONS POST
.TRAN .05m 40m
.PROBE V(1) V(2) V(3)
Vs 1 0 SIN(0,1,100)
Vc 2 0 PULSE(1,-1,0,1n,1n,.5m,1m)
Rs 1 0 1+
Rc 2 0 1
G 0 3 CUR='(1+.5*V(1))*V(2)'
Re 3 0 1
.END
```

Figure 17-22: Amplitude Modulator Waveforms



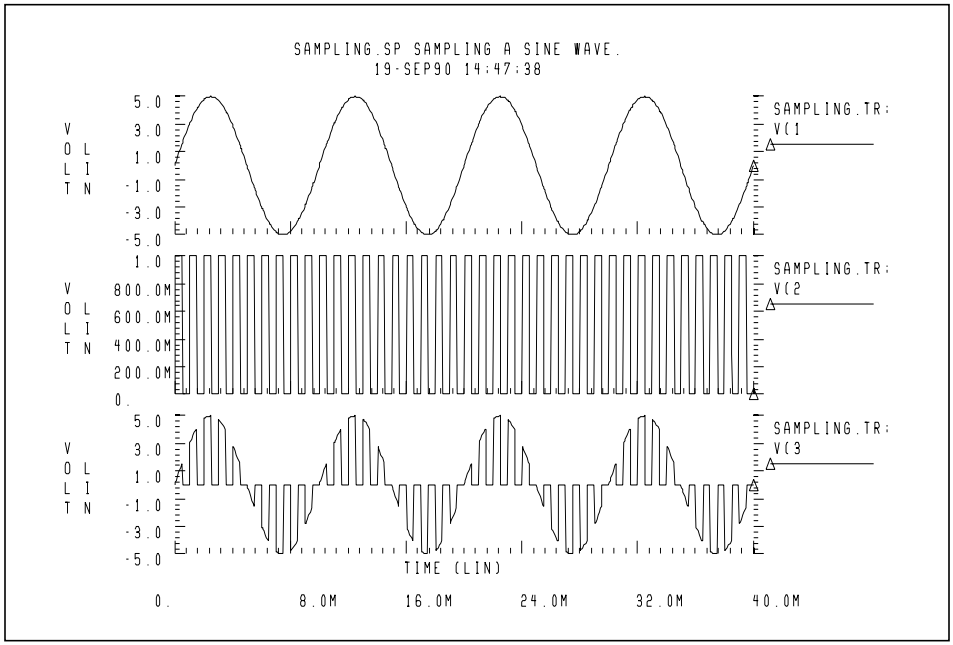
Behavioral Data Sampler

A behavioral data sample follows.

Example

```
sampling.sp sampling a sine wave.
.OPTIONS POST
.TRAN .05m 40m
.PROBE V(1) V(2) V(3)
Vc 1 0 SIN(0,5,100)
Vs 2 0 PULSE(0,1,0,1n,1n,.5m,1m)
Rc 1 0 1
Rs 2 0 1
E 3 0 VOL='V(1)*V(2)'
Re 3 0 1
.END
```

Figure 17-23: Sampled Data



Using Op-Amps, Comparators, and Oscillators

This section describes the benefits of using Star-Hspice's op-amps, comparators, and oscillators when performing simulation.

Star-Hspice Op-Amp Model Generator

Star-Hspice uses the model generator for the automatic design and simulation of both board level and IC op-amp designs. You can take the existing electrical specifications for a standard industrial operational amplifier, enter the specifications in the op-amp model statement, and Star-Hspice automatically generates the internal components of the op-amp to meet the specifications. You can then call the design from a library for a board level simulation.

The Star-Hspice op-amp model is a subcircuit that 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 and may significantly differ from actual amplifiers in predicting high frequency instabilities. Normal amplifier characteristics, including input offsets, small signal gain, and transient effects are represented in this model.

The op-amp subcircuit generator consists of two parts, a model and one or more elements. Each element is in the form of a subcircuit call. The model generates an output file of the op-amp equivalent circuit for collection in libraries. The file name is the name of the model (mname) with an *.inc* extension.

Once the output file is generated, other Star-Hspice input files may reference this subcircuit using a .SUBCKT call to the model name. The .SUBCKT call automatically searches the present directory for the file, then the directories specified in any .OPTION SEARCH ='directory_path_name', and finally the directory where the DDL (Discrete Device Library) is located.

The amplifier element references the amplifier model.

Convergence

If DC convergence problems are encountered with op-amp models created by the model generator, use the .IC or .NODESET statement to set the input nodes to the voltage halfway between the VCC and VEE. This balances the input nodes and stabilizes the model.

Op-Amp Element Statement Format

COMP=0 (internal compensation)

The syntax is:

```
xa1 in- in+ out vcc vee modelname AV=val
```

COMP=1 (external compensation)

The syntax is:

```
xa1 in- in+ out comp1 comp2 vcc vee modelname AV=val
```

in-	the inverting input
in+	the noninverting input
out	the output, single ended
vcc	the positive supply
vee	the negative supply
modelname	the subcircuit reference name

Op-Amp .MODEL Statement Format

The syntax is:

```
.MODEL mname AMP parameter=value …
```

mname	model name. Elements reference the model by this name.
AMP	identifies an amplifier model
parameter	any model parameter described below
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

The model parameters for op-amps are shown below. The defaults for these parameters depend on the DEF parameter setting. Defaults for each of the three DEF settings are shown in the following table.

Table 17-1: Op-Amp Model Parameters *(Sheet 1 of 7)*

Names (Alias)	Units	Default	Description
AV (AVD)	volt/ volt		Amplifier gain in volts out per volt in. It is the DC ratio of the voltage in to the voltage out. Typical gains are from 25k to 250k. If the frequency comes out too low, try increasing the negative and positive slew rates or decreasing DELPHS.
AV1K	volt/ volt		Amplifier gain at 1 kilohertz. This is a convenient method of estimating the unity gain bandwidth. The gain can be expressed in actual voltage gain or in dB. Decibel is now a standard unit conversion for Star-Hspice. If AV1K is set, then FREQ is ignored. A typical value for AV1K is AV1K=(unity gain freq)/1000.

Table 17-1: Op-Amp Model Parameters *(Sheet 2 of 7)*

Names (Alias)	Units	Default	Description
C2	farad		Internal feedback compensation capacitance. If the amplifier is internally compensated and no capacitance value is given, assume 30 pF. If the gain is high (above 500k), the internal compensation capacitor is probably different (typically 10 pF). If the amplifier is externally compensated, (COMP=1) set C2 to about 0.5 pF as the residual internal capacitance.
CMRR	volt/ volt		Common mode rejection ratio. This is usually between 80 and 110 dB. This can be entered as 100 dB or as 100000.
COMP			Compensation level selector. This modifies the number of nodes in the equivalent to include external compensation nodes if set to one. See C2 for external compensation settings. ■ COMP=0 internal compensation (Default) ■ COMP=1 external compensation
DEF			Default model selector. Allows choice of three default settings. ■ 0= generic (0.6 MHz bandwidth) (Default) ■ 1= ua741 (1.2 MHz bandwidth) ■ 2= mc4560 (3 MHz bandwidth)

Table 17-1: Op-Amp Model Parameters *(Sheet 3 of 7)*

Names (Alias)	Units	Default	Description
DELPHS	deg		Excess phase at the unity gain frequency. Also called the phase margin. DELPHS is measured in degrees. Typical excess phases range from 5° to 50°. To determine DELPHS, subtract the phase at unity gain from 90°; this gives 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, particularly values greater than 20°. The model does not have enough poles to always give correct phase and frequency response. It is usually best to pick the DELPHS closest to measured value that does not reduce unity gain bandwidth more than 20%.
DIS	amp	1e-16	Diode and BJT saturation current
FREQ (GBW,*BW*)	Hz		Unity gain frequency. Measured in hertz and typical frequencies range from 100 kHz to 3 MHz. If not specified, measure open loop frequency response at 0 dB voltage gain and the actual compensation capacitance. Typical compensation is 30 pF and single pole compensation configuration. **Note:** If AV1K is greater than zero, Star-Hspice calculates the unity gain frequency from AV1K, and ignores FREQ.

Table 17-1: Op-Amp Model Parameters *(Sheet 4 of 7)*

Names (Alias)	Units	Default	Description
IB	amp		Input bias current. The amount of current required to bias the input differential transistors. This is generally a fundamental electrical characteristic. Typical values are between 20 and 400 nA.
IBOS	amp		Input bias offset current, also called input offset current. This is the amount of unbalanced current between the input differential transistors. Generally 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 between 5 and 25 mA. ISC can also be determined 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 value of ISC, ROUT is automatically reduced.
JIS	amp		JFET saturation current. Default=1e-16 and need not be changed.
LEVIN			Input level type selector. Allows only BJT differential pair creation. LEVIN=1 BJT differential input stage.
LEVOUT			Output level type selector. Allows only single-ended output stage creation. LEVOUT=1 single-ended output stage.

Table 17-1: Op-Amp Model Parameters *(Sheet 5 of 7)*

Names (Alias)	Units	Default	Description
MANU			Manufacturer's name. This can be added to the model parameter list to identify the source of the model parameters. The name is printed in the final equivalent circuit.
PWR (PD)	watt		Total power dissipation value for the amplifier. This includes the calculated value for the op-amp input differential pair. If high slew rate and very low power is specified a warning is issued and the input differential pair alone gives the power dissipation.
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. This can be determined using 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 the effective DC impedance, generally in the frequency region between 1k and 10 kHz. Typical values for ROUT are 50 to 100 ohms.
SRNEG *(SRN)*	volt		Negative going output slew rate. This is found from the graph of the voltage follower pulse response. This is generally a 4 or 5 volt output change with 10 to 20 volt supplies. Measures the negative going change in voltage and the amount of time for the change.

Table 17-1: Op-Amp Model Parameters *(Sheet 6 of 7)*

Names (Alias)	Units	Default	Description
SRPOS (SRP)	volt		Positive going output slew rate. This is found from the graph of the voltage follower pulse response. This is generally a 4 or 5 volt output change with 10 to 20 volt supplies. Measures the positive going change in voltage and the amount of time for the change. Typical slew rates are in the range of 70k to 700k.
TEMP	°C		Temperature in degrees Celsius. This usually is set to the temperature at which the model parameters were measured, which typically is 25 °C.
VCC	volt		Positive power supply reference voltage for VOPOS. The amplifier VOPOS was measured with respect to VCC.
VEE	volt		Negative power supply voltage. The amplifier VONEG was measured 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.

Table 17-1: Op-Amp Model Parameters *(Sheet 7 of 7)*

Names (Alias)	Units	Default	Description
VOS	volt		Input offset voltage. This is the voltage required between the input differential transistors to zero the output voltage. This is generally a fundamental electrical characteristic. Typical values for bipolar amplifiers are in the range 0.1 mV to 10 mV. VOS is measured in volts. VOS can cause a failure to converge for some amplifiers. If this occurs, try setting VOS to 0 or use the initial conditions described above for convergence.

Op-Amp Model Parameter Defaults

Table 17-2: Op-Amp Model Parameter Defaults

Parameter	Description	Defaults		
		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	Unity gain frequency	600 k	-	-
IB	Input bias current	30 n	250 n	40 n
IBOS	Input bias offset current	1.5 n	0.7 n	5 n
ISC	Output short circuit current	25 mA	25 mA	25 mA
LEVIN	Input circuit level selector	1	1	1
LEVOUT	Output circuit level selector	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

Table 17-2: Op-Amp Model Parameter Defaults *(Continued)*

Parameter	Description	Defaults		
		DEF=0	DEF=1	DEF=2
SRPOS	Positive output slew rate	450 k	1 meg	1 meg
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

Op-Amp Subcircuit Example

AUTOSTOP Option

This example uses the .OPTION AUTOSTOP option to shorten simulation time. Once Star-Hspice makes the measurements specified by the .MEASURE statement, the associated transient analysis and AC analysis stops whether or not the full sweep range for each has been covered.

AC Resistance

AC=10000G parameter in the Rfeed element statement installs a 10000 GΩ feedback resistor for the AC analysis in place of the 10 kΩ feedback resistor – used in the DC operating point and transient analysis – which is open-circuited for the AC measurements.

Simulation Results

The simulation results give the DC operating point analysis for an input voltage of 0 v and power supply voltages of 15v. 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 given as 907.885 kHz, which is within 10% of the 1 MHz specified in the .MODEL statement with the parameter FREQ. The required time rate for a 1 volt change in the output (from the .MEASURE statement) is 2.3 μs (from the SRPOS simulation result listing) providing a slew rate of 0.434 Mv/s. This compares to within about 12% of the 0.5 Mv/s given by the SRPOS parameter in 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 10MEGHZ
.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
*
```

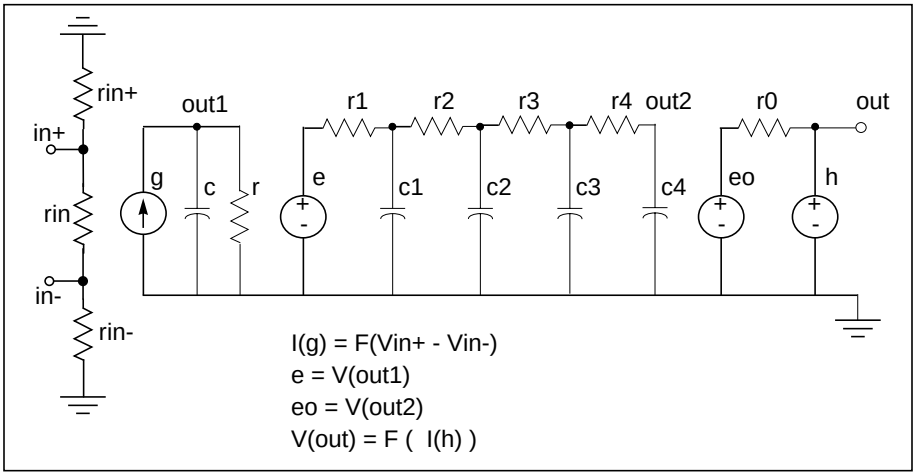
Unity Gain Resistor Divider Mode

```
*
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
```

741 Op-Amp from Controlled Sources

The µA741 op-amp is modeled by PWL controlled sources. The output is limited to ±15 volts by a piecewise linear CCVS (source “h”).

Figure 17-24: Op-Amp Circuit



Example

```
0p_amp.sp --- operational amplifier
*
.options post=2
.tran .001ms 2ms
.ac dec 10 .1hz 10me’
*.graph tran vout=v(output)
*.graph tran vin=v(input)
*.graph ac model=grap voutdb=vdb(output)
*.graph ac model=grap vphase=vp(output)
.probe tran vout=v(output) vin=v(input)
.probe ac voutdb=vdb(output) vphase=vp(output)
.model grap plot xscal=2
```

Main Circuit

```
xamp input 0 output opamp
vin input 0 sin(0,1m,1k) ac 1
* subcircuit definitions
* input subckt
.subckt  opin   in+   in-   out
         rin    in+   in-   2meg
         rin+   in+   0     500meg
         rin-   in-   0     500meg
g 0 out pwl(1) in+ in- -68mv,-68ma 68mv,68ma delta=1mv
         c      out   0     .136uf
         r      out   0     835k
.ends
```

RC Circuit With Pole At 9 MHz

```
.subckt oprc in out
e out1 0 in 0 1
r1 out1 out2 168
r2 out2 out3 1.68k
r3 out3 out4 16.8k
r4 out4 out 168k
c1 out2 0 100p
c2 out3 0 10p
c3 out4 0 1p
c4 out 0 .1p
r out 0 1e12
.ends
```

Output Limiter to 15 v

```
.subckt opout in out
eo out1 0 in 0 1
ro out1 out 75
vdum out dum 0
h dum 0 pwl(1) vdum delta=.01ma -.1ma,-15v .1ma,15v
.ends
* op-amp subckt
.subckt opamp in+ in- out
xin in+ in- out1 opin
xrc out1 out2 oprc
xout out2 out opout
.ends
.end
```

Figure 17-25: AC Analysis Response

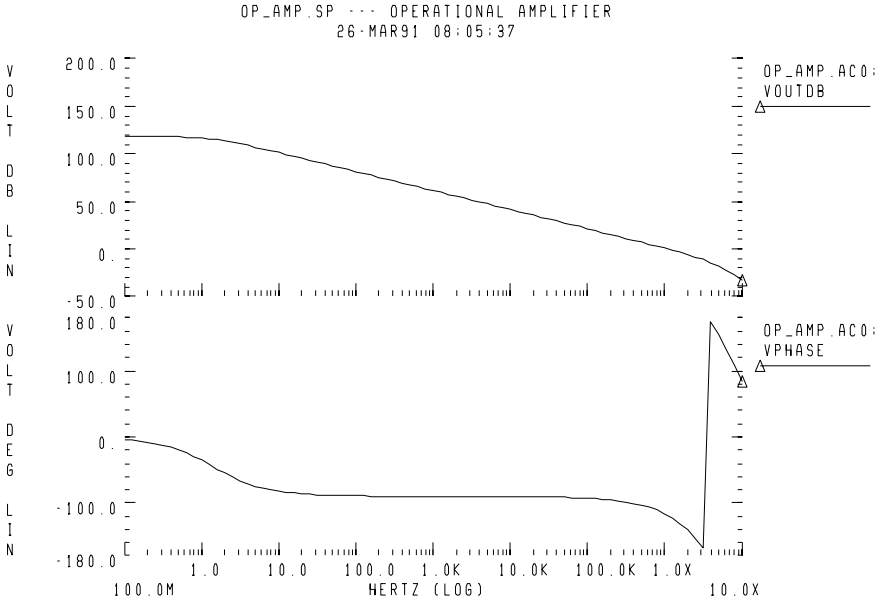
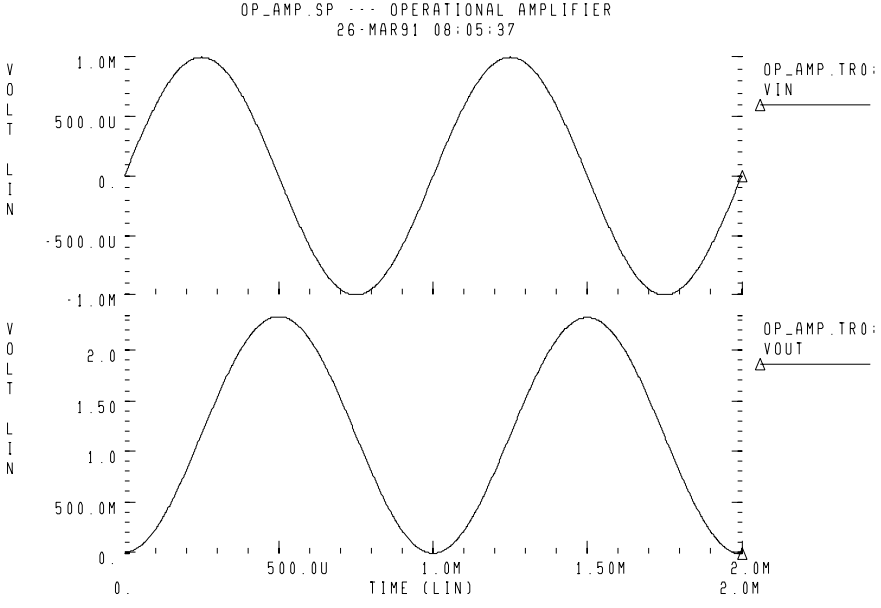


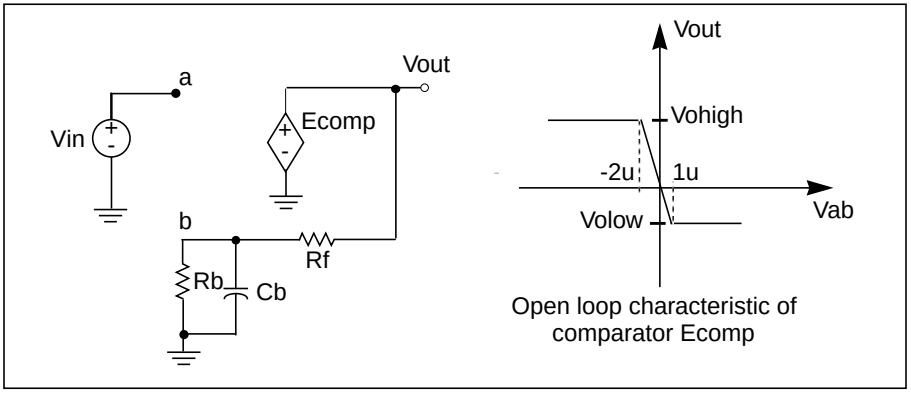
Figure 17-26: Transient Analysis Response[Figure 1]



Inverting Comparator with Hysteresis

An inverting comparator is modelled by a piecewise linear VCVS.

Figure 17-27: Inverting Comparator with Hysteresis



Two reference voltages correspond to volow and vohigh of Ecomp:

$$Vreflow = \frac{Volow \cdot Rb}{Rb + Rf}$$

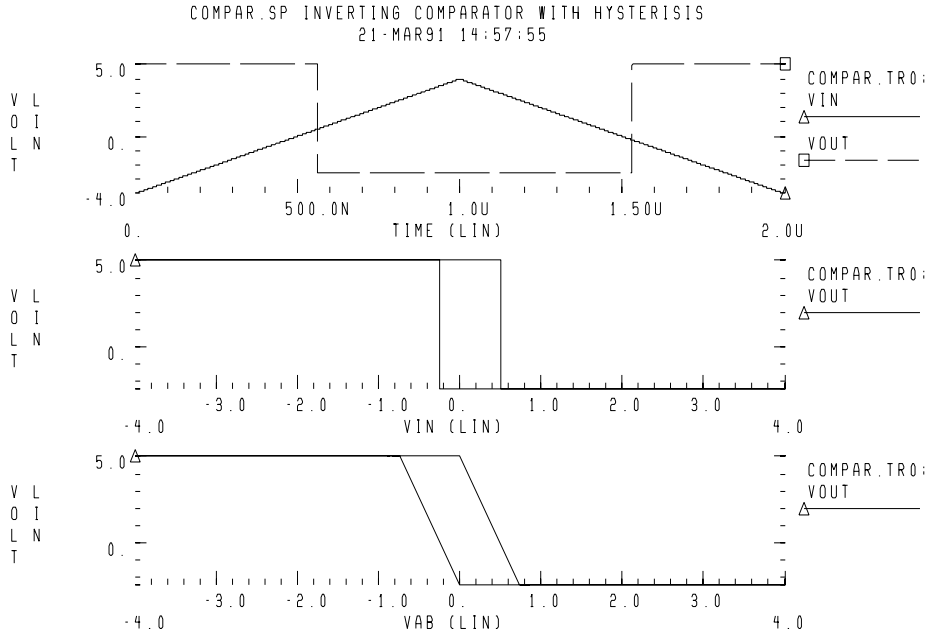
$$Vrefhigh = \frac{Vohigh \cdot Rb}{Rb + Rf}$$

When Vin exceeds Vrefhigh, the output Vout goes to Volow. For Vin less than Vreflow, the output goes to Vohigh.

Example

```
Compar.sp Inverting comparator with hysteresis
.OPTIONS POST PROBE
.PARAM vohigh=5v volow=-2.5v rbval=1k rfval=9k
Ecomp out 0 PWL(1) a b   -2u,vohigh 1u,volow
Rb b 0   rbval
Rf b out rfval
Cb b 0 1ff
Vin a 0 PWL(0,-4 1u,4 2u,-4)
.TRAN .1n 2u
.PROBE Vin=V(a) Vab=V(a,b) Vout=V(out)
.END
```

Figure 17-28: 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 is used to switch 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

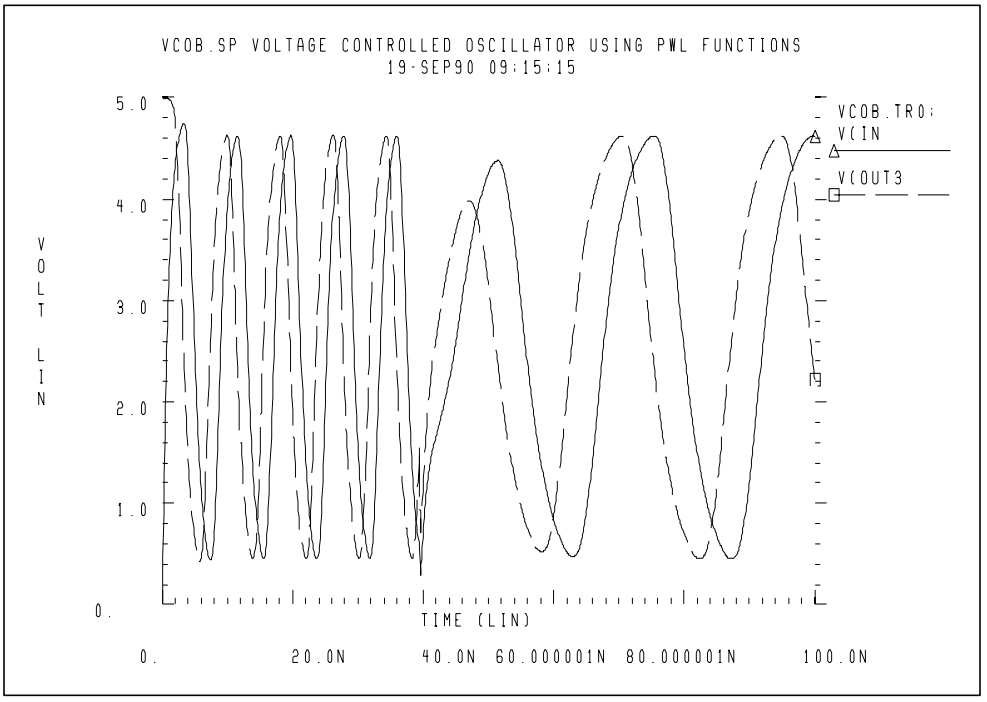
```
vcob.sp voltage controlled oscillator using
+ pwl functions

.OPTION POST
.GLOBAL ctrl
.TRAN 1n 100n
.IC V(in)=0 V(out1)=5
```

```
.PROBE TRAN V(in) V(out1) V(out2) V(out3) V(out4)
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)
```

Subcircuit Definition

```
.SUBCKT inv in out rout=1k

* The following G Element is functioning as PWL
* capacitance.

Gcout out 0 VCCAP PWL(1) ctrl 0 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 17-29: Voltage Controlled Oscillator Response



LC Oscillator

The capacitor is initially charged to 5 volts. The value of capacitance is the function of voltage at node 10. The value of capacitance becomes four times higher at time t2. The frequency of this LC circuit is given by:

$$\text{freq} = \frac{1}{6.28 \cdot \sqrt{L \cdot C}}$$

At time t2, 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:

$$E = (0.5 \cdot C \cdot V^2) + (0.5 \cdot L \cdot I^2)$$

$$E = 0.5 \cdot C \cdot Vm^2, I = 0$$

$$E = 0.5 \cdot L \cdot Im^2, V = 0$$

Assuming at time t2, when V=0, C changes to A · 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 to consider is when V=Vin, C changes to A· C. In this case:

$$Qm = Qm'$$

$$C \cdot Vm = A \cdot C \cdot Vm'$$

$$Vm' = \frac{Vm}{A}$$

Therefore, the voltage amplitude is modified between Vm/sqrt(A) and Vm/A depending on the circuit condition at the switching time. 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

```
calg2.sp voltage variable capacitance
*
.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
*
.TRAN .1u 60u UIC SWEEP DATA=par

.MEAS TRAN period1 TRIG V(1) VAL=0 RISE=1
+ TARG V(1) VAL=0 RISE=2

.MEAS TRAN period2 TRIG V(1) VAL=0 RISE=5
+ TARG V(1) VAL=0 RISE=6

.PROBE TRAN V(1) q1=LX0(C1)
*
.PROBE TRAN V(2) q2=LX0(C2)
.DATA par t1 t2
15.65us 15.80us
17.30us 17.45us

.ENDDATA

.END
```

Figure 17-30: Correct Result Corresponding to CTYPE=1

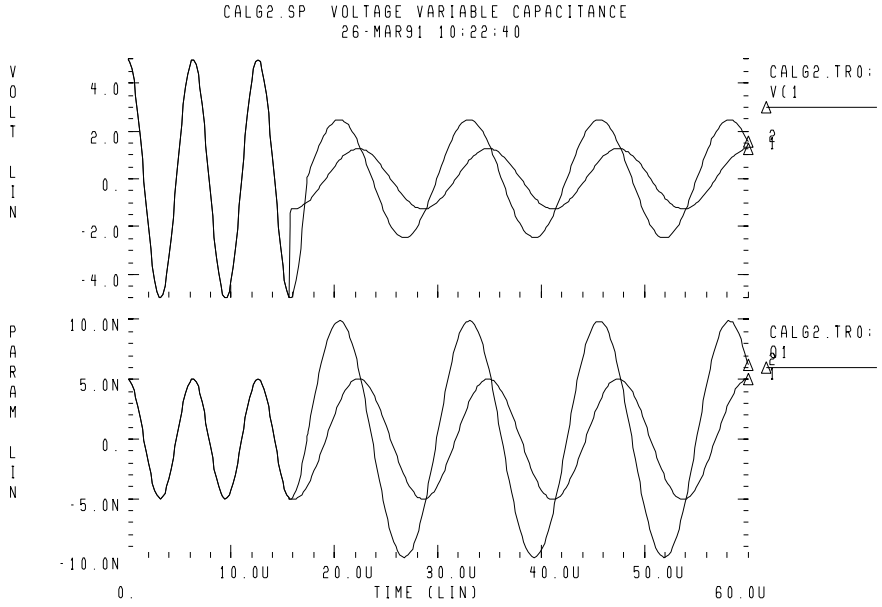
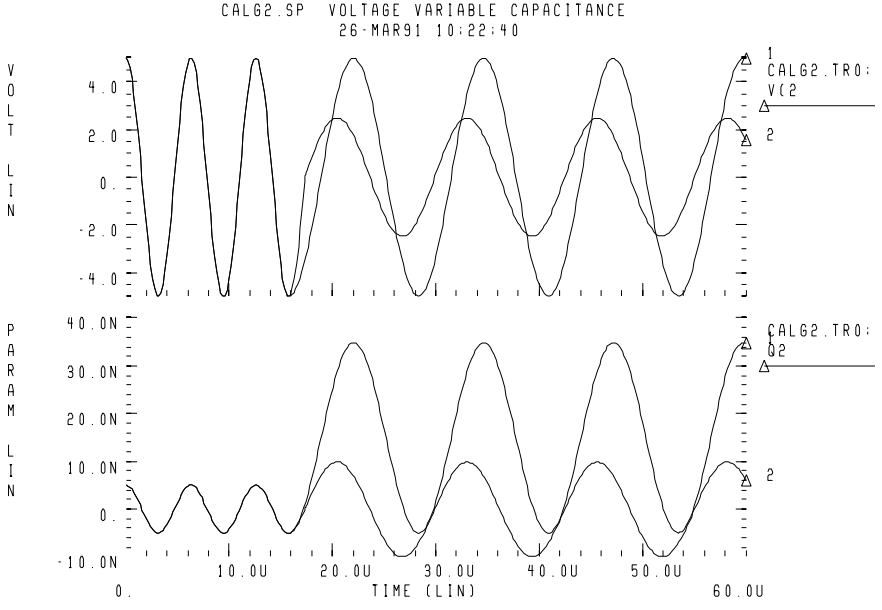


Figure 17-31: Incorrect Result Corresponding to CTYPE=0

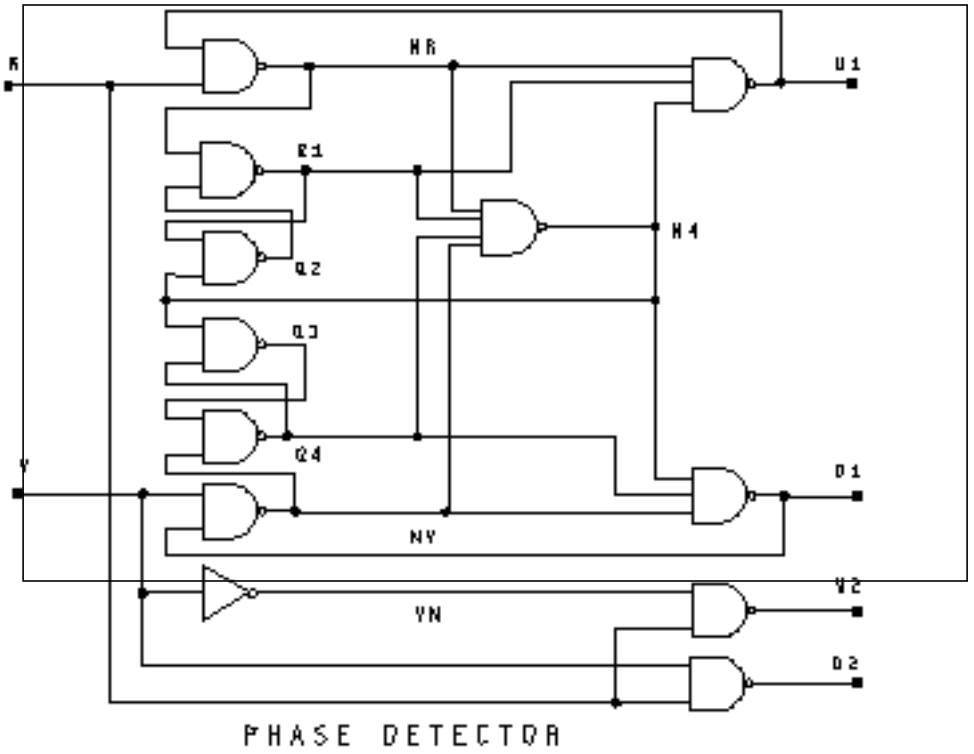


Using a Phase Locked Loop Design

Phase Detector Using Multi-Input NAND Gates

This circuit uses the behavioral elements to implement inverters, 2, 3, and 4 input NAND gates.

Figure 17-32: Circuit Schematic of Phase Detector



Example

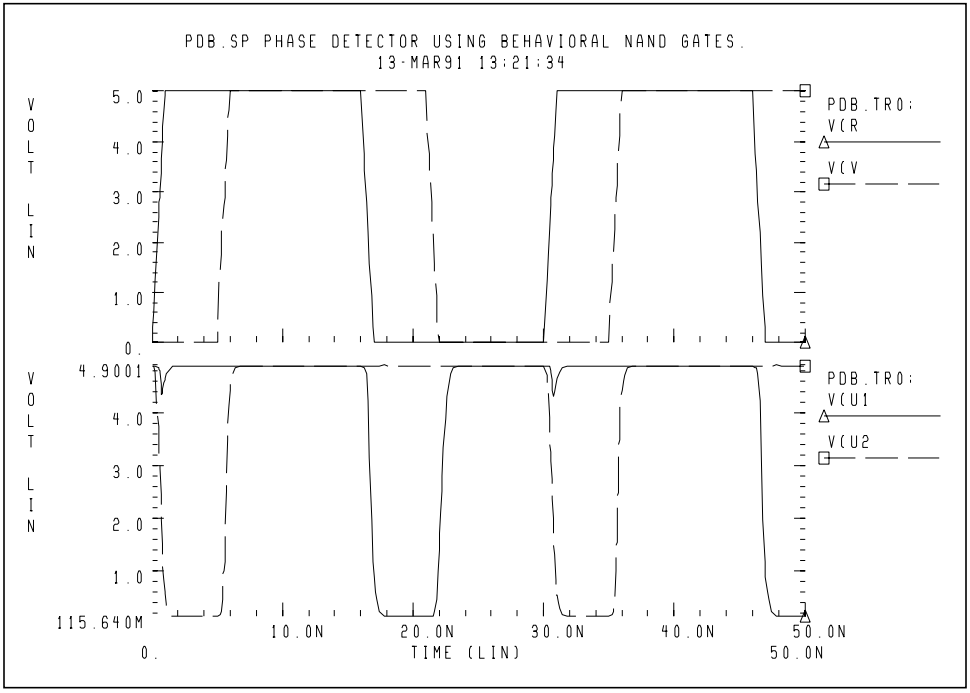
```
pdb.sp phase detector using behavioral nand gates.
.option post=2
.tran .25n 50ns
*.graph tran v(r) v(v) v(u1)
*.graph tran v(r) v(v) v(u2) $ v(d2)
.probe tran v(r) v(v) v(u1)
.probe tran v(r) v(v) v(u2) $ v(d2)
xnr r u1 nr nand2 capout=.1p
xq1 nr q2 q1 nand2 capout=.1p
xq2 q1 n4 q2 nand2
xq3 q4 n4 q3 nand2
xq4 q3 nv q4 nand2
xnv v d1 nv nand2
xu1 nr q1 n4 u1 nand3
xd1 nv q4 n4 d1 nand3
xvn v vn inv
xu2 vn r u2 nand2
xd2 r v d2 nand2
xn4 nr q1 q4 nv n4 nand4
*
* waveform vv lags waveform vr
vr r 0 pulse(0,5,0n,1n,1n,15n,30n)
vv v 0 pulse(0,5,5n,1n,1n,15n,30n)
*
* waveform vr lags waveform vv
*vr r 0 pulse(0,5,5n,1n,1n,15n,30n)
*vv v 0 pulse(0,5,0n,1n,1n,15n,30n)
```

Subcircuit Definitions

```
.SUBCKT inv in out capout=.1p
cout out 0 capout
rout out 0 1.0k
gn 0 out nand(1) in 0 scale=1
+ 0. 4.90ma
+ 0.25 4.88ma
+ 0.5 4.85ma
+ 1.0 4.75ma
+ 1.5 4.42ma
+ 3.5 1.00ma
+ 4.5 0.2ma
+ 5.0 0.1ma
.ENDS inv
```

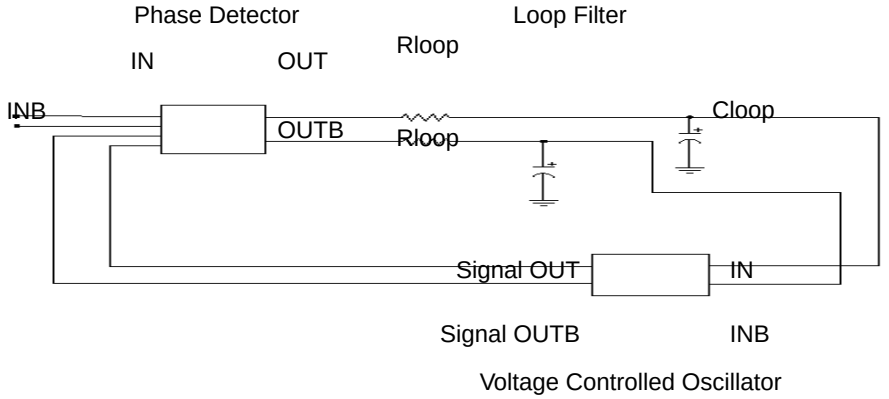
```
.SUBCKT nand2 in1 in2 out capout=.15p
cout out 0 capout
rout out 0 1.0k
gn 0 out nand(2) in1 0 in2 0 scale=1
+ 0. 4.90ma
+ 0.25 4.88ma
+ 0.5 4.85ma
+ 1.0 4.75ma
+ 1.5 4.42ma
+ 3.5 1.00ma
+ 4.000 0.50ma
+ 4.5 0.2ma
+ 5.0 0.1ma
.ENDS nand2

.SUBCKT nand3 in1 in2 in3 out capout=.2p
cout out 0 capout
rout out 0 1.0k
gn 0 out nand(3) in1 0 in2 0 in3 0 scale=1
+ 0. 4.90ma
+ 0.25 4.88ma
+ 0.5 4.85ma
+ 1.0 4.75ma
+ 1.5 4.42ma
+ 3.5 1.00ma
+ 4.000 0.50ma
+ 4.5 0.2ma
+ 5.0 0.1ma
.ENDS nand3

.SUBCKT nand4 in1 in2 in3 in4 out capout=.5p
cout out 0 capout
rout out 0 1.0k
gn 0 out nand(4) in1 0 in2 0 in3 0 in4 0 scale=1
+ 0. 4.90ma
+ 0.25 4.88ma
+ 0.5 4.85ma
+ 1.0 4.75ma
+ 1.5 4.42ma
+ 3.5 1.00ma
+ 4.000 0.50ma
+ 4.5 0.2ma
+ 5.0 0.1ma
.ENDS nand4
.end
```

Figure 17-33: Phase Detector Response



PLL BJT Behavioral Modeling

Figure 17-34: PLL Schematic

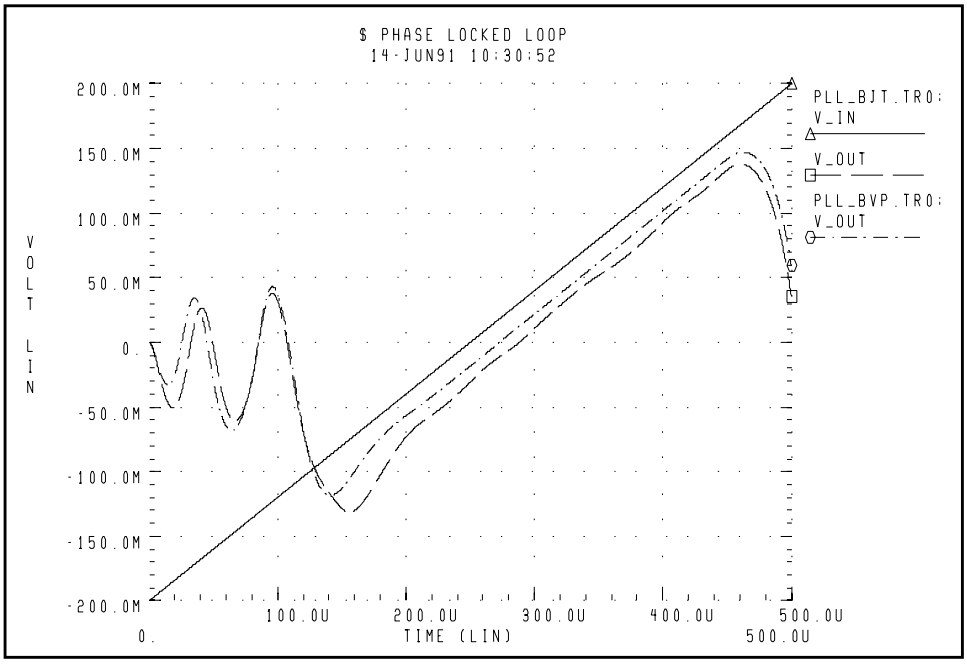


A Phase Locked Loop (PLL) circuit synchronizes to an input waveform within a selected frequency range, returning an output voltage proportional to variations in the input frequency. It has three basic components: a voltage controlled oscillator (VCO), which returns an output waveform proportional to 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, and a loop filter, which filters the phase detector voltage, returning an output voltage which forms the VCO input and the external voltage output of the PLL.

The following example shows a Star-Hspice simulation of a full bipolar implementation of a PLL; its transfer function shows a linear region of voltage vs. (periodic) time which is defined as the “lock” range. The phase detector is modeled behaviorally, effectively implementing a logical XNOR function. This model was then substituted into the full PLL circuit and resimulated. The behavioral model for the VCO was then substituted into the PLL circuit, and this behavioral PLL was then simulated. The results of the transient simulations (Figure 17-35) show minimal difference between implementations, but from the standpoint of run time statistics, the behavioral model shows a factor of five reduction in simulation time versus that of the full circuit.

Include the behavioral model if you use this PLL in a larger system simulation (for example, an AM tracking system) because it substantially reduces run time while still representing the subcircuit accurately.

Figure 17-35: Behavioral (PLL_BVP Curve) vs. Bipolar (PLL_BJT Curve) Circuit Simulation



Example

This is an example of a phase locked loop:

```
$ phase locked loop
.option post probe acct
.option relv=1e-5
$
$ wideband FM example, Grebene gives:
$ f0=1meg kf=250kHz/V
$ kd=0.1 V/rad
$ R=10K C=1000p
$ f_lock = kf*kd*pi/2 = 39kHz, v_lock = kd*pi/2 = 0.157
$ f_capture/f_lock ~= 1/sqrt(2*pi*R*C*f_lock)
$ = 0.63, v_capture ~= 0.100
```

```
*.ic v(out)=0 v(fin)=0
.tran .2u 500u
.option delmax=0.01u interp
.probe v_in=v(inc,0) v_out=v(out,outb)
.probe v(in) v(osc) v(mout) v(out)
```

Input

```
vin inc 0 pwl 0u,-0.2 500u,0.2
*vin inc 0 0

xin inc 0 in inb vco f0=1meg kf=125k phi=0 out_off=0
+ out_amp=0.3

$ vco
xvco e eb osc oscb vco f0=1meg kf=125k phi=0
+ out_off=-1 out_amp=0.3

$ phase detector
xpd in inb osc oscb mout moutb pd kd=0.1 out_off=-2.5

$ filter
rf mout e 10k
cf e 0 1000p
rfb moutb eb 10k
cfb eb 0 1000p

$ final output
rout out e 100k
cout out 0 100p
routb outb eb 100k
coutb outb 0 100p

.macro vco in inb out outb f0=100k kf=50k phi=0.0
+ out_off=0.0 out_amp=1.0

gs 0 s poly(2) c 0 in inb 0 '6.2832e-9*f0'
+ 0 0 '6.2832e-9*kf'
gc c 0 poly(2) s 0 in inb 0 '6.2832e-9*f0'
+ 0 0 '6.2832e-9*kf'
cs s 0 1e-9
cc c 0 1e-12
e1 s_clip 0 pwl(1) s 0 -0.1,-0.1 0.1,0.1
eout 0 s_clip 0 out_off vol='10*out_amp'
eboutb 0 s_clip 0 out_off vol='-10*out_amp'
.ic v(s)='sin(phi)' v(c)='cos(phi)'
.eom
```

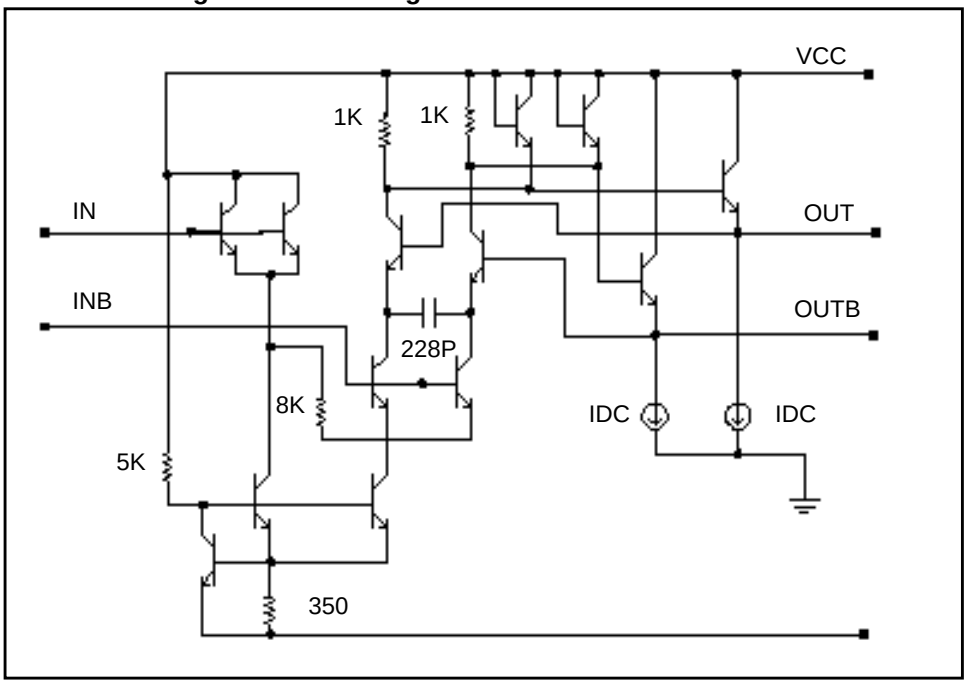
```
.macro pd in inb in2 in2b out outb kd=0.1 out_off=0
e1 clip1 0 pwl(1) in inb -0.1,-0.1 0.1,0.1
e2 clip2 0 pwl(1) in2 in2b -0.1,-0.1 0.1,0.1
e3 n1 0 poly(2) clip1 0 clip2 0 0 0 0 0 '78.6*kd'
e4 outb 0 n1 0 out_off 1
e5 out 0 n1 0 out_off -1
.eom

.end
```

VCO Example

This is an example of a BJT LEVEL Voltage Controlled Oscillator (VCO):

```
$ phase locked loop
.option post probe acct
.option relv=1e-5
$
$ wideband FM example, Grebene gives:
$ f0=1meg kf=250kHz/V
$ kd=0.1 V/rad
$ R=10K C=1000p
$ f_lock = kf*kd*pi/2 = 39kHz, v_lock = kd*pi/2 = 0.157
$ f_capture/f_lock ~= 1/sqrt(2*pi*R*C*f_lock)
$ = 0.63, v_capture ~= 0.100

*.ic v(out)=0 v(fin)=0
.tran .2u 500u
.option delmax=0.01u interp
.probe v_in=v(inc,0) v_out=v(out,outb)
.probe v(in) v(osc) v(mout) v(out) v(e)

vcc vcc 0 6
vee vee 0 -6

$ input
vin inc 0 pwl 0u,-0.2 500u,0.2
xin inc 0 in inb vco f0=1meg kf=125k phi=0 out_off=0
+ out_amp=0.3

$ vco
xvco1 e eb osc oscb 0 vee vco1
.ic v(osc)=-1.4 v(oscb)=-0.7
```

Figure 17-36: Voltage Controlled Oscillator Circuit



BJT Level Phase Detector

Example

```
$ phase detector
xpd1 in inb osc oscb mout moutb vcc vee pd1
```

Filter

```
rf mout e 10k
cf e 0 1000p
rfb moutb eb 10k
cfb eb 0 1000p
```

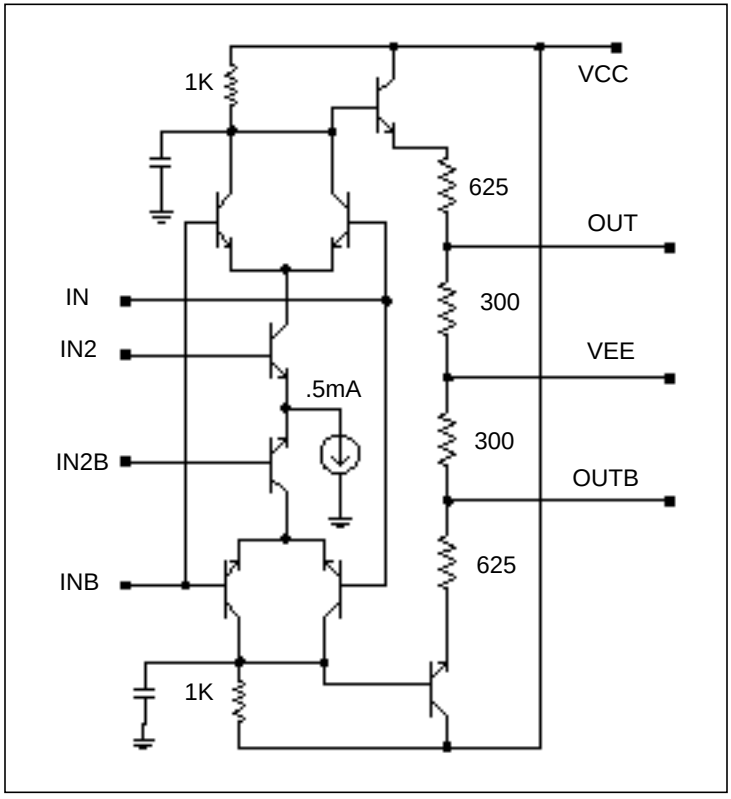
Final Output

```
rout out e 100k
cout out 0 100p
routb outb eb 100k
coutb outb 0 100p
```

```
.macro vco in inb out outb f0=100k kf=50k phi=0.0
+ out_off=0.0 out_amp=1.0
gs 0 s poly(2) c 0 in inb 0 '6.2832e-9*f0'
+ 0 0 '6.2832e-9*kf'
gc c 0 poly(2) s 0 in inb 0 '6.2832e-9*f0'
+ 0 0 '6.2832e-9*kf'
cs s 0 1e-9
cc c 0 1e-9
e1 s_clip 0 pwl(1) s 0 -0.1,-0.1 0.1,0.1
e out 0 s_clip 0 out_off '10*out_amp'
eb outb 0 s_clip 0 out_off '-10*out_amp'
.ic v(s)='sin(phi)' v(c)='cos(phi)'
.eom

.macro pd in inb in2 in2b out outb kd=0.1 out_off=0
e1 clip1 0 pwl(1) in inb -0.1,-0.1 0.1,0.1
e2 clip2 0 pwl(1) in2 in2b -0.1,-0.1 0.1,0.1
e3 n1 0 poly(2) clip1 0 clip2 0 0 0 0 0 '78.6*kd'
e4 outb 0 n1 0 out_off 1
e5 out 0 n1 0 out_off -1
.eom

.macro vco1 in inb e7 e8 vcc vee vco_cap=228.5p
qout vcc vcc b7 npn1
qoutb vcc vcc b8 npn1
rb vcc c0 5k $ 1ma
q0 c0 b0 vee npn1
q7 vcc b7 e7 npn1
r4 vcc b7 1k
i7 e7 0 1m
q8 vcc b8 e8 npn1
r5 vcc b8 1k
i8 e8 0 1m
q9 b7 e8 e9 npn1
q10 b8 e7 e10 npn1
c0 e9 e10 vco_cap
q11 e9 in 2 npn1 $ ic=i0
q12 e10 in 2 npn1 $ ic=i0
q15 2 c0 b0 npn1 $ ic=2*i0
q16 3 c0 b0 npn1 $ ic=2*i0
rx 2 3 8k
q13 vcc inb 3 npn1
q14 vcc inb 3 npn1
rt b0 vee 350 $ i=4*i0=2m
.eom
```

```
.model npn1 npn
+ eg=1.1 af=1 xcjc=0.95 subs=1
+ cjs=0 tf=5p
+ tr=500p cje=0.2p cjc=0.2p fc=0.8
+ vje=0.8 vjc=0.8 mje=0.33 mjc=0.33
+ rb=0 rbm=0 irb=10u
+ is=5e-15 ise=1.5e-14 isc=0
+ vaf=150 bf=100 ikf=20m
+ var=30 br=5 ikr=15m
+ rc=0 re=0
+ nf=1 ne=1.5 nc=1.2
+ tbf1=8e-03

.macro pd1 in inb in2 in2b out outb vcc vee
rl vcc n1 1k
rlb vcc n1b 1k
q3 n1 in c1 npn1
q4 n1b inb c1 npn1
q5 n1 inb c2 npn1
q6 n1b in c2 npn1
q1 c1 in2 e npn1
q2 c2 in2b e npn1
ie e 0 0.5m
c1 n1 0 1p
c1b n1b 0 1p
q7 vcc n1 e7 npn1
q8 vcc n1b e8 npn1
r1 e7 out 625
r2 out vee 300
r1b e8 outb 625
r2b outb vee 300
.eom

.end
```

Figure 17-37: Phase Detector Circuit



References

1. Chua & Lin. *Computer Aided Analysis of Electronic Circuit*s. Englewood Cliffs: Prentice-Hall, 1975, page 117. See also “SPICE2 Application Notes for Dependent Sources,” by Bert Epler, *IEEE Circuits & Devices Magazine*, September 1987.

Avant!

Chapter 18
Performing Pole/Zero Analysis

Pole/zero analysis is a useful method for studying the behavior of linear, time-invariant networks, and may be applied to the design of analog circuits, such as amplifiers and filters. It may be used for determining the stability of a design, and it may also be used to calculate the poles and zeroes for specification in a POLE statement as Using Pole/Zero Analysis on page 18-3 describes.

Pole/zero analysis is characterized by the use of the .PZ statement, as opposed to pole/zero and Laplace transfer function modeling, which employ the LAPLACE and POLE functions respectively. These are described in Using Pole/Zero Analysis on page 18-3.

This chapter covers these topics:

Understanding Pole/Zero Analysis

In pole/zero analysis, a network is described by its network transfer function which, for any linear time-invariant network, can be written in the general form:

$$H(s) = \frac{N(s)}{D(s)} = \frac{a_0 s^m + a_1 \cdot s^{(m-1)} + \ldots + a_m}{b_0 s^n + b_1 \cdot s^{(n-1)} + \ldots + b_n}$$

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 called the zeros of the network function, and the roots of the denominator D(s) (that is, p_j) are called the poles of the network function. S is a complex frequency[1].

The dynamic behavior of the network depends upon the location of the poles and zeros on the network function curve. The poles are called the natural frequencies of the network. In general, you can graphically deduce the magnitude and phase curve of any network function from the location of its poles and zeros (see 2 on page 18-20).

References on page 18-20, lists a variety of source material addressing transfer functions of physical systems[3], design of systems and physical modeling (see 4 on page 18-20), and interconnect transfer function modeling (see 5 on page 18-20).

Using Pole/Zero Analysis

Star-Hspice uses the Muller method (see 7 on page 18-20) to calculate the roots of polynomials N(s) and D(s). This method approximates the polynomial with a quadratic equation that fits through three points in the vicinity of a root. Successive iterations toward a particular root are obtained by finding the nearer root of a quadratic whose curve passes through the last three points.

In Muller's method, the selection of the three initial points affects the convergence of the process and accuracy of the roots obtained. If the poles or zeros are spread over a wide frequency range, choose (X0R, X0I) close to the origin to find poles or zeros at zero frequency first. Then find the remaining poles or zeros in increasing order. The values (X1R, X1I) and (X2R, X2I) may be orders of magnitude larger than (X0R, X0I). If there are poles or zeros at high frequencies, X1I and X2I should be adjusted 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.

.PZ (Pole/Zero) Statement

The syntax is:

```
.PZ output input
```

PZ	Invokes the pole/zero analysis
input	Input source, which may be any independent voltage or current source name
output	Output variables, which may be any node voltage, V(n), or any branch current, I(element name)

Example

```
.PZ   V(10)   VIN
.PZ   I(RL)   ISORC
.PZ   I1(M1) VSRC
```

Pole/Zero Control Options

CSCAL Sets the capacitance scale. Capacitances are multiplied by CSCAL. Default=1e+12.

FMAX Sets the maximum pole and zero angular frequency value. Default=1.0e+12 rad/sec.

FSCAL Sets the frequency scale. Frequency is multiplied by FSCAL. Default=1e-9.

GSCAL Sets the conductance scale. Conductance is multiplied, and resistance is divided, by GSCAL. Default=1e+3.

ITLPZ Sets the pole/zero analysis iteration limit. Default=100.

LSCAL Sets the inductance scale. Inductances are multiplied by LSCAL. Default=1e+6.

Note: Scale factors must satisfy the following relations.

$$GSCAL = CSCAL \cdot FSCAL$$

$$GSCAL = \frac{1}{LSCAL \cdot FSCAL}$$

If scale factors are changed, the initial Muller points, (X0R, X0I), (X1R, X1I) and (X2R, X2I), may have to be modified, even though internally the program multiplies the initial values by (1e-9/GSCAL).

PZABS Sets absolute tolerances for poles and zeros. This option affects the low frequency poles or zeros. It is used as follows:

$$\text{If } (|X_{real}| + |X_{imag}| < PZABS),$$

$$\text{then } X_{real} = 0 \text{ and } X_{imag} = 0 .$$

This option is also used for convergence tests. Default=1e-2.

PZTOL	Sets the relative error tolerance for poles or zeros. Default=1.0e-6.
RITOL	Sets the minimum ratio value for (real/imaginary) or (imaginary/real) parts of the poles or zeros. Default1.0e-6. RITOL is used as follows: If $\lvert X_{imag}\rvert \le RITOL \cdot \lvert X_{real}\rvert$, then $X_{imag} = 0$ If $\lvert X_{real}\rvert \le RITOL \cdot \lvert X_{imag}\rvert$, then $X_{real} = 0$
(X0R,X0I) *(x1R,X1I)* *(X2R,X21)*	the three complex starting trial points in the Muller algorithm for pole/zero analysis. Defaults: X0R=-1.23456e6 X0I=0.0 X1R=1.23456e5 X1I=0.0 X2R=+1.23456e6 X21=0.0 These initial points and FMAX are multiplied by FSCAL.

Pole/Zero Analysis Examples

Example 1 – Low-Pass Filter

The following is an HSPICE input file for a low-pass prototype filter for pole/zero and AC analysis (see 8 on page 18-20). This file is in the *$installdir/demo/hspice/filters/flp5th.sp* directory.

Fifth-Order Low-Pass Filter HSPICE File

```
*FILE: FLP5TH.SP
5TH-ORDER LOW_PASS FILTER
****
* T = I(R2) / 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)
*****

.OPTIONS POST
.PZ I(R2) IN
.AC DEC 100  .001HZ 10HZ
```

```
.PLOT AC IDB(R2) IP(R2)
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 18-1: Low-Pass Prototype Filter

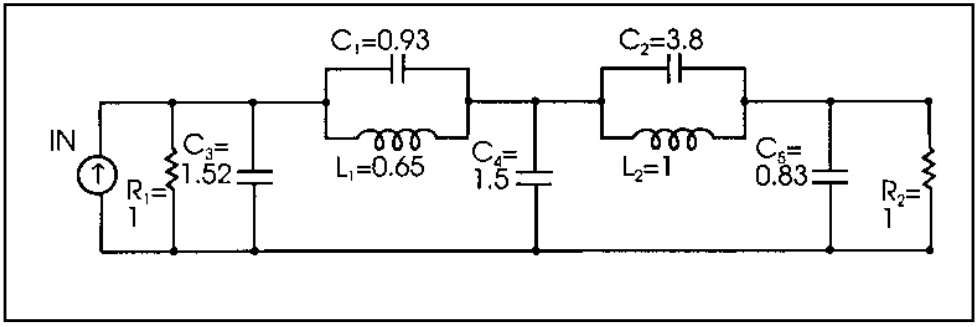


Table 18-1 shows the magnitude and phase variation of the current output resulting from AC analysis. These results are consistent with the pole/zero analysis. The pole/zero unit is radians per second or hertz. The X-axis unit in the plot is in hertz.

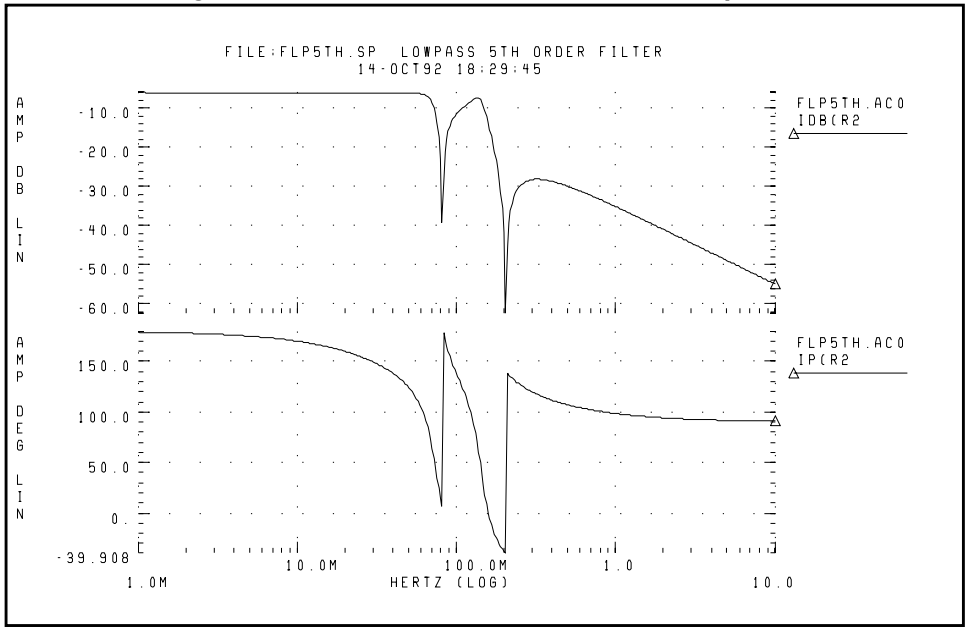
Table 18-1: Pole/Zero Analysis Results for Low-Pass Filter

Poles (rad/sec)		Poles (hertz)	
Real	**Imag**	**Real**	**Imag**
-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

Table 18-1: Pole/Zero Analysis Results for Low-Pass Filter *(Continued)*

-1.182742e-01	8.914907e-01	-1.882392e-02	1.418852e-01
-5.450890e-01	0.000000e+00	-8.675361e-02	0.000000e+00
Zeros (rad/sec)		**Zeros (hertz)**	
Real	**Imag**	**Real**	**Imag**
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 18-2: Fifth-Order Low-Pass Filter Response



Example 2 – Kerwin's Circuit

The following is an HSPICE input file for pole/zero analysis of Kerwin's circuit (see 9 on page 18-20). This file can be found in *$installdir/demo/hspice/filters/fkerwin.sp*. Table 18-2 lists the results of the analysis.

Kerwin's Circuit HSPICE File

```
*FILE: FKERWIN.SP
KERWIN'S CIRCUIT   HAVING JW-AXIS TRANSMISSION ZEROS.
**
* T = V(5) / VIN
*   = 1.2146 (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)
*****

.PZ V(5) VIN
VIN 1 0 1
C1   1 2 0.7071
C2   2 4 0.7071
C3   3 0 1.4142
C4   4 0 0.3536
R1   1 3 1.0
R2   3 4 1.0
R3   2 5 0.5
E1   5 0 4 0 2.4293
.END
```

Figure 18-3: Design Example for Kerwin's Circuit

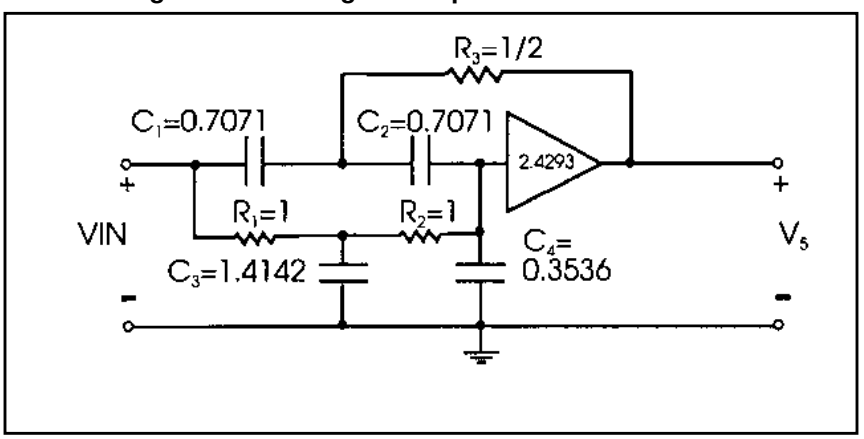


Table 18-2: Pole/Zero Analysis Results for Kerwin's Circuit

Poles (rad/sec)		Poles (hertz)	
Real	**Imag**	**Real**	**Imag**
-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
Zeros (rad/sec)		**Zeros (hertz)**	
Real	**Imag**	**Real**	**Imag**
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

The following is an HSPICE input file for pole/zero analysis of a high-pass Butterworth filter see 10 on page 18-20. This file can be found in *$installdir/demo/hspice/filters/fhp4th.sp*. Table 18-3 shows the analysis results.

Fourth-Order High-Pass Butterworth Filter HSPICE File

```
*FILE: FHP4TH.SP
*****
* T = V(10) / VIN
* = (S**4) / ((S**2 + 0.7653*S + 1) * (S**2
*  + 1.8477*S + 1))
*
* POLES, (-0.38265, +0.923895), (-0.38265, -0.923895)
*          (-0.9239, +0.3827),   (-0.9239, -0.3827)
* ZEROS, FOUR ZEROS AT (0.0, 0.0)
*****
```

```
.OPTIONS ITLPZ=200
.PZ   V(10)   VIN
VIN 1 0 1
C1   1 2 1
C2   2 3 1
R1   3 0 2.613
R2   2 4 0.3826
E1   4 0 3 0 1
C3   4 5 1
C4   5 6 1
R3   6 0 1.0825
R4   5 10 0.9238
E2   10 0 6 0 1
RL   10 0 1E20
.END
```

Figure 18-4: Fourth-Order High-Pass Butterworth Filter

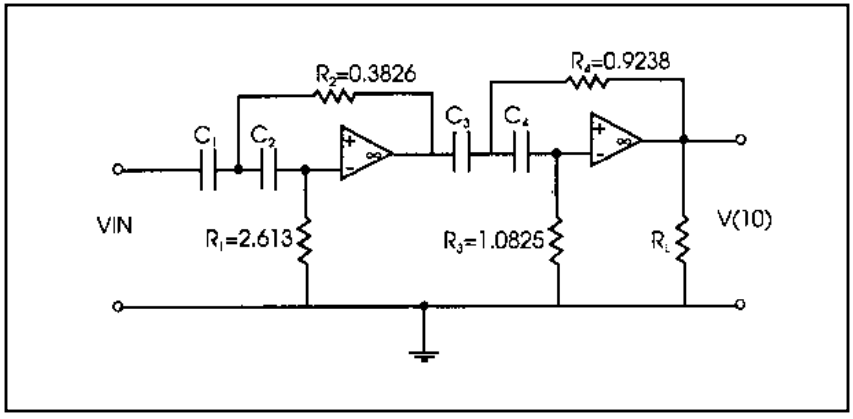


Table 18-3: Pole/Zero Analysis Results for High-Pass Butterworth Filter

Poles (rad/sec)		Poles (hertz)	
Real	**Imag**	**Real**	**Imag**
-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
Zeros (rad/sec)		**Zeros (hertz)**	
Real	**Imag**	**Real**	**Imag**
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

The following is an HSPICE input file for pole/zero analysis of a CMOS differential amplifier for pole/zero and AC analysis. The file can be found in *$installdir/demo/hspice/apps/mcdiff.sp*. Table 18-4 shows the analysis results.

CMOS Differential Amplifier HSPICE File

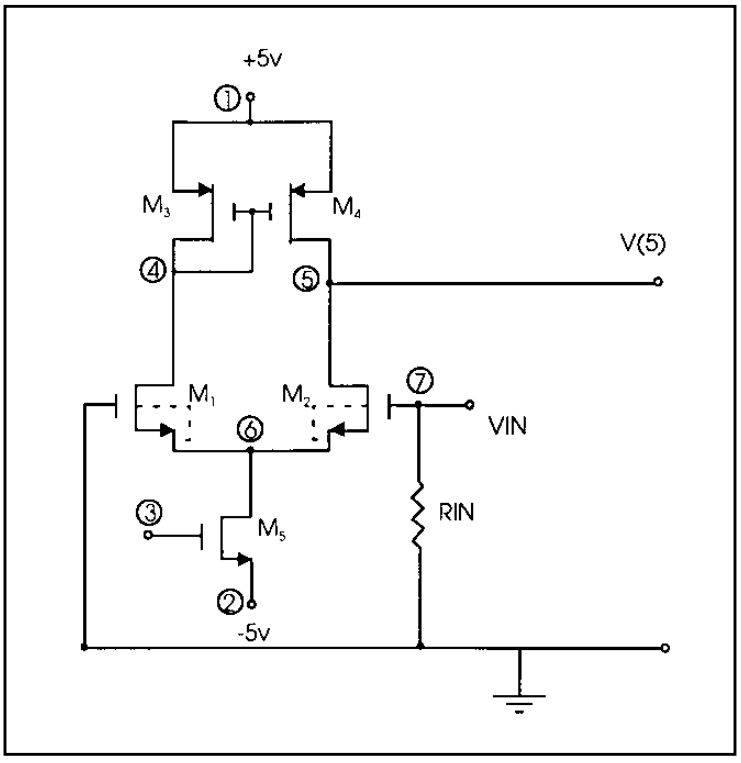
```
FILE: MCDIFF.SP
CMOS DIFFERENTIAL AMPLIFIER

.OPTIONS PIVOT SCALE=1E-6 SCALM=1E-6  WL
.PZ V(5) VIN
VIN 7 0 0 AC 1
.AC DEC 10 20K 500MEG
```

```
.PRINT 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
+ 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 18-4: Pole/Zero Analysis Results for CMOS Differential Amplifier

Poles (rad/sec)		**Poles (hertz)**	
Real	**Imag**	**Real**	**Imag**
-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
Zeros (rad/sec		**Zeros (hertz)**	
Real	**Imag**	**Real**	**Imag**
-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 18-5: CMOS Differential Amplifier



Example 5 – Simple Amplifier

The following is an HSPICE input file for pole/zero analysis of an equivalent circuit of a simple amplifier with RS =RPI=RL=1000 ohms, gm=0.04 mho, CMU=1.0e-11 farad, and CPI =1.0e-9farad (see 11 on page 18-20). The file can be found in *$installdir/demo/hspice/apps/ampg.sp*. The analysis results are shown in Table 18-5.

Amplifier HSPICE File

```
FILE: AMPG.SP
A SIMPLE AMPLIFIER.

* T = V(3) / VIN
* T = 1.0D6*(S - 4.0D9) / (S**2 + 1.43D8*S + 2.0D14)
* POLES = (-0.14D7, 0.0), (-14.16D7, 0.0)
* ZEROS = (+4.00D9, 0.0)
```

```
.PZ   V(3)   VIN
RS   1 2 1K
RPI 2 0 1K
RL   3 0 1K
GMU 3 0 2 0 0.04
CPI 2 0 1NF
CMU 2 3 10PF
VIN 1 0 1
.END
```

Figure 18-6: Simple Amplifier

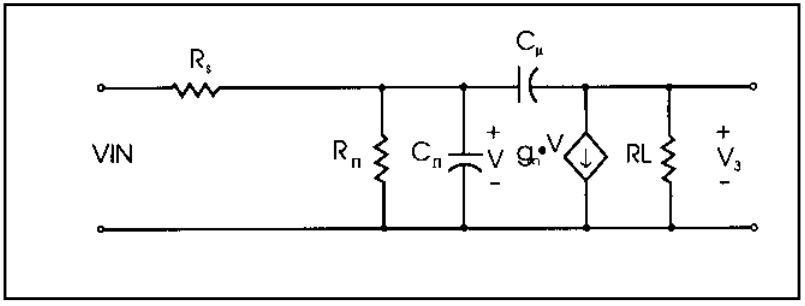


Table 18-5: Pole/Zero Analysis Results for Amplifier

Poles (rad/sec)		**Poles (hertz)**	
Real	**Imag**	**Real**	**Imag**
-1.412555+06	0.000000e+00	-2.248151e+05	0.000000e+00
-1.415874+08	0.000000e+00	-2.253434e+07	0.000000e+00
Zeros (rad/sec)		**Zeros (hertz)**	
Real	**Imag**	**Real**	**Imag**
4.000000e+09	0.000000e+00	6.366198e+08	0.000000e+00
Constant Factor = 1.000000e+06			

Example 6— Active Low-Pass Filter

The following is an HSPICE input file for pole/zero analysis of an active ninth-order low-pass filter (see 12 on page 18-20) using the ideal op-amp element. AC analysis is performed. The file can be found in *$installdir/demo/hspice/filters/flp9th.sp*. Table 18-6 shows the analysis results.

Ninth Order Low-Pass Filter HSPICE File

```
FILE: FLP9TH.SP
******
VIN IN 0 AC 1
.PZ V(OUT) VIN
.AC DEC 50  .1K 100K

.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

.PLOT AC VDB(OUT)
.SUBCKT OPAMP IN+ IN- OUT GM1=2 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 18-7: Linear Model of the 741C Op-Amp

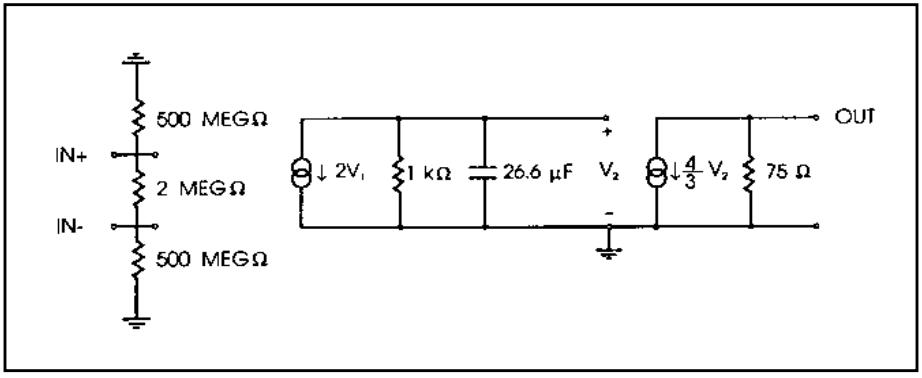


Figure 18-8: The FDNR Subcircuit

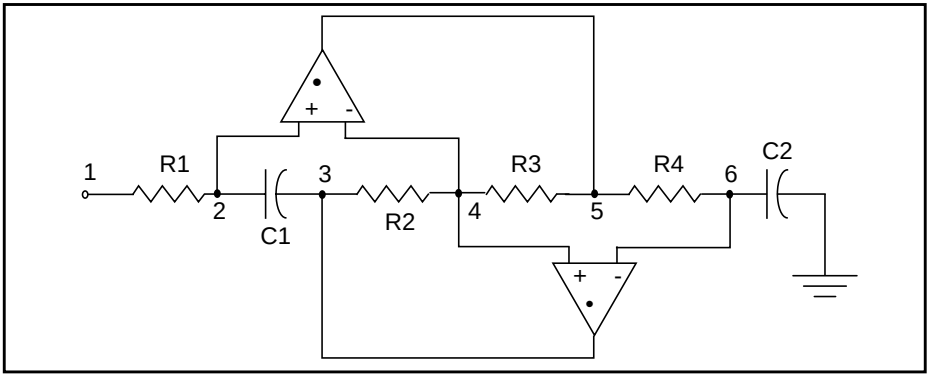


Figure 18-9: Active Realization of the Low-Pass Filter

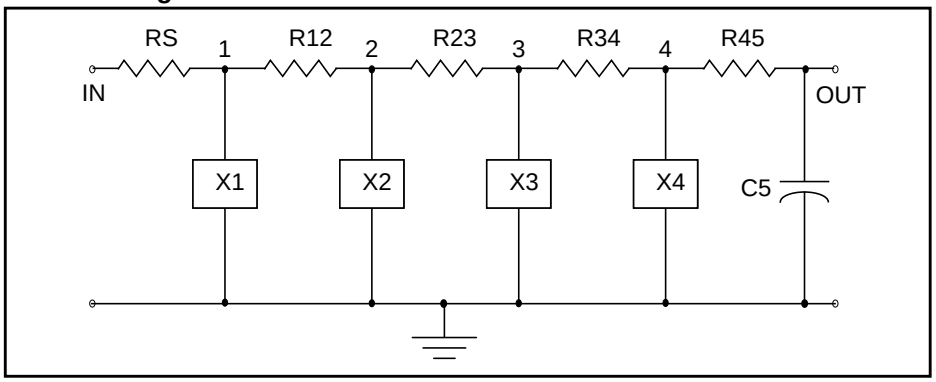


Table 18-6: Pole/Zero Analysis Results for the Active Low-Pass Filter *(Sheet 1 of 3)*

Poles (rad/sec)		Poles (hertz)	
Real	**Imag**	**Real**	**Imag**
-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

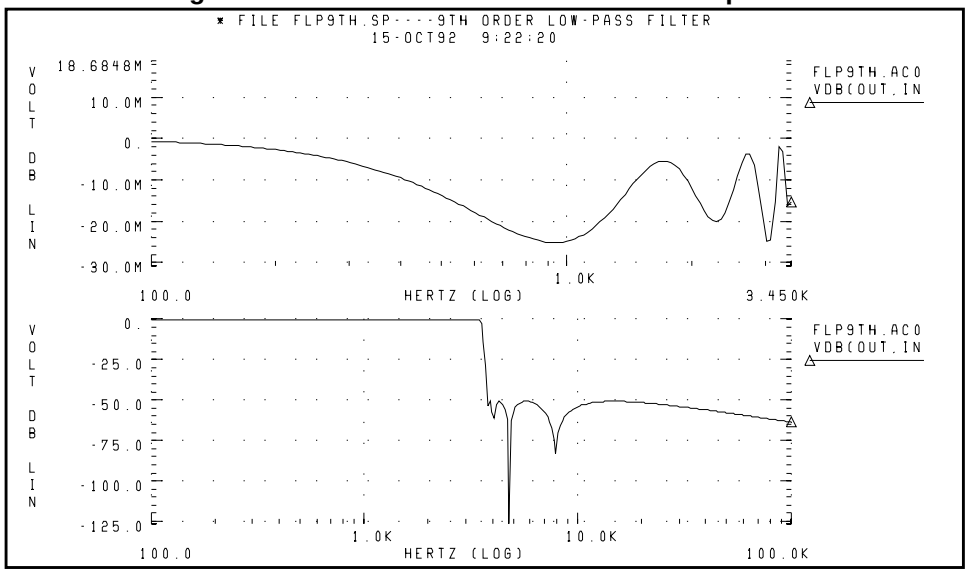
Table 18-6: Pole/Zero Analysis Results for the Active Low-Pass Filter *(Sheet 2 of 3)*

-3.676439e+06	3.676184e+06	-5.851234e+05	5.850828e+05
-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
Zeroes (rad/sec)		**Zeroes (hertz)**	
Real	**Imag**	**Real**	**Imag**
-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

Table 18-6: Pole/Zero Analysis Results for the Active Low-Pass Filter *(Sheet 3 of 3)*

-3.693882e+06	3.693739e+06	5.878995e+05	5.878768e+05
-3.693882e+06	-3.693739e+06	-5.878995e+05	-5.878768e+05
Constant Factor = 4.451586e+02			

Figure 18-10: 9th Order Low-Pass Filter Response



The top graph in Table 18-10 plots the bandpass response of the Pole/Zero Example 6 low-pass filter. The bottom graph shows the 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 on 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 on interconnect transfer function modeling which deals with transfer function extraction 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 provides another source of interconnect transfer function modeling.
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.

Avant!

Chapter 19

Performing FFT Spectrum Analysis

Spectrum analysis represents a time-domain signal within the frequency domain. It most commonly employs the Fourier transform. A Discrete Fourier Transform (DFT) determines the frequency content of analog signals found in circuit simulation, using sequences of time values. The Fast Fourier Transform (FFT) calculates the DFT, which Star-Hspice uses for spectrum analysis.

The .FFT statement in Star-Hspice uses the internal time point values and, by default, through a second-order interpolation, obtains waveform samples based on the number of points that you specify.

Note: New accuracy improvement feature added in the Hspice 99.4 release. The .option fft_accurate or .option accurate (which internally turns on the fft_accurate option) will force Hspice to dynamically adjust the time step so that each FFT point will be a real simulation point. This eliminates the interpolation error and provides the highest FFT accuracy with minimal overhead in simulation time.

Windowing functions, can be used to reduce the effects of truncation of the waveform on the spectral content. The .FFT command also allows you to specify the desired output format, to specify a frequency of interest, and to obtain any number of harmonics, as well as the total harmonic distortion (THD).

This chapter describes the following topics:

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 because of “spectral leakage,” due primarily to the periodic extension assumption underlying DFT.

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 FFT is used for signal detection or estimation – that is, for detecting weak signals in the presence of strong signals or resolving 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 that peak in the middle frequencies and decrease to zero at the edges, thus reducing the effects of the discontinuities as a result of finite duration. Figure 19-1 shows the windows available in Star-Hspice. Table 19-1 lists the common performance parameters for FFT windows available in Star-Hspice.

Figure 19-1: FFT Windows

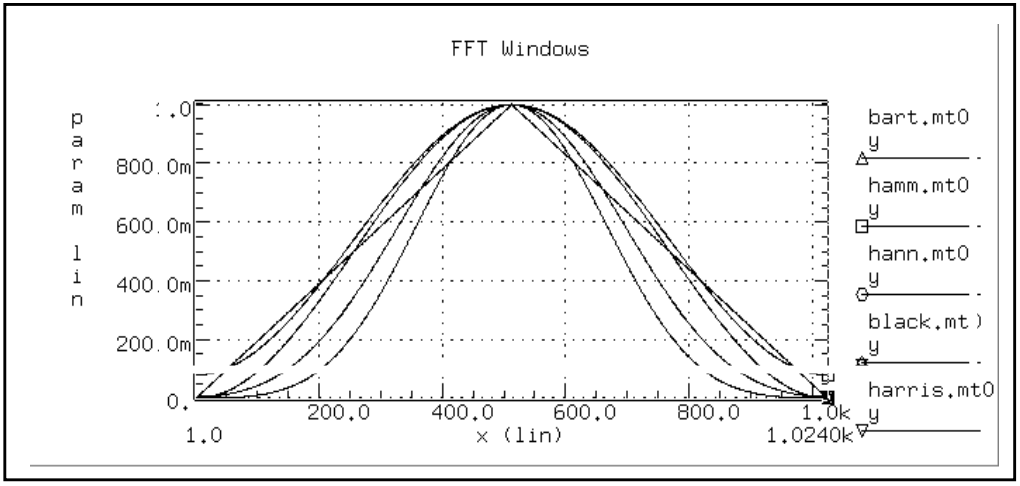


Table 19-1: Window Weighting Characteristics in FFT Analysis

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	$W(n)=1$, $0 \le n < NP^{\dagger}$	-13	-6	0.89	3.92
Bartlett	$W(n)=2n/(NP-1)$, $0 \le n \le (NP/2)-1$ $W(n)=2-2n/(NP-1)$, $NP/2 \le n < NP$	-27	-12	1.28	3.07
Hanning	$W(n)=0.5-0.5[\cos(2\pi n/(NP-1))]$, $0 \le n < NP$	-32	-18	1.44	3.18
Hamming	$W(n)=0.54-0.46[\cos(2\pi n/(NP-1))]$, $0 \le n < NP$	-43	-6	1.30	3.10
Blackman	$W(n)=0.42323-0.49755[\cos(2\pi n/(NP-1))]+0.07922\cos[\cos(4\pi n/(NP-1))]$, $0 \le n < NP$	-58	-18	1.68	3.47

Table 19-1: 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)
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 ≤ *n* < NP	-92	-6	1.90	3.85
Gaussian a=2.5 a=3.0 a=3.5	W(n)=exp[-0.5a2(NP/2-1-n)2/(NP)2], 0 ≤ n ≤ (NP/2)-1 W(n)=exp[-0.5a2(n-NP/2)2/(NP)2], NP/2 ≤ n < NP	 -42 -55 -69	 -6 -6 -6	 1.33 1.55 1.79	 3.14 3.40 3.73
Kaiser-Bessel a=2.0 a=2.5 a=3.0 a=3.5	W(n)=I0(x2)/I0(x1) x1=pa x2=x1*sqrt[1-(2(NP/2-1-n)/NP)2], 0 ≤ n ≤ (NP/2)-1 x2=x1*sqrt[1-(2(n-NP/2)/NP)2], NP/2 ≤ n < NP I0 is the zero-order modified Bessel function	 -46 -57 -69 -82	 -6 -6 -6 -6	 1.43 1.57 1.71 0.89	 3.20 3.38 3.56 3.74

†NP is the number of points used for the FFT analysis.

The most important parameters in Table 19-1 are the highest side-lobe level (to reduce bias, the lower the better) and the worst-case processing loss (to increase detectability, the lower the better). Some compromise usually is necessary to find a suitable window filtering for each application. As a rule, the window performance improves with functions of higher complexity (those listed lower in the table). The Kaiser window has an ALFA parameter that allows adjustment of 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 that provide smooth truncation and a wide range of side-lobe level and processing loss. The last two windows in the table are parameterized windows that allow you to adjust the side-lobe level, the 3 dB bandwidth, and the processing loss.[1]

Figure 19-2 and Figure 19-3 show the characteristics of two typical windows.

Figure 19-2: Bartlett Window Characteristics

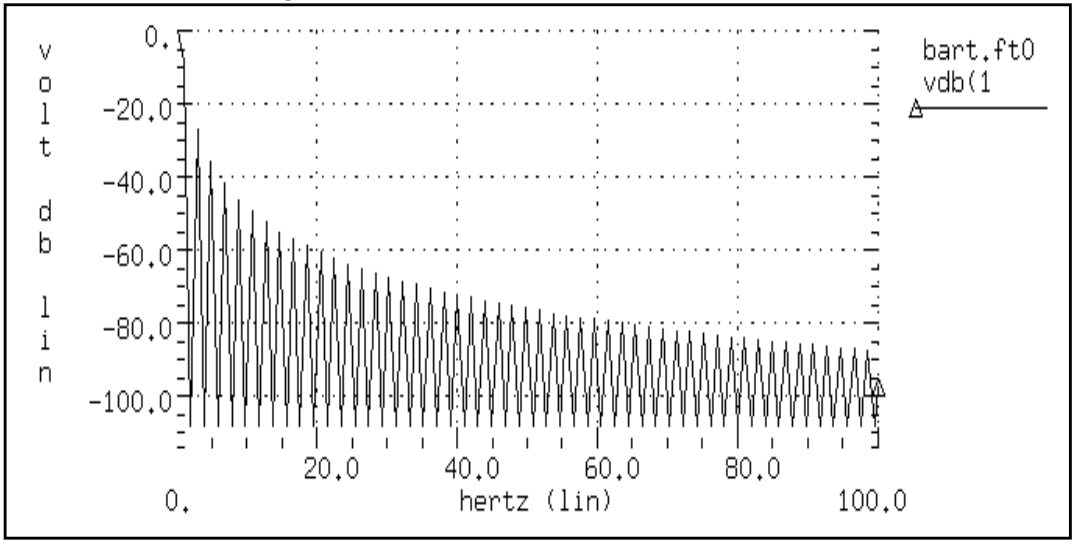
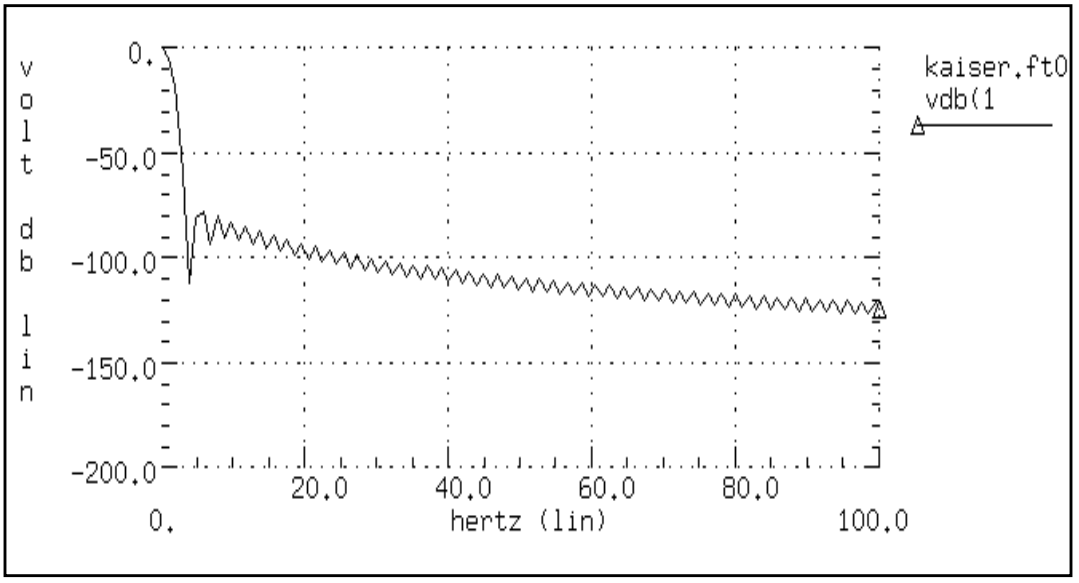


Figure 19-3: Kaiser-Bessel Window Characteristics, ALFA=3.0



Using the .FFT Statement

The general form of the .FFT statement is shown below. The parameters are described in Table 19-2.

Syntax

```
.FFT <output_var> <START=value> <STOP=value> <NP=value>
+ <FORMAT=keyword> <WINDOW=keyword> <ALFA=value>
+ <FREQ=value> <FMIN=value> <FMAX=value>
```

Table 19-2: .FFT Statement Parameters

Parameter	Default	Description
output_var		Can be any valid output variable, such as voltage, current, or power
START	see Description	Specifies the beginning of the output variable waveform to analyze. Default is the START value in the .TRAN statement, which defaults to 0 s.
FROM	see START	An alias for START in .FFT statements
STOP	see Description	Specifies the end of the output variable waveform to be analyzed. Defaults to the TSTOP value in the .TRAN statement.
TO	see STOP	An alias for STOP in .FFT statements
NP	1024	Specifies the number of points used in the FFT analysis. NP must be a power of 2; if NP is not a power of 2, Star-Hspice automatically adjusts it to the closest higher number that is a power of 2.
FORMAT	NORM	Specifies the output format: ■ NORM= normalized magnitude ■ UNORM=unnormalized magnitude

Table 19-2: .FFT Statement Parameters *(Continued)*

Parameter	Default	Description
WINDOW	RECT	Specifies the window type to be used: ■ RECT=simple rectangular truncation window ■ BART=Bartlett (triangular) window ■ HANN=Hanning window ■ HAMM=Hamming window ■ BLACK=Blackman window ■ HARRIS=Blackman-Harris window ■ GAUSS=Gaussian window ■ KAISER=Kaiser-Bessel window
ALFA	3.0	Specifies the parameter used in GAUSS and KAISER windows to control the highest side-lobe level, bandwidth, and so on 1.0 <= ALFA <= 20.0
FREQ	0.0 (Hz)	Specifies a frequency of interest. If FREQ is nonzero, the output listing is limited to the harmonics of this frequency, based on FMIN and FMAX. The THD for these harmonics also is printed.
FMIN	1.0/T (Hz)	Specifies the minimum frequency for which FFT output is printed in the listing file or which is used in THD calculations. T = (STOP-START)
FMAX	0.5*NP*FMIN (Hz)	Specifies the maximum frequency for which FFT output is printed in the listing file or which is used in THD calculations.

Example

Below are four examples of valid .FFT statements.

```
.fft v(1)

.fft v(1,2) np=1024 start=0.3m stop=0.5m freq=5.0k
+ window=kaiser alfa=2.5

.fft I(rload) start=0m to=2.0m fmin=100k fmax=120k
+ format=unorm

.fft par('v(1) + v(2)') from=0.2u stop=1.2u
+ window=harris
```

Only one output variable is allowed in an .FFT command. The following is an incorrect use of the command.

```
.fft v(1) v(2) np=1024
```

The correct use of the command is shown in the example below. In this case, an *.ft0* and an *.ft1* file are generated for the FFT of v(1) and v(2), respectively.

```
.fft v(1) np=1024
.fft v(2) np=1024
```

Examining the FFT Output

Star-Hspice prints the results of the FFT analysis in a tabular format in the *.lis* file, based on the parameters in the .FFT statement. The normalized magnitude values are printed unless you specify FORMAT= UNORM, in which case unnormalized magnitude values are printed. The number of printed frequencies is half the number of points (NP) specified in the .FFT statement.

If you specify a minimum or a maximum frequency using FMIN or FMAX, the printed information is limited to the specified frequency range. Moreover, if you specify a frequency of interest using FREQ, then the output is limited to the harmonics of this frequency, along with the percent of total harmonic distortion.

In the sample output below, 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 THD up to 10 kHz, which corresponds to the first ten harmonics.

Note: The highest frequency in the Star-Hspice FFT output might not match the specified FMAX, due to adjustments that Star-Hspice makes.

Table 19-3 describes the output of the Star-Hspice FFT analysis.

Table 19-3: .FFT Output Description

Column Heading	Description
Frequency Index	Runs from 1 to NP/2, or the corresponding index for FMIN and FMAX. Note that the DC component corresponding to the index 0 is displayed independently.
Frequency	The actual frequency associated with the index
fft_mag (dB), fft_mag	There are two FFT magnitude columns, the first in dB and the second in the units of the output variable. The magnitude is normalized unless UNORM format is specified.
fft_phase	The associated phase, in degrees

A *.ft#* file is generated, in addition to the listing file, for each FFT output variable. The *.ft#* file contains the graphical data needed to display the FFT analysis results in AvanWaves. The magnitude in dB and the phase in degrees are available for display.

Notes:

1. The following formula should be used as a guideline when specifying a frequency range for FFT output:

   ```
   frequency increment = 1.0/(STOP - START)
   ```

   Each frequency index corresponds to a multiple of this increment. Hence, to obtain a finer frequency resolution you should maximize the duration of the time window.

2. FMIN and FMAX have no effect on the *.ft0*, *.ft1*, ..., *.ftn* files.

AM Modulation

This example input listing on the following page shows a 1 kHz carrier (FC) that is modulated by a 100 Hz signal (FM). The voltage at node 1, which is an AM signal, can be described by

$$1) = sa \cdot (offset + \sin(\omega_m(Time - td))) \cdot \sin(\omega_c(Time - td$$

The preceding equation can be expanded 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 frequency

$$f_c\,,\ (f_c - f_m)\ \text{, and}\ (f_c + f_m)$$

namely, the carrier frequency and the two sidebands.

Input Listing

```
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 0.01m 52m
.FFT V(1) START=10m STOP=40m FMIN=833 FMAX=1.16K
.END
```

Output Listing

The relevant portion of the listing file is shown below.

```
*********
am modulation
****** fft analysis  tnom= 25.000 temp= 25.000
****** fft components of transient response v(1)
Window:       Rectangular
Start Freq:   833.3333
Stop Freq:    1.1667k

dc component: mag(db)= -1.480D+02 mag=   3.964D-08
+ phase=   0.000D+00

frequency frequency  fft_mag     fft_mag    fft_phase
index      (hz)         (db)                  (deg)
 25     833.3333     -129.4536   336.7584n   -113.0047
 26     866.6667     -143.7912    64.6308n     45.6195
 27     900.0000       -6.0206   500.0008m     35.9963
 28     933.3333     -125.4909   531.4428n    112.6012
 29     966.6667     -142.7650    72.7360n    -32.3152
 30       1.0000k       0.          1.0000     -90.0050
 31       1.0333k    -132.4062   239.7125n     -9.0718
 32       1.0667k    -152.0156    25.0738n      3.4251
 33       1.1000k      -6.0206   499.9989m    143.9933
 34       1.1333k    -147.0134    44.5997n     -3.0046
 35       1.1667k    -147.7864    40.8021n     -4.7543

         ***** job concluded
```

Graphical Output

Figure 19-4 and Figure 19-5 display the results. Figure 19-4 shows the time domain curve of node 1. Figure 19-5 shows the frequency domain components of the magnitude of node 1. Note the carrier frequency at 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. Since the time domain data contains an integer multiple of the period, no windowing is needed.

Figure 19-4: AM Modulation

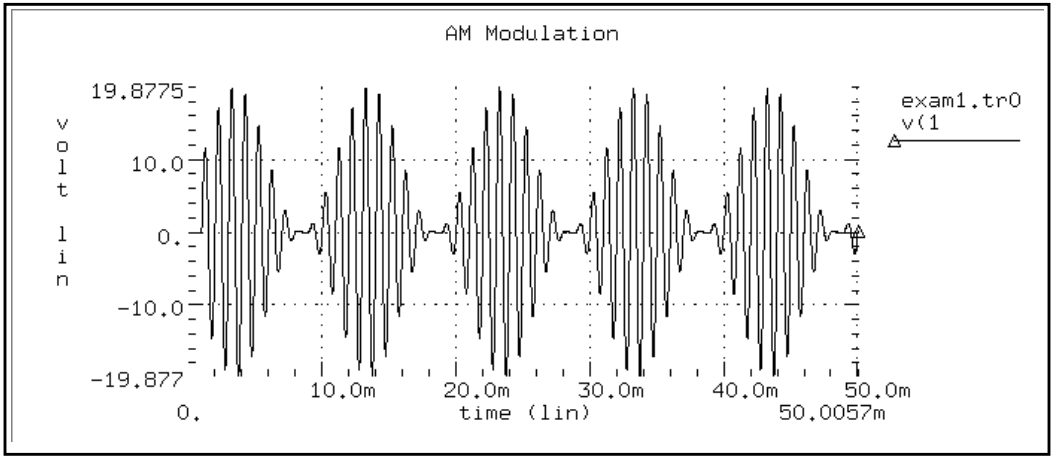
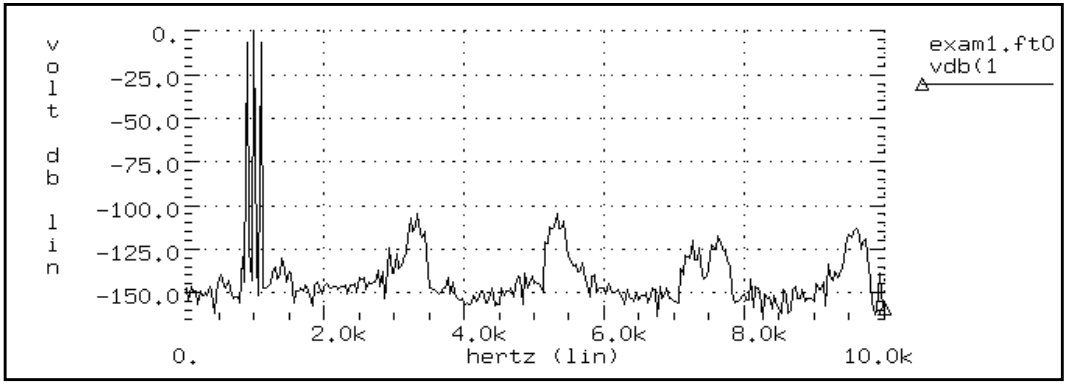


Figure 19-5: AM Modulation Spectrum



Balanced Modulator and Demodulator

Demodulation, or detection, is the process of recovering a modulating signal from the modulated output voltage. The netlist below illustrates this process, using Star-Hspice behavioral models and FFT analysis to confirm the validity of the process in the frequency domain. The Laplace element is used in the low-pass filter. This filter introduces some delay in the output signal, which causes spectral leakage if no windowing is used in FFT. However, when window weighting is used to perform FFT, the spectral leakage is virtually eliminated. This can be verified from the THD of the two outputs shown in the output listing that follows. Since a 1 kHz output signal is expected, a frequency of 1 kHz is specified in the .FFT command. Additionally, specifying the desired FMAX provides the first few harmonics in the output listing for THD calculations.

Input Listing

```
Balanced Modulator & Demodulator Circuit
V1 mod1 GND sin(0 5 1K 0 0 0) $ modulating signal
r1 mod1 2 10k
r2 2 3 10k
r3 2 GND 10K

E1 3 GND OPAMP 2 GND
$ buffered output of modulating signal
V2 mod2 GND sin(0 5 10K 0 0 0) $ modulated signal
E2 modout GND vol='(v(3)*v(mod2))/10.0'
$ multiply to modulate

V3 8 GND sin(0 5 10K 0 0 0)
E3 demod GND vol='(v(modout)*v(8))/10.0'
$ multiply to demodulate
* use a laplace element for filtering

E_filter lpout 0 laplace demod 0 67.11e6 / 66.64e6
+ 6.258e3 1.0 $ filter out +the modulating signal
*
.tran 0.2u 4m
.fft v(mod1)
.fft v(mod2)
.fft v(modout)
.fft v(demod)
```

```
.fft v(lpout) freq=1.0k fmax=10k
$ ask to see the first few harmonics

.fft v(lpout) window=harris freq=1.0k fmax=10k
$ window should reduce spectral leakage

.probe tran v(mod1) V(mod2) v(modout) v(demod) v(lpout)
.option acct post probe
.end
```

Output Listing

The relevant portion of the output listing is shown below to illustrate the effect of windowing in reducing spectral leakage and consequently, reducing the THD.

```
balanced modulator & demodulator circuit
****** fft analysis  tnom= 25.000 temp= 25.000
****** fft components of transient response v(lpout)
Window:           Rectangular
First Harmonic:   1.0000k
Start Freq:       1.0000k
Stop Freq:        10.0000k

dc component: mag(db)= -3.738D+01 mag=   1.353D-02
+ phase=   1.800D+02

frequency  frequency  fft_mag     fft_mag   fft_phase
index        (hz)      (db)                   (deg)
  4       1.0000k       0.         1.0000     35.6762
  8       2.0000k     -26.6737    46.3781m    122.8647
 12       3.0000k     -31.4745    26.6856m    108.1100
 16       4.0000k     -34.4833    18.8728m    103.6867
 20       5.0000k     -36.6608    14.6880m    101.8227
 24       6.0000k     -38.3737    12.0591m    100.9676
 28       7.0000k     -39.7894    10.2455m    100.6167
 32       8.0000k     -40.9976     8.9150m    100.5559
 36       9.0000k     -42.0524     7.8955m    100.6783
 40      10.0000k     -42.9888     7.0886m    100.9240

total harmonic distortion =     6.2269   percent

******
balanced modulator & demodulator circuit
****** fft analysis  tnom= 25.000 temp= 25.000
****** fft components of transient response v(lpout)
```

```
Window:           Blackman-Harris
First Harmonic:   1.0000k
Start Freq:       1.0000k
Stop Freq:        10.0000k

dc component: mag(db)= -8.809D+01 mag=   3.938D-05
+ phase=   1.800D+02

frequency  frequency  fft_mag     fft_mag   fft_phase
index        (hz)      (db)                   (deg)
  4       1.0000k       0.         1.0000     34.3715
  8       2.0000k     -66.5109   472.5569u    -78.8512
 12       3.0000k     -97.5914    13.1956u    -55.7167
 16       4.0000k    -107.8004     4.0736u    -41.6389
 20       5.0000k    -117.9984     1.2592u    -23.9325
 24       6.0000k    -125.0965   556.1309n     33.3195
 28       7.0000k    -123.6795   654.6722n     74.0461
 32       8.0000k    -122.4362   755.4258n     86.5049
 36       9.0000k    -122.0336   791.2570n     91.6976
 40      10.0000k    -122.0388   790.7840n     94.5380

    total harmonic distortion =    47.2763m percent
*******
```

The signals and their spectral content are shown in Figure 19-6 through Figure 19-14. 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 the carrier frequency is recovered in the demodulated signal, which also shows a 10 kHz frequency shift in the above signals (to 19 kHz and 21 kHz).

A low-pass filter is used to extract the carrier frequency using a second order Butterworth filter. Use of 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, it appears that a filter with a steeper transition region and better delay characteristics is needed to suppress the modulating frequencies below the -60 dB level. The “Filtered Output Signal” waveform in Figure 19-9 is normalized.

Figure 19-6: Modulating and Modulated Signals

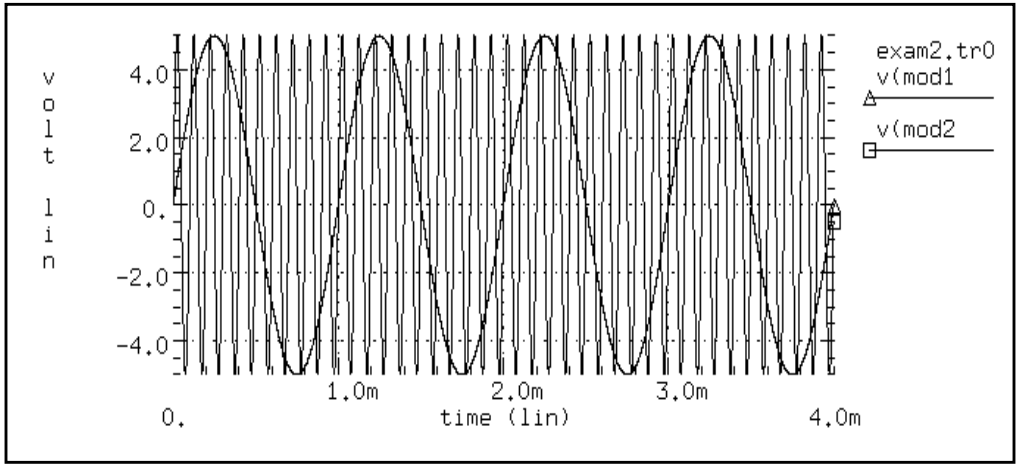


Figure 19-7: Modulated Signal

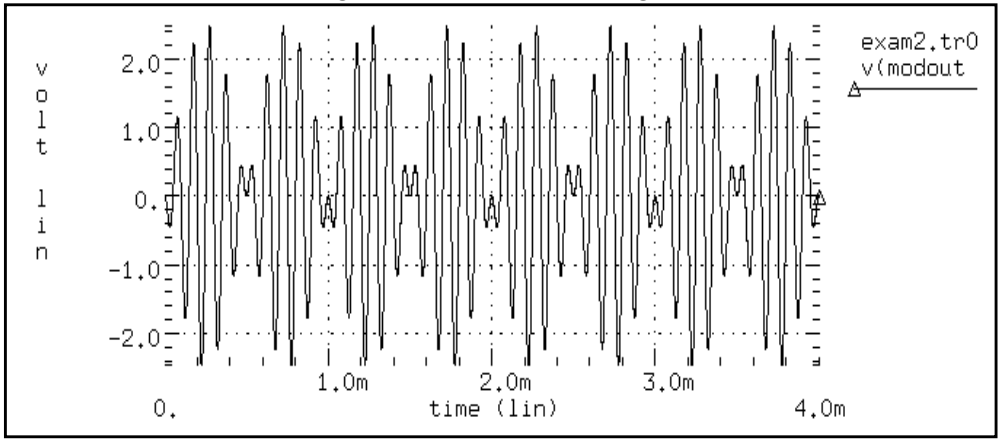


Figure 19-8: Demodulated Signal

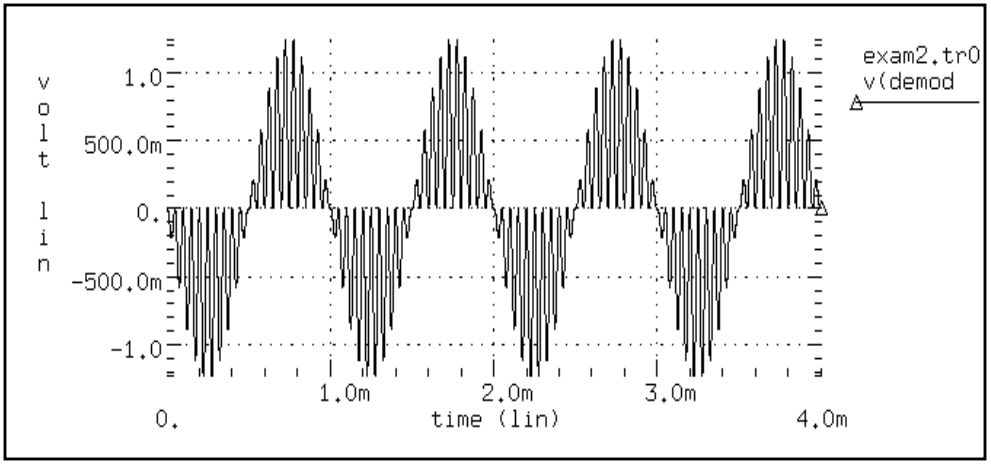


Figure 19-9: Filtered Output Signal

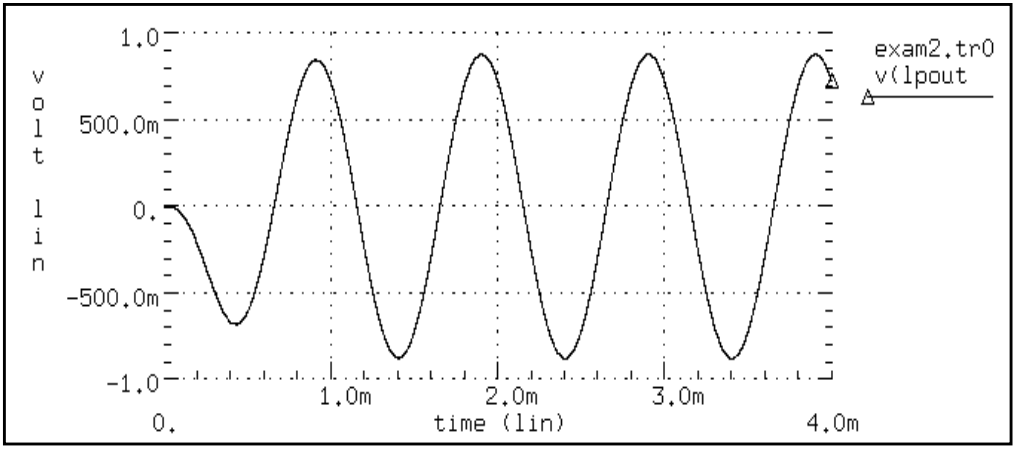


Figure 19-10: Modulating and Modulated Signal Spectrum

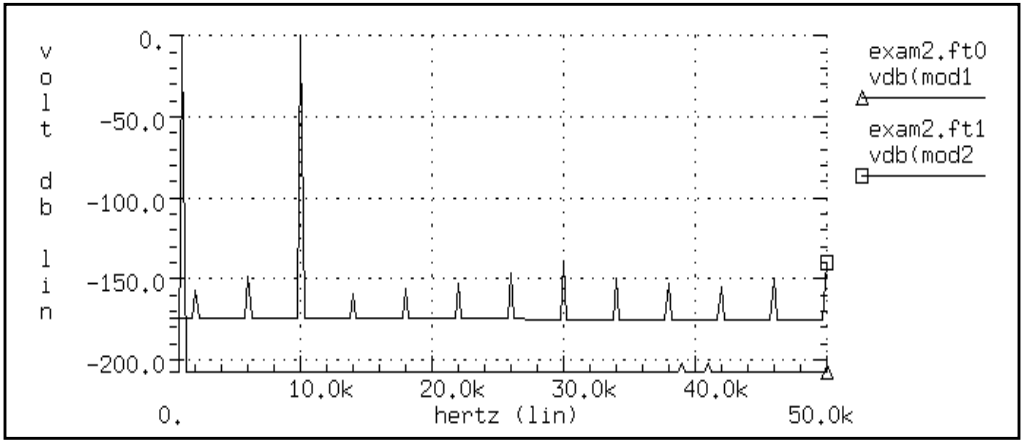


Figure 19-11: Modulated Signal Spectrum

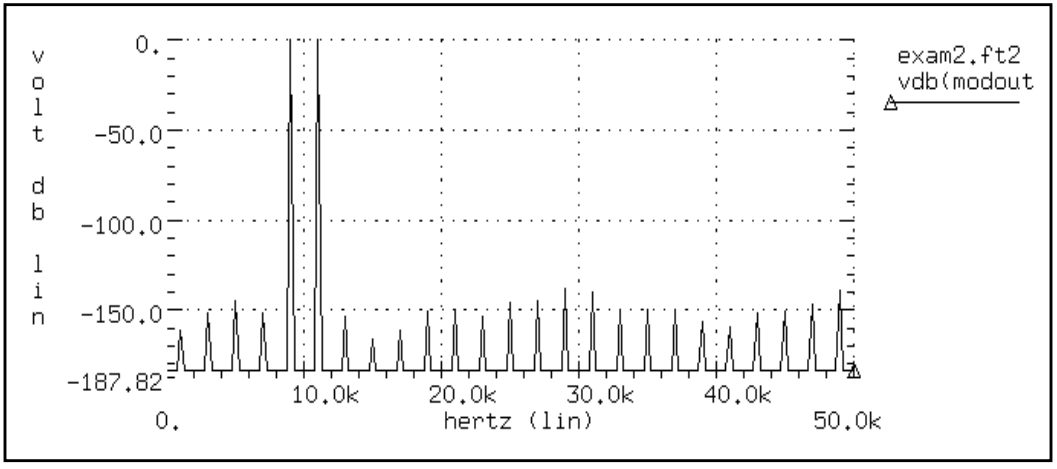


Figure 19-12: Demodulated Signal Spectrum

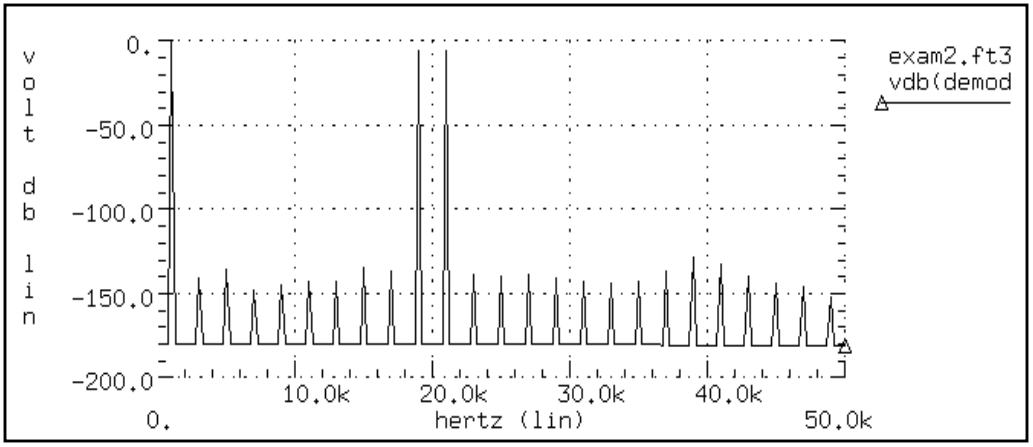


Figure 19-13: Filtered Output Signal (no window)

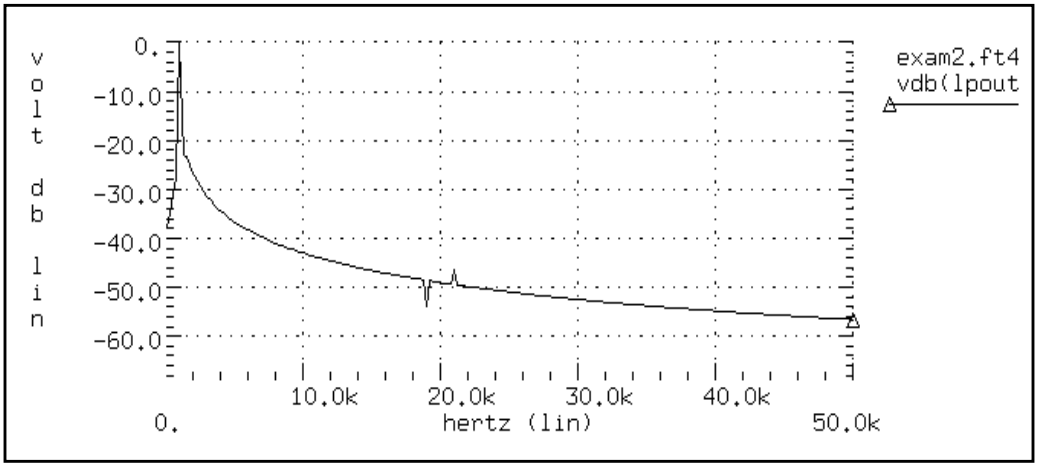
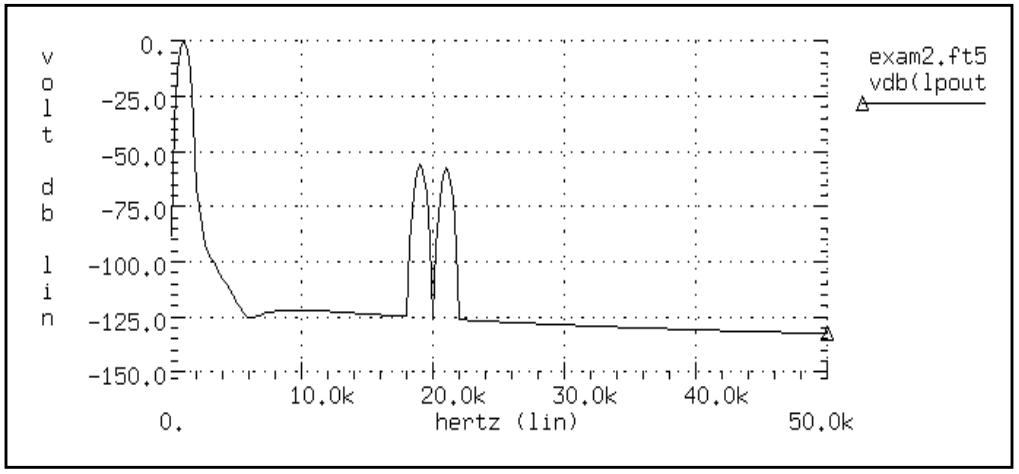


Figure 19-14: Filtered Output Signal (Blackman-Harris window)



Signal Detection Test Circuit

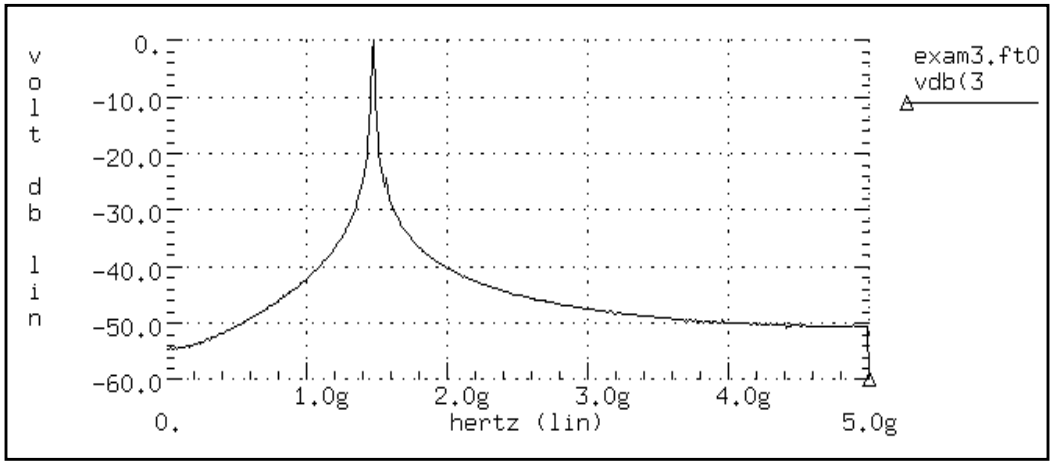
This example is a high frequency mixer test circuit, illustrating the effect of using a window to detect a weak signal in the presence of a strong signal at a nearby frequency. Two high frequency signals are added that have a 40 dB separation (that is, amplitudes are 1.0 and 0.01).

Input Listing

```
Signal Detection Test Circuit For FFT
v1 1 0 sin(0 1 1470.2Meg 0 0 90)
r1 1 0 1
v2 2 0 sin(0 0.01 1560.25Meg 0 0 90)
r2 2 0 1
E1 3 0 vol='v(1)+v(2)'
r3 3 0 1
.tran 0.1n 102.4n
.option post probe
.fft v(3)
.fft v(3) window=Bartlett fmin=1.2g fmax=2.2g
.fft v(3) window=hanning fmin=1.2g fmax=2.2g
.fft v(3) window=hamminn fmin=1.2g fmax=2.2g
.fft v(3) window=blackman fmin=1.2g fmax=2.2g
.fft v(3) window=harris fmin=1.2g fmax=2.2g
.fft v(3) window=gaussian fmin=1.2g fmax=2.2g
.fft v(3) window=kaiser fmin=1.2g fmax=2.2g
.end
```

For comparison with the rectangular window in Figure 19-15, the spectra of the output for all of the FFT window types are shown in Figure 19-16 through Figure 19-22. Without windowing, the weak signal is essentially undetectable due to spectral leakage.

Figure 19-15: Mixer Output Spectrum, Rectangular Window



In the Bartlett window in Figure 19-16, notice the dramatic decrease in the noise floor over 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 degree of separation of the two tones and the noise floor is best with the Blackman-Harris window. The final two windows (Figure 19-21 and Figure 19-22) are parameterized with ALFA=3.0, which is the default value in Star-Hspice. These two windows also produce acceptable results, especially the Kaiser-Bessel window, which gives sharp separation of the two tones and almost a -100-dB noise floor.

Such processing of high frequencies, as demonstrated in this example, shows the numerical stability and accuracy of the FFT spectrum analysis algorithms in Star-Hspice.

Figure 19-16: Mixer Output Spectrum, Bartlett Window

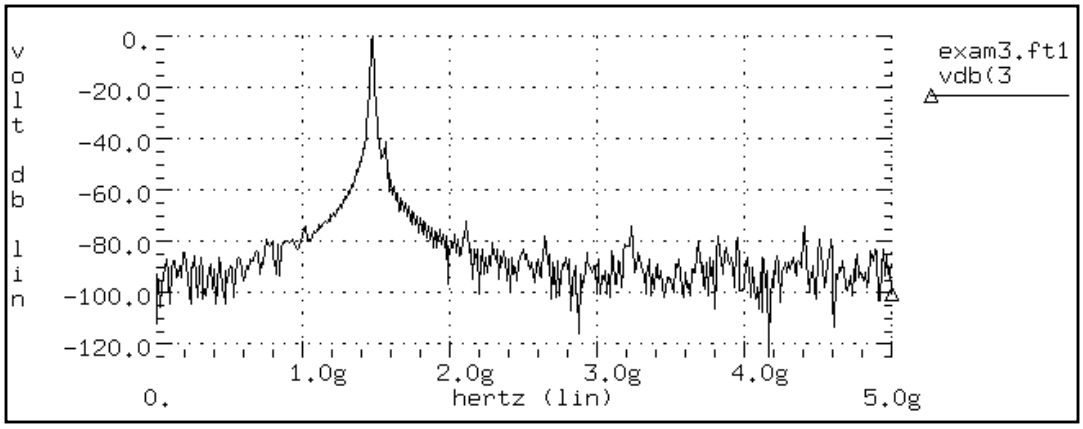


Figure 19-17: Mixer Output Spectrum, Hanning Window

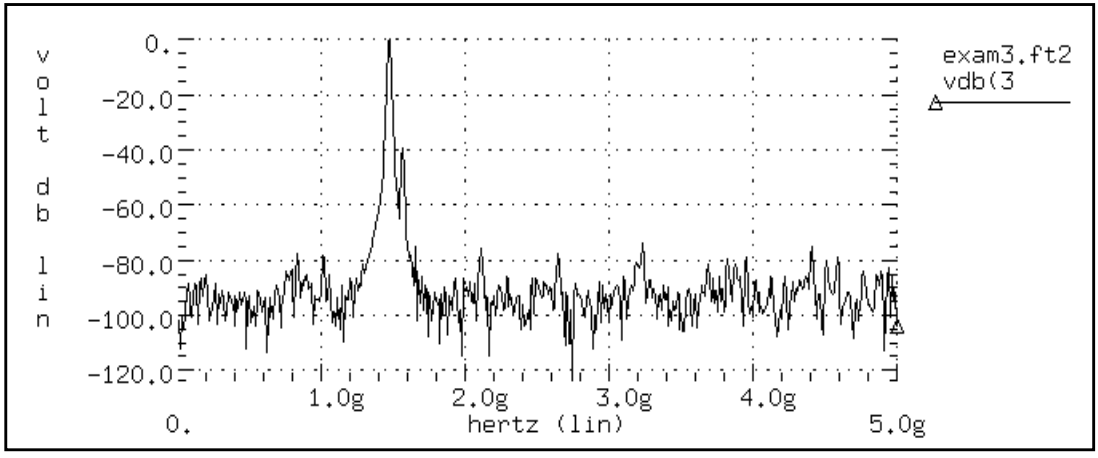


Figure 19-18: Mixer Output Spectrum, Hamming Window

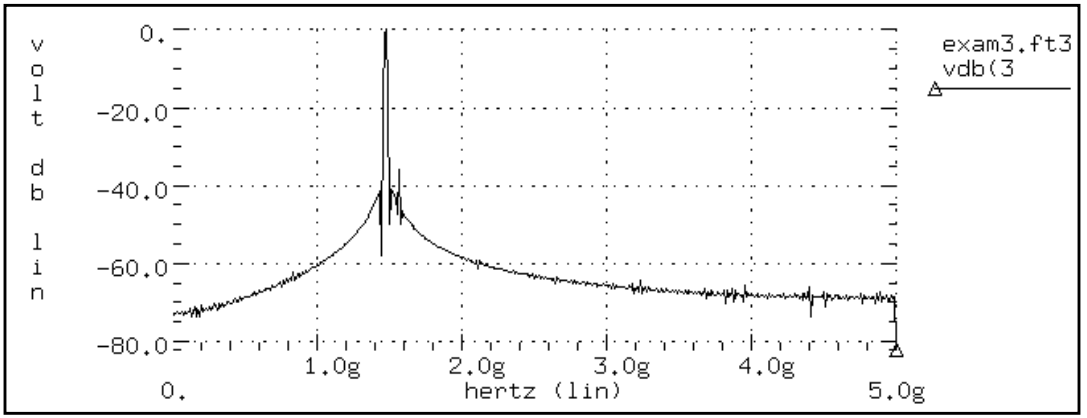


Figure 19-19: Mixer Output Spectrum, Blackman Window

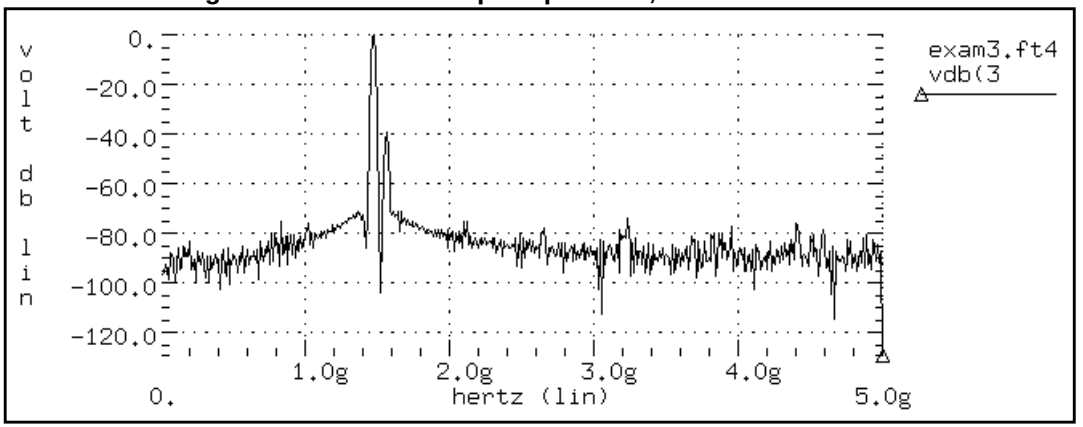


Figure 19-20: Mixer Output Spectrum, Blackman-Harris Window

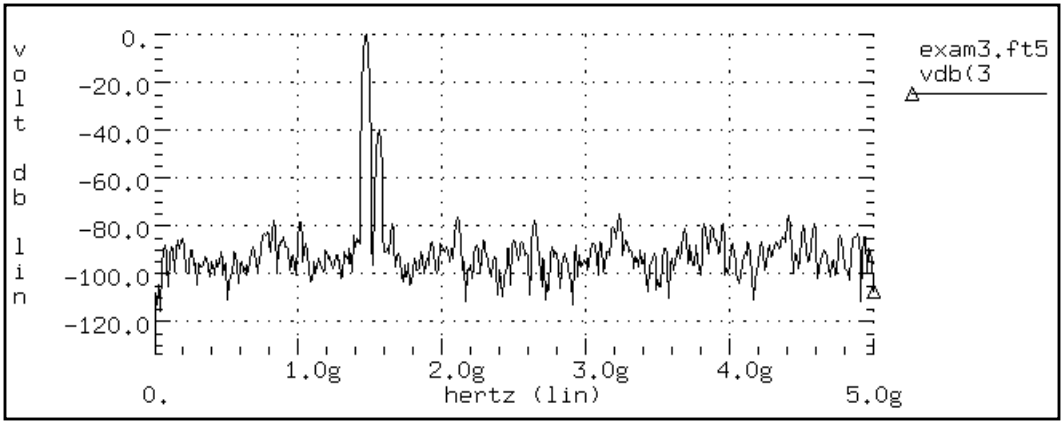


Figure 19-21: Mixer Output Spectrum, Gaussian Window

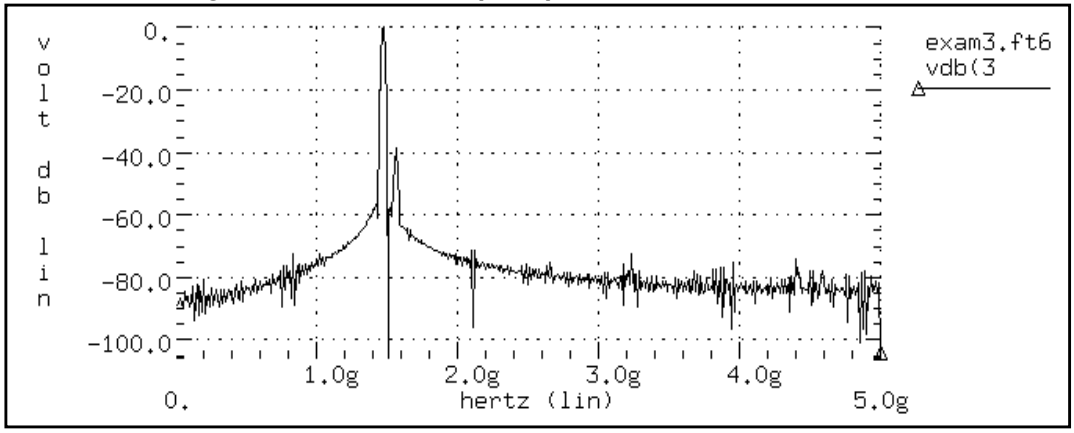
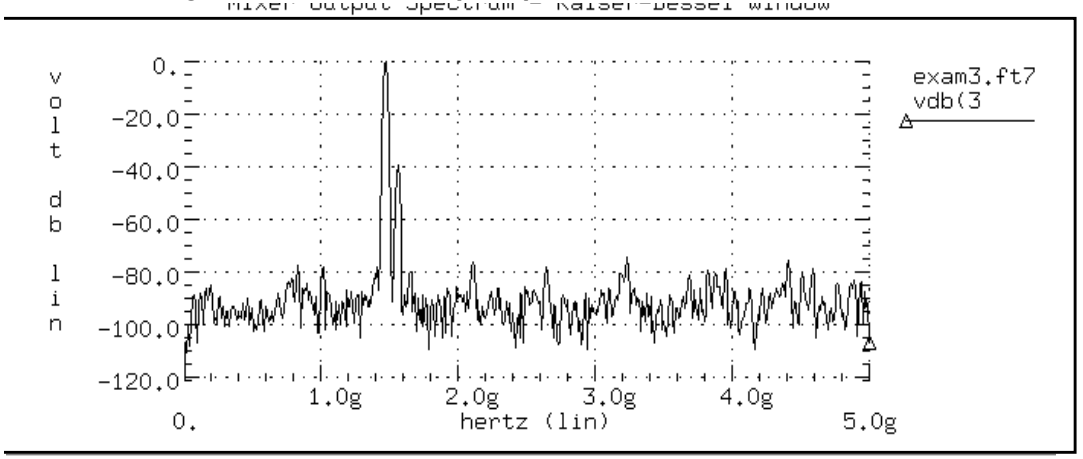


Figure 19-22: Mixer Output Spectrum, Kaiser-Bessel Window



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.

Avant!

Chapter 20
Modeling Filters and Networks

Applying Kirchhoff's laws to circuits containing energy storage elements results in simultaneous differential equations in the time domain that must be solved to analyze the circuit's behavior. The solution of any equation of higher than first order can be difficult, and some driving functions cannot be solved easily by classical methods.

In both cases, the solution might be simplified using Laplace transforms to convert time domain equations containing integral and differential terms to algebraic equations in the frequency domain.

This chapter covers the following topics:

Understanding Transient Modeling

The Laplace transform method also provides an easy way of relating a circuit's behavior in time and frequency-domains, facilitating simultaneous work in those domains.

The performance of the algorithm Star-Hspice uses for Laplace and pole/zero transient modeling is better than the performance of the Fast Fourier Transform (FFT) algorithm. Laplace and pole/zero transient modeling is invoked by using a LAPLACE or POLE function call in a source element statement.

Laplace transfer functions are especially useful in top-down system design, using ideal transfer functions instead of detailed circuit designs. Star-Hspice also allows you to mix Laplace transfer functions with transistors and passive components. Using this capability, a system may be modeled as the sum of the contributing ideal transfer functions, which can be progressively replaced by detailed circuit models as they become available. Laplace transfer functions are also conveniently used in control systems and behavioral models containing nonlinear elements.

Using Laplace transforms can reduce the long simulation times (as well as design time) of large interconnect systems, such as clock distribution networks, for which you can use methods such as asymptotic waveform evaluation (AWE) to create a Laplace transfer function model. The AWE model can represent the large circuit with just a few poles. 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.

Pole/zero analysis is important in determining the stability of the design. The POLE function in Star-Hspice is useful when the poles and zeros of the circuit are provided, or they can be derived from the transfer function. (You can use the Star-Hspice .PZ statement to find poles and zeros. See .PZ (Pole/Zero) Statement on page 18-3 for information about the .PZ statement).

Frequency response, an important analog circuit property, is normally specified as a ratio of two complex polynomials (functions of complex frequencies) with positive real coefficients. Frequency response can be given in the form of the locations of poles and zeros or can be in the form of a frequency table.

Complex circuits are usually designed by interconnecting smaller functional blocks of known frequency response, either in pole/zero or frequency table form. For example, you can design a band-reject filter by interconnecting a low-pass filter, a high-pass filter, and an adder. The designer should study the function of the complex circuit in terms of its component blocks before designing the actual circuit. After testing the functionality of the component blocks, they can be used as a reference in using optimization techniques to determine the complex element's value.

Using G and E Elements

This section describes how to use the G and E Elements.

Laplace Transform Function Call

Use the Star-Hspice G and E Elements (controlled behavioral sources) as linear functional blocks or elements with specific frequency responses in the following forms:

- Laplace Transform
- Pole-Zero Function
- Frequency Response Table

The frequency response is called the impulse response and is denoted by H(s), where s is a complex frequency variable ($s = j2\pi f$). In Star-Hspice, the frequency response is obtained by performing an AC analysis with AC=1 in the input source (the Laplace transform of an impulse is 1). The input and output of the G and E Elements with specified frequency response are related by the expression:

$$Y(j2\pi f) = Hj(2\pi f) \cdot Xj(2\pi f)$$

where X, Y and H are the input, the output, and the transfer function at frequency f.

For AC analysis, the frequency response is determined by the above relation at any frequency. 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

$$y(k\Delta) = \Delta \sum_{m=0}^{k} x(m\Delta) \cdot h[(k-m) \cdot \Delta], k = 0, 1, 2, ...$$

where the h(t) can be obtained from H(f) by the inverse Fourier integral:

$$h(t) = \int_{-\infty}^{\infty} H(f) \cdot e^{j2\pi ft} \cdot df$$

The inverse discrete Fourier transform is given by

$$h(m\Delta) = \frac{1}{N \cdot \Delta} \sum_{n=0}^{N-1} H(f_n) \cdot e^{\frac{j2\pi nm}{N}}, m = 0, 1, 2, ..., N\text{-}1$$

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, Star-Hspice's performance-enhanced algorithm is used to convert H(f) to h(t). This algorithm requires N to be a power of 2. The frequency point f_n is determined by

$$f_n = \frac{n}{N \cdot \Delta}, \quad n = 0, 1, 2, ..., N\text{-}1$$

where n > N/2 represents the negative frequencies. The Nyquist critical frequency is given by

$$f_c = f_{N/2} = \frac{1}{2 \cdot \Delta}$$

Since the negative frequencies responses are the image of the positive ones, only N/2 frequency points are required to evaluate N time points of h(t). The larger f_c is, the more accurate the transient analysis results are. However, for large f_c, 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, f_c can be set to the frequency at which the response drops by 60 dB (a factor of 1000).

$$|H(f_c)| = \frac{|H_{max}|}{1000}$$

Once f_c is selected or calculated, then Δ can be determined by

$$\Delta = \frac{1}{2 \cdot f_c}$$

Notice the frequency resolution

$$\Delta f = f_1 = \frac{1}{N \cdot \Delta}$$

is inversely proportional to the maximum time ($N \cdot \Delta$) over which h(t) is evaluated. 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 maximum frequency MAXF in the G or E Element statement. N is calculated by 2·MAXF/DELF. Then, N is modified to be a power of 2. The effective DELF is determined by 2·MAXF/N to reflect the changes in N.

Laplace Transform

The syntax of the LAPLACE function is:

Transconductance H(s):

```
Gxxx n+ n- LAPLACE in+ in-   k0, k1, ..., kn / d0,
+ d1, ..., dm
<SCALE=val> <TC1=val> <TC2=val> <M=val>
```

Voltage Gain H(s):

```
Exxx n+ n- LAPLACE in+ in-   k0, k1, ..., kn / d0,
+ d1, ..., dm
<SCALE=val> <TC1=val> <TC2=val>
```

H(s) is a rational function in the following form:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

All the coefficients k_0, k_1, ..., d_0, d_1, ..., can be parameterized.

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

The syntax is:

Transconductance H(s):

Gxxx n+ n- POLE in+ in- *a* α_{z1}, *f*$_{z1}$, ..., α_{zn}, *f*$_{zn}$ / *b*,
+ α_{p1}, *f*$_{p1}$, ..., α_{pm}, *f*$_{pm}$ <SCALE=val> <TC1=val>
+ <TC2=val> <M=val>

Voltage Gain H(s):

Exxx n_+ n_- POLE in_+ in_- *a* α_{z1}, *f*$_{z1}$, ..., α_{zn}, *f*$_{zn}$ / *b*,
+ α_{p1}, *f*$_{p1}$, ..., α_{pm}, *f*$_{pm}$ <SCALE=val> <TC1=val>
+ <TC2=val>

H(s) in terms of poles and zeros is defined by:

$$H(s) = \frac{a \cdot (s + \alpha_{z1} - j2\pi f_{z1})\ldots(s + \alpha_{zn} - j2\pi f_{zn})(s + \alpha_{zn} + j2\pi f_{zn})}{b \cdot (s + \alpha_{p1} - j2\pi f_{p1})\ldots(s + \alpha_{pm} - j2\pi f_{pm})(s + \alpha_{pm} + j2\pi f_{pm})}$$

Notice the complex poles or zeros are in conjugate pairs. In the element description, only one of them is specified, and the program includes the conjugate. The a, b, α, and f values can be parameterized.

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 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 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

The syntax is:

Transconductance H(s):

Gxxx n+ n- FREQ in+ in- f_1, a_1, ϕ_1, ..., f_i, a_i, ϕ_1

```
+ <DELF=val> <MAXF=val> <SCALE=val> <TC1=val>
+ <TC2=val> <M=val> <LEVEL=val>
+ <INTERPOLATION=val> <EXTRAPOLATION=val>
+ <ACCURACY=val>
```

Voltage Gain H(s):

Exxx n+ n- FREQ in+ in- f_1, a_1, ϕ_1, ..., f_i, a_i, ϕ_1

```
+ <DELF=val> <MAXF=val> <SCALE=val> <TC1=val>
+ <TC2=val>
```

Each f_i is a frequency point in hertz, a_i is the magnitude in dB, and ϕ_1 is the phase in degrees. At each frequency the network response, magnitude, and phase are calculated by interpolation. The magnitude (in dB) is interpolated logarithmically as a function of frequency. The phase (in degrees) is interpolated linearly as a function of frequency.

$$|H(j2\pi f)| = \left(\frac{a_i - a_k}{\log f_i - \log f_k}\right)(\log f - \log f_i) + a_i$$

$$\angle H(j2\pi f) = \left(\frac{\phi_i - \phi_k}{f_i - f_k}\right)(f - f_i) + \phi_i$$

Note: A new frequency table G element, with improved accuracy, was introduced in Star-Hspice 2001.2. You can choose how to interpolate and extrapolate this new element, as described in the next section, Element Statement Parameters.

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 is magnitude in dB, and third is phase in degrees. The LEVEL must be set to 1 for a high-pass filter, and the last frequency point must be the highest frequency response value that is a real number with zero phase. The frequency, magnitude, and phase in the table can be parameterized.

Element Statement Parameters

These keywords are common to the three forms, Laplace, pole-zero, and frequency response table described above.

ACCURACY Used only in G element with frequency response table.

0: default. The most accurate control voltage at each time point is achieved by linear interpolation of closest 2 time points

1: faster mode. The control voltage at each time point is determined from the simulated time point just before the target. The smaller the time step is set, the closer the result will be to the most accurate mode.

2: method used in release 2000.4 and prior.

DELF, DELTA Frequency resolution Δf. The inverse of DELF is the time window over which h(t) is calculated from H(s). The smaller DELF is, the more accurate the transient analysis, and the longer the CPU time. The number of points, N, used in the conversion of H(s) to h(t) is $N=2 \cdot MAXF/DELF$. Since N must be a power of 2, the DELF is adjusted. The default is 1/TSTOP. In G element with FREQ and ACCURACY = 0 or 1, circular convolution for periodic input will be done by limiting the period by setting DELF = 1/T:T < TSTOP.

EXTRAPOLATION Extrapolation scheme used only in G element with frequency response table.

0: default. Linear interpolation using the last two boundary points

1: the last boundary point is used

FREQ Keyword to indicate that the transfer function is described by a frequency response table. Do not use FREQ as a node name in a G or E Element.

INTERPOLATION Interpolation scheme used only in G element with frequency response table.

0: default. piece wise linear
1: piece wise step
2: b-spline curve fit

LAPLACE Keyword to indicate the transfer function is described by a Laplace transform function. Do not use LAPLACE as a node name on a G or E Element.

LEVEL	Used only in elements with frequency response table. This parameter must be set to 1 if the element represents a high-pass filter.
M	G Element multiplier. This parameter is used to represent *M* G Elements in parallel. Default is 1.
MAXF, MAX	Maximum or the Nyquist critical frequency. The larger the MAXF the more accurate the transient results and the longer is the CPU time. The default is $1024 \cdot DELF$. Thee parameters are applicable only when the FREQ parameter is also used.
POLE	Keyword to indicate the transfer function is described by the pole and zero location. Do not use POLE as a node name on a G or E Element.
SCALE	Element value multiplier
TC1,TC2	First- and second-order temperature coefficients. The default is zero. The SCALE is updated by temperature: $SCALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$

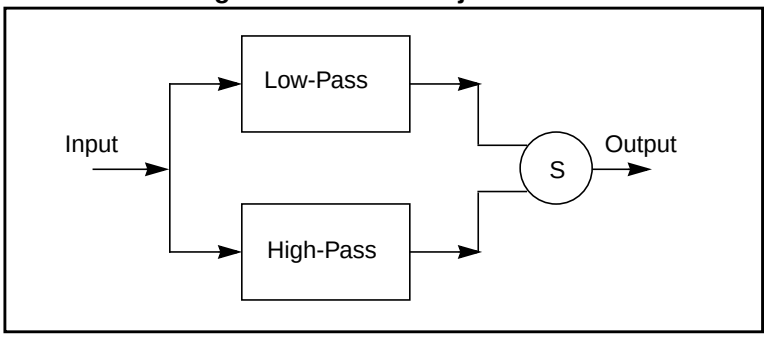
Notes

- Pole/zero analysis is not allowed when the data file contains elements with frequency response specifications. If you include a MAXF=<par> specification in a G or Element Statement, Star-Hspice issues a warning that MAXF is ignored. This is normal.
- Interpolation, Extrapolation, and Accuracy options are only for G (with FREQ) element
- The interpolation and extrapolation parameters noted above are available if ACCURACY = 0 or 1. If ACCURACY = 2, then interpolation/ extrapolation is performed as mentioned earlier under Frequency Response Table on page 20-8.
- Circular convolution is performed when G element ACCURACY = 0 or 1 (see Figure 20-7)

Laplace Band-Reject Filter

This example models an active band-reject filter (see 1 on page 20-54) 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 20-1: Band-Reject Filter



Example

```
BandstopL.sp band_reject filter
.OPTIONS PROBE POST=2
.AC DEC 50 10 5k
.PROBE AC VM(out_low) VM(out_high) VM(out)
.PROBE AC VP(out_low) VP(out_high) VP(out)
.TRAN .01m 12m
.PROBE V(out_low) V(out_high) V(out)
.GRAPH v(in) V(out)
Vin in 0 AC 1 SIN(0,1,250)
```

Band_Reject Filter Circuit

```
Elp3 out_low3 0 LAPLACE in 0
+ 1 / 1 6.729m 15.62988u 27.7976n
Elp out_low   0 LAPLACE out_low3 0
+ 1 / 1 0.364m 2.7482u
Ehp3 out_high3 0 LAPLACE in 0  0,0,0,9.261282467p /
+ 1,356.608u,98.33419352n,9.261282467p
Eph out_high 0 LAPLACE out_high3 0
+ 0 0 144.03675n / 1 83.58u 144.03675n
Eadd   out 0 VOL='-V(out_low) - V(out_high)'
Rl out 0 1e6
.END
```

Figure 20-2: Frequency Response of the Band-Reject Filter

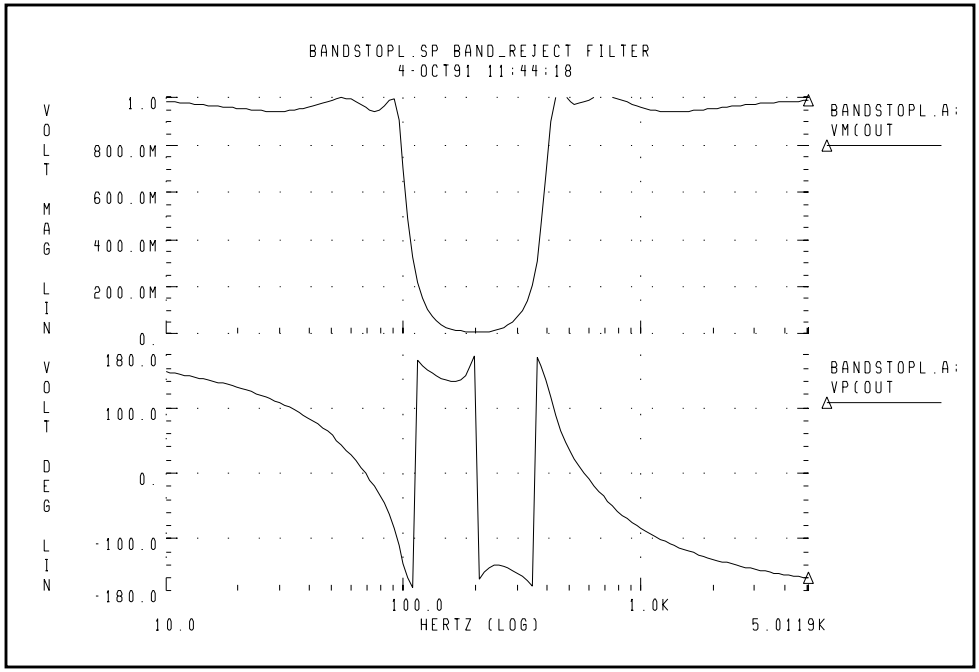
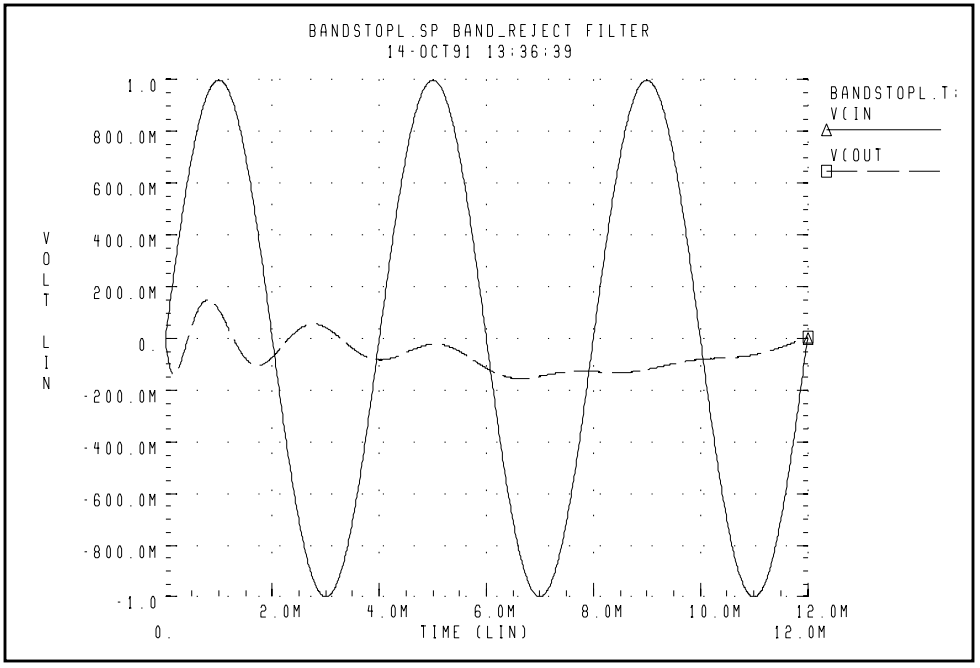


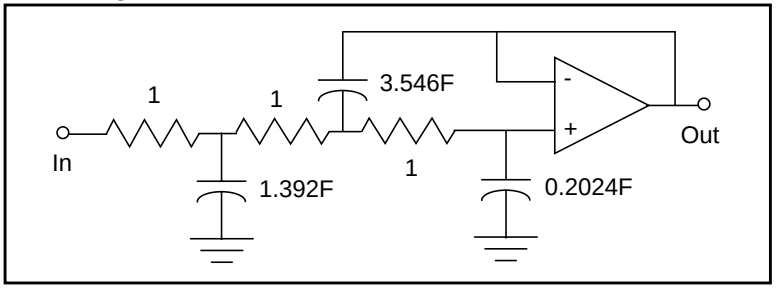
Figure 20-3: Transient Response of the Band-Reject Filter to a 250 Hz Sine Wave



Laplace Low-Pass Filter

This example simulates a third-order low-pass filter with a Butterworth transfer function, comparing the results of the actual circuit and the functional G Element with third-order Butterworth transfer function for AC and transient analysis.

Figure 20-4: 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 is the input listing of the above filter. Notice the pole locations are parameterized on the G Element. Also, only one of the complex poles is specified. The conjugate pole is derived by the program. The output of the circuit is node “out” and the output of the functional element is “outg”.

Example

This is an example of a third-order low-pass Butterworth filter:

```
Low_Pass.sp 3rd order low-pass Butterworth
.OPTIONS POST=2 PROBE INTERP=1 DCSTEP=1e8
.PARAM a=1.0 b=1.0 ap1=1.0 fp1=0.0 ap2=0.5 fp2=0.1379
.AC DEC 25 0.01 10
.PROBE AC VDB(out) VDB(outg) VP(out) VP(outg)
.TRAN .5 200
.PROBE V(in) V(outg) V(out)
.GRAPH V(outg) V(out)
VIN in 0 AC 1   PULSE(0,1,0,1,1,48,100)
* 3rd order low-pass described by G Element
Glow_pass 0 outg POLE in 0 a / b ap1,fp1 ap2,fp2
Rg outg 0 1
```

Circuit Description

```
R1 in 2 1
R2 2 3 1
R3 3 4 1
C1 2 0 1.392
C2 4 0 0.2024
C3 3 out 3.546
Eopamp out 0   OPAMP 4 out
.END
```

Figure 20-5: Frequency Response of Circuit and Functional Element

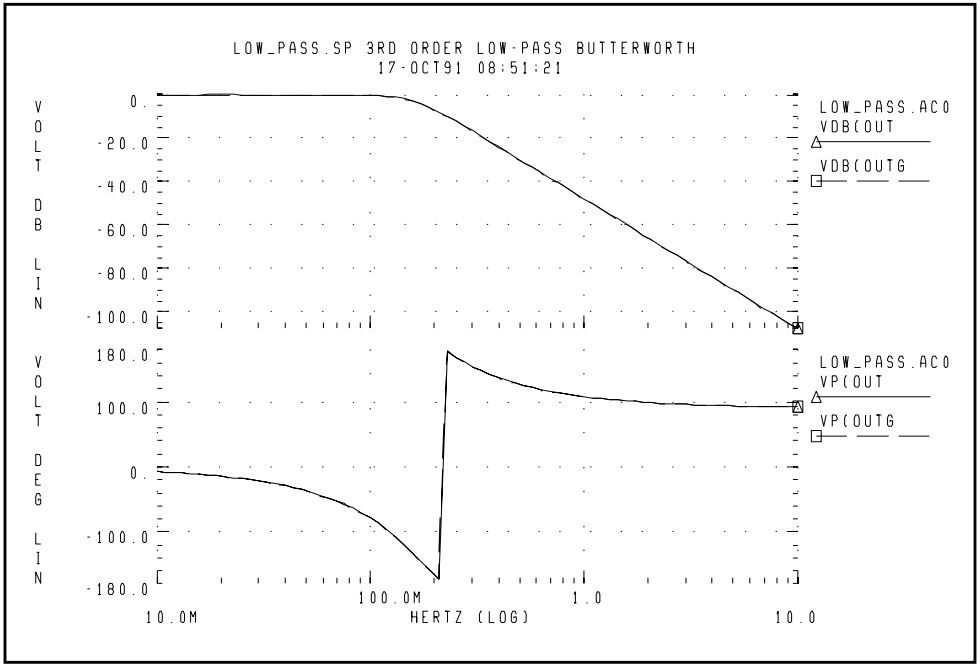
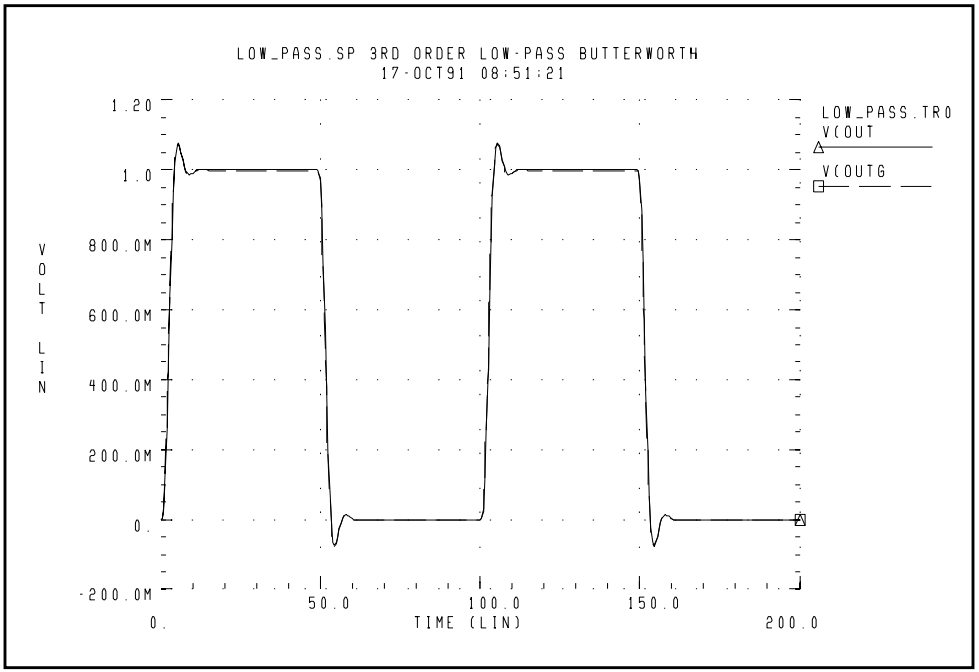


Figure 20-6: Transient Response of Circuit and Functional Element to a Pulse



Circular Convolution Example

This example simulates a 30 degree phase shift filter using circular convolution. By setting DELF = 10MHz, the period of time domain response of the G element which will be obtained by IFFT based on input frequency table will be set to 100n seconds. FREQ G element performs convolution integral from t - T to t assuming that all the control voltage at t<0 is zero. Here, t is the target time point and T is the period of time domain response of the G element.

In this example, during time point is from 0 to 100n seconds, higher than 10MHz harmonics components due to input transition at t=0 is taken. So the circuit does not behave as a phase shift filter. After one period (t>100n seconds), circular convolution based on 100n seconds period will be performed and the transient result represents 30 degree phase shift for continuous periodic control voltage.

Notes

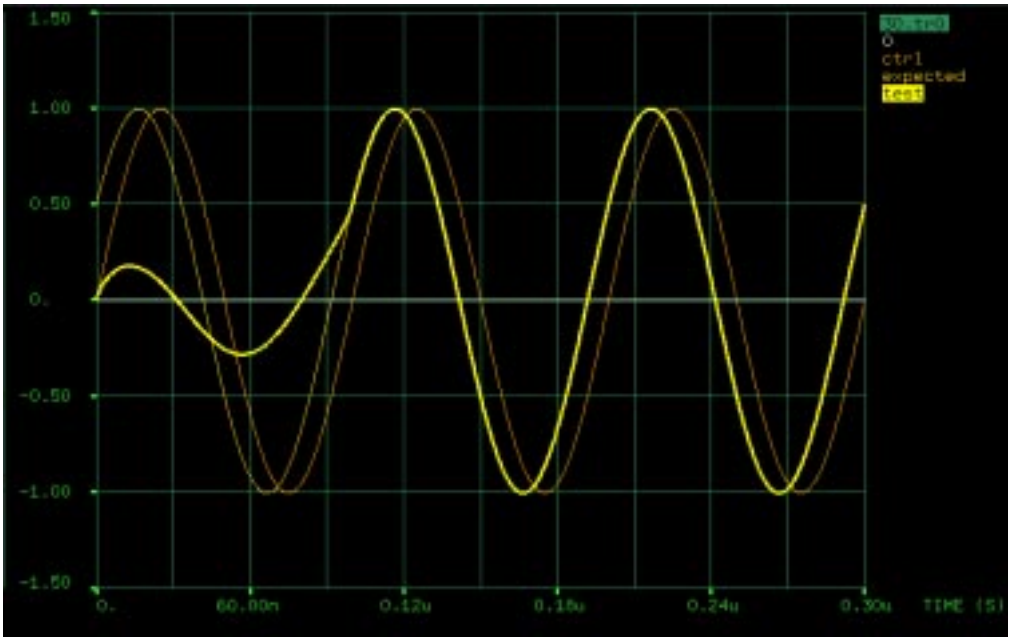
- V(ctrl): control voltage input
- V(expected): node which represents ideal 30 degree shifted wave of input
- V(test): output of the G element

30 Degree Phase Shift Circuit File

This example illustrates a 30 degree phase shift filter.

```
.tran 0.1n 300n
.options post=2 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 2 3 expected 0v
Rexpected expected gnd 1
.end
```

Figure 20-7: Transient Response of the 30 Degree Phase Shift Filter



Laplace and Pole-Zero Modeling

Laplace Transform (LAPLACE) Function

There are two forms of the Star-Hspice LAPLACE function call, one for transconductance and one for voltage gain transfer functions. See Using G and E Elements on page 20-4 for the general forms and Element Statement Parameters on page 20-9 for descriptions of the parameters.

General Form of the Transfer Function

To use the Star-Hspice 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 of a single-source circuit to the input, with initial conditions set to zero. The Laplace transfer function is represented by:

$$H(s) = \frac{Y(s)}{X(s)},$$

where *s* is the complex frequency $j2\pi f$, *Y(s)* is the Laplace transform of the output signal, and *X(s)* is the Laplace transform of the input signal.

Note: In Star-Hspice, the impulse response H(s) is obtained by performing an AC analysis, with AC=1 representing 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, Star-Hspice uses the value of the EPSMIN option (the smallest number possible on the platform) for the transfer function in its calculations.

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 of the transfer function cannot be greater than the order of the denominator, except for differentiators, for which the transfer function *H(s) = ks*. All of the transfer function's *k* and *d* coefficients can be parameterized in the Star-Hspice 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 by transforming each element's value into its *s*-domain equivalent form.

Table 20-1 and Table 20-2 show transforms used to convert some common functions to the *s*-domain (see 2 on page 20-54). The next section provides examples of using transforms to determine transfer functions.

Table 20-1: Laplace Transforms for Common Source Functions

f(t), t>0	Source Type	L {f(t)}= F(s)
$\delta(t)$	impulse	1
$u(t)$	step	$\frac{1}{s}$
t	ramp	$\frac{1}{s^2}$
e^{-at}	exponential	$\frac{1}{s+a}$
$\sin\ \omega t$	sine	$\frac{\omega}{s^2+\omega^2}$
$\cos\ \omega t$	cosine	$\frac{s}{s^2+\omega^2}$
$\sin(\omega t+\theta)$	sine	$\frac{s\sin(\theta)+\omega\cos(\theta)}{s^2+\omega^2}$

Table 20-1: Laplace Transforms for Common Source Functions *(Continued)*

f(t), t>0	Source Type	L {f(t)}= F(s)
$\cos(\omega t + \theta)$	cosine	$\frac{s\cos(\theta) - \omega\sin(\theta)}{s^2 + \omega^2}$
$\sinh \omega t$	hyperbolic sine	$\frac{\omega}{s^2 - \omega^2}$
$\cosh \omega t$	hyperbolic cosine	$\frac{s}{s^2 - \omega^2}$
te^{-at}	damped ramp	$\frac{1}{(s + a)^2}$
$e^{-at} \sin \omega t$	damped sine	$\frac{\omega}{(s + a)^2 + \omega^2}$
$e^{-at} \cos \omega t$	damped cosine	$\frac{s + a}{(s + a)^2 + \omega^2}$

Table 20-2: Laplace Transforms for Common Operations

f(t)	L {f(t)} = F(s)
$Kf(t)$	$KF(s)$
$f_1(t) + f_2(t) - f_3(t) + \ldots$	$F_1(s) + F_2(s) - F_3(s) + \ldots$
$\frac{d}{dt}f(t)$	$sF(s) - f(0-)$
$\frac{d^2}{dt^2}f(t)$	$s^2F(s) - sf(0) - \frac{d}{dt}f(0-)$

Table 20-2: Laplace Transforms for Common Operations *(Continued)*

f(t)	L {f(t)} = F(s)
$\frac{d^n}{dt^n}f(t)$	$s^nF(s) - s^{n-1}f(0) - s^{n-2}\frac{d}{dt}f(0)$
$\int_{-\infty}^{t} f(t)dt$	$\frac{F(s)}{s} + \frac{f^{-1}(0)}{s}$
$f(t-a)u(t-a), a > 0$ (u is the step function)	$e^{-as}F(s)$
$e^{-at}f(t)$	$F(s+a)$
$f(at), a > 0$	$\frac{1}{a}F\left(\frac{s}{a}\right)$
$tf(t)$	$-\frac{d}{ds}(F(s))$
$t^nf(t)$	$-(1)^n\frac{d^n}{ds^n}F(s)$
$\frac{f(t)}{t}$	$\int_s^{\infty} F(u)du$ (u is the step function)
$f(t-t_1)$	$e^{-t_1s}F(s)$

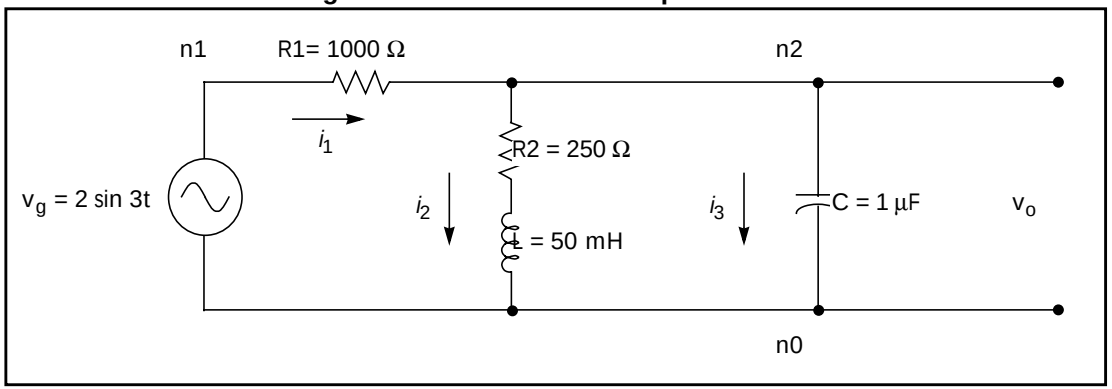
Determining the Laplace Coefficients

The following examples describe how to determine the appropriate coefficients for the Laplace modeling function call in Star-Hspice.

LAPLACE Example 1 – Voltage Gain Transfer Function

To find the voltage gain transfer function for the circuit in Figure 20-8, convert the circuit to its equivalent *s*-domain circuit and solve for v_o / v_g.

Figure 20-8: LAPLACE Example 1 Circuit



Use transforms from Table 20-2 to convert the inductor, capacitor, and resistors. $L\{f(t)\}$ represents the Laplace transform of $f(t)$:

$$L\left\{L\frac{d}{dt}f(t)\right\} = L\cdot(sF(s)-f(0)) = 50\times10^{-3}\cdot(s-0) = 0.05s$$

$$L\left\{\frac{1}{C}\int_0^t f(t)d\tau\right\} = \frac{1}{C}\cdot\left(\frac{F(s)}{s}+\frac{f^{-1}(0)}{s}\right) = \frac{1}{10^{-6}}\cdot\left(\frac{1}{s}+0\right) = \frac{10^6}{s}$$

$$L\{R1\cdot f(t)\} = R1\cdot F(s) = R1 = 1000\ \Omega$$

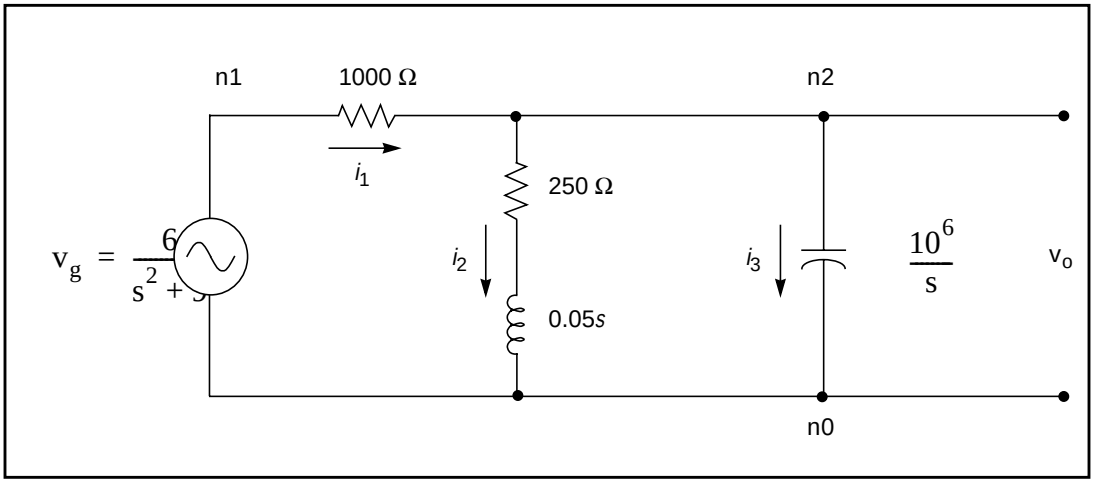
$$L\{R2\cdot f(t)\} = R2\cdot F(s) = R2 = 250\ \Omega$$

To convert the voltage source to the *s*-domain, use the sin ωt transform from Table 20-1:

$$L\{2\sin 3t\} = 2\cdot\frac{3}{s^2+3^2} = \frac{6}{s^2+9}$$

Figure 20-9 displays the *s*-domain equivalent circuit.

Figure 20-9: S-Domain Equivalent of the LAPLACE Example 1 Circuit



Summing the currents leaving node n2:

$$\frac{v_o - v_g}{1000} + \frac{v_o}{250 + 0.05s} + \frac{v_o s}{10^6} = 0$$

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 Star-Hspice Laplace function call, use 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 voltage gain transfer function above are:

$k_0 = 5x10^6$ $k_1 = 1000$

$d_0 = 25x10^6$ $d_1 = 6000$ $d_2 = 1$

Using these coefficients, a Star-Hspice Laplace modeling function call for the voltage gain transfer function of the circuit in Figure 20-8 is:

```
Eexample1 n1 n0 LAPLACE n2 n0 5E6 1000 / 25E6 6000 1
```

LAPLACE Example 2 – Differentiator

You can model a differentiator using either G or E Elements as shown in the following example.

In the frequency domain:

$$\text{E Element: } V_{out} = ksV_{in}$$

$$\text{G Element: } I_{out} = ksV_{in}$$

In the time domain:

$$\text{E Element: } v_{out} = k\frac{dV_{in}}{dt}$$

$$\text{G Element: } i_{out} = k\frac{dV_{in}}{dt}$$

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 coefficients $k_1 = k$ and $d_0 = 1$ in the Laplace modeling, the Star-Hspice 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

An integrator can be modeled by G or E Elements as follows:

In the frequency domain:

$$\text{E Element: } V_{out} = \frac{k}{s} V_{in}$$

$$\text{G Element: } I_{out} = \frac{k}{s} V_{in}$$

In the time domain:

$$\text{E Element: } v_{out} = k \int V_{in} dt$$

$$\text{G Element: } i_{out} = k \int V_{in} dt$$

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:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

Like the previous example, if you make $k_0 = k$ 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 and provides examples of converting specific transfer functions into pole/zero circuit descriptions.

POLE Function Call

The POLE function in Star-Hspice is useful when the poles and zeros of the circuit are available. The poles and zeros can be derived from the transfer function, as described in this chapter, or you can use the Star-Hspice .PZ statement to find them, as described in .PZ (Pole/Zero) Statement on page 18-3.

There are two forms of the Star-Hspice LAPLACE function call, one for transconductance and one for voltage gain transfer functions. See Using G and E Elements for the general forms and list of 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 of 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:

$$H(s) = \frac{a(s + \alpha_{z1} + j2\pi f_{z1})(s + \alpha_{z1} - j2\pi f_{z1})\ldots(s + \alpha_{zn} + j2\pi f_{zn})(s + \alpha_{zn} - j2\pi f_{zn})}{b(s + \alpha_{p1} + j2\pi f_{p1})(s + \alpha_{p1} - j2\pi f_{p1})\ldots(s + \alpha_{pm} + j2\pi f_{pm})(s + \alpha_{pm} - j2\pi f_{pm})} \quad (1)$$

The a, b, α, and f values can be parameterized.

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 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 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 Star-Hspice pole/zero circuit description for an element, you need to know the element's transfer function *H(s)* in terms of the *a*, *b*, *f*, and α coefficients. Use the values of these coefficients in POLE function calls in the Star-Hspice circuit description.

First, however, simplify the transfer function, as described in the next section.

Star-Hspice 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):

$$(s + \alpha_{pm} + j2\pi f_{pm})(s + \alpha_{pm} - j2\pi f_{pm}), \text{ for poles}$$

and

$$(s + \alpha_{zn} + j2\pi f_{zn})(s + \alpha_{zn} - j2\pi f_{zn}), \text{ for zeros}$$

To write the transfer function in Star-Hspice pole/zero format, supply coefficients for one term of each conjugate pair and Star-Hspice 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*)}.

Find the reduced form by collecting all the general form terms with negative complex roots:

$$H(s) = \frac{a(s+\alpha_{z1}+j2\pi f_{z1})\ldots(s+\alpha_{zn}+j2\pi f_{zn})}{b(s+\alpha_{p1}+j2\pi f_{p1})\ldots(s+\alpha_{pm}+j2\pi f_{pm})} \cdot \frac{a(s+\alpha_{z1}-j2\pi f_{z1})\ldots(s+\alpha_{zn}-j2\pi f_{zn})}{b(s+\alpha_{p1}-j2\pi f_{p1})\ldots(s+\alpha_{pm}-j2\pi f_{pm})}$$

(1)

Then discard the right-hand term, which contains all the terms with negative roots. What remains is the reduced form:

$$\text{Reduced}\{H(s)\} = \frac{a(s+\alpha_{z1}+j2\pi f_{z1})\ldots(s+\alpha_{zn}+j2\pi f_{zn})}{b(s+\alpha_{p1}+j2\pi f_{p1})\ldots(s+\alpha_{pm}+j2\pi f_{pm})} \quad (2)$$

For this function find the *a*, *b*, *f*, and α coefficients to use in a Star-Hspice 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 given 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 (1) shown previously, so that you can see the conjugate pairs of complex roots. You need to supply only one of each conjugate pair of roots in the Laplace function call. Star-Hspice automatically inserts the other root.

To get the function into a form more similar to the general form of the transfer function, rewrite the given transconductance transfer function as:

$$H(s) = \frac{1.0(s+0.0)}{1.0(s+0.001)}$$

Since this function has no negative imaginary parts, it is already in the Star-Hspice reduced form (2) shown previously. Now you can identify the *a*, *b*, *f*, and α coefficients so that the transfer function *H*(*s*) matches the reduced form.

This matching process obtains the following values:

$$n = 1, m = 1,$$

$$a = 1.0 \quad \alpha_{z1} = 0.0 \quad f_{z1} = 0.0$$

$$b = 1.0 \quad \alpha_{p1} = 0.001 \quad f_{p1} = 0.0$$

Using these coefficients in the reduced form provides the desired transfer function,

$$\frac{s}{(s + 0.001)} .$$

So the general transconductance transfer function POLE function call,

```
Gxxx n+ n- POLE in+ in- a αz1,fz1...αzn,fzn /
b αp1,fp1...αpm,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 given 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 given voltage gain transfer function as:

$$H(s) = \frac{1.0}{1.0(s + 1.0 + j2\pi \cdot 0.0)(s + 0.5 + j2\pi \cdot 0.15)} \cdot \frac{1.0}{(s + 0.5 - j2\pi \cdot 0.15)}$$

$$= \text{Reduced}\{H(s)\} \cdot \frac{1.0}{(s + 0.5 - j2\pi \cdot 0.15)}$$

So:

$$\text{Reduced}\{H(s)\} = \frac{1.0}{1.0(s + 1.0)(s + 0.5 + j2\pi \cdot 0.15)}$$

or:

$$\frac{a(s + \alpha_{z1} + j2\pi f_{z1})\ldots(s + \alpha_{zn} + j2\pi f_{zn})}{b(s + \alpha_{p1} + j2\pi f_{p1})\ldots(s + \alpha_{pm} + j2\pi f_{pm})} = \frac{1.0}{1.0(s + 1.0 + j2\pi \cdot 0.0)(s + 0.5 + j2\pi \cdot 0.15)}$$

Now assign coefficients in the reduced form to match the given voltage transfer function. The following coefficient values produce the desired transfer function:

$$n = 0,\ m = 2,$$

$$a = 1.0 \quad b = 1.0 \quad \alpha_{p1} = 1.0 \quad f_{p1} = 0 \quad \alpha_{p2} = 0.5 \quad f_{p2} = 0.15$$

These coefficients can be substituted in the POLE function call for a voltage gain transfer function:

```
Exxx n+ n- POLE in+ in- a αz1,fz1...αzn,fzn /
b αp1,fp1...αpm,fpm
```

for an element named *Elow_pass* to obtain the Star-Hspice statement:

```
Elow_pass out GND POLE in 1.0 / 1.0 1.0,0.0 0.5,0.15
```

RC Line Modeling

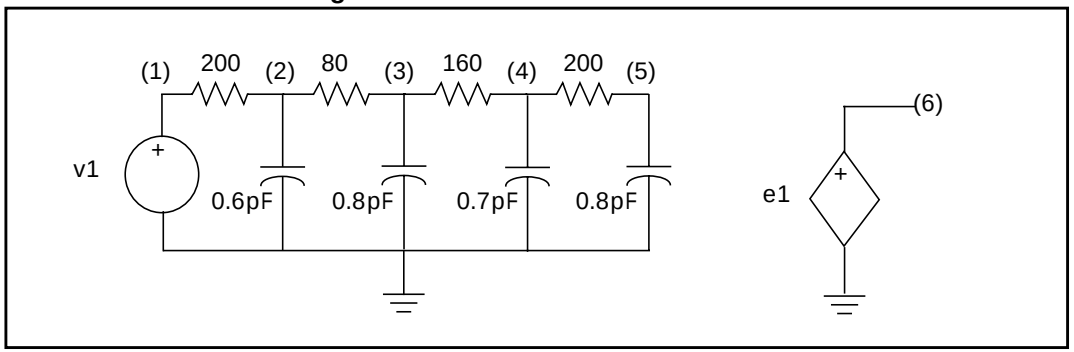
Most RC lines can have very simple models, with just a single dominant pole. The dominant pole can be found by AWE methods, computed based on the total series resistance and capacitance (see 4 on page 20-54), or determined by the Elmore delay (see 5 on page 20-54).

The Elmore delay uses the value (d1-k1) as the time constant of a single-pole approximation to the complete H(*s*), where H(*s*) is the transfer function of the RC network to a given 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 20-10: Circuits for an RC Line

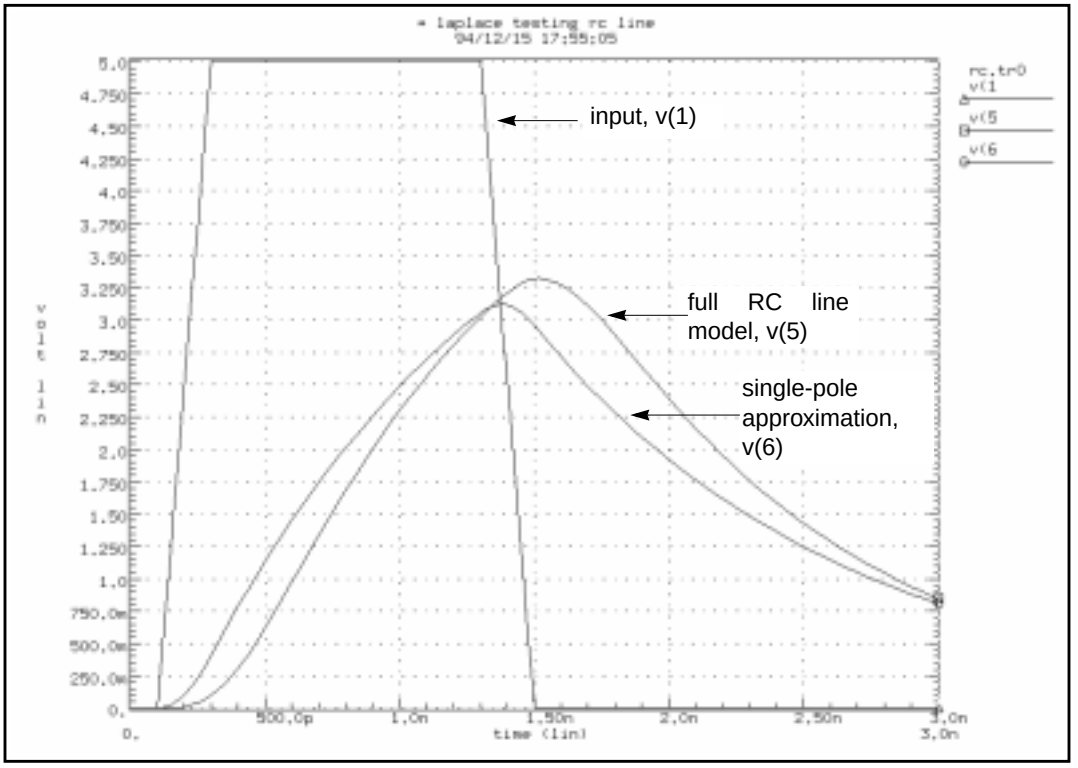


RC Line Circuit File

```
* Laplace testing RC line
.Tran 0.02ns 3ns
.Options 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
```

The output of the RC circuit shown in Figure 20-10 can be closely approximated by a single pole response, as shown in Figure 20-11.

Figure 20-11: Transient Response of the RC Line and Single-Pole Approximation

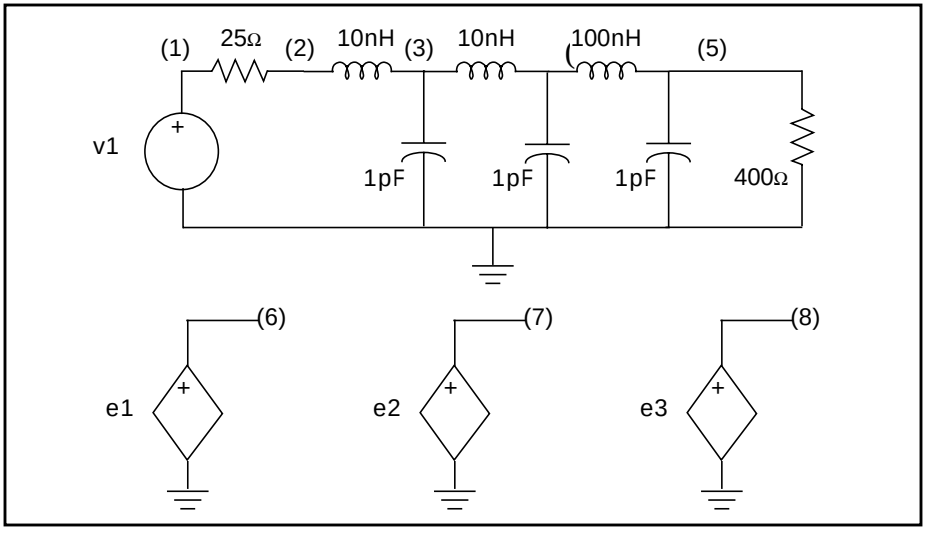


In Figure 20-11, the single pole approximation has less delay: 1 ns compared to 1.1 ns for the full RC line model at 2.5 volts. The single pole approximation also has a lower peak value than the RC line model. All other things being equal, a circuit with a shorter time constant results in less filtering 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 gives very good results for many problems.

AWE Transfer Function Modeling

Single-pole transfer function approximations can cause larger errors for low-loss lines than for RC lines since lower resistance allows ringing. Because circuit ringing creates complex pole pairs in the transfer function approximation, at least one complex pole pair is needed to represent low-loss line response. Figure 20-12 shows a typical low-loss line, along with the transfer function sources used to test the various approximations. The transfer functions were obtained by asymptotic waveform evaluation (see 6 on page 20-54).

Figure 20-12: Circuits for a Low-Loss Line



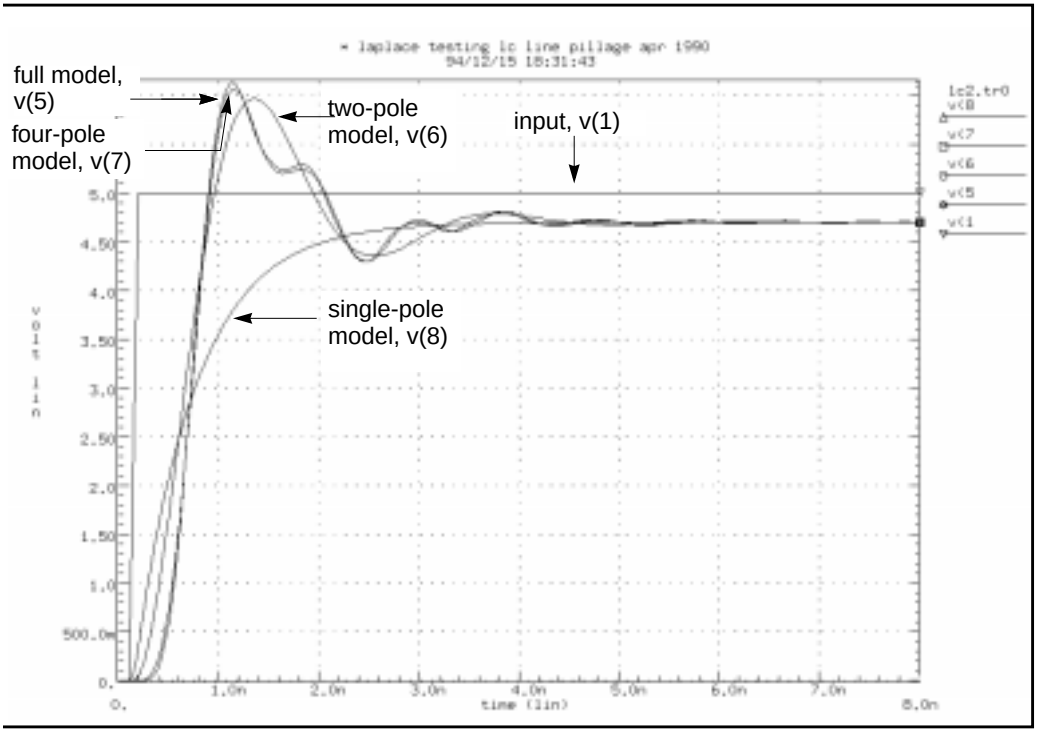
Low-Loss Line Circuit File

```
* Laplace testing LC line Pillage Apr 1990
.Tran 0.02ns 8ns
.Options Post Accurate List Probe
v1 1 0 PWL 0ns 0 0.1ns 0 0.2ns 5
r1 1 2 25
L1 2 3 10nH
c2 3 0 1pF
L2 3 4 10nH
c3 4 0 1pF
L3 4 5 100nH
c4 5 0 1pF

r4 5 0 400
e3 8 0 LAPLACE 1 0 0.94 / 1.0 0.6n

e2 7 0 LAPLACE 1 0 0.94e20 / 1.0e20 0.348e11 14.8
+ 1.06e-9 2.53e-19 SCALE=1.0e-20

e1 6 0 LAPLACE 1 0 0.94 / 1 0.2717e-9 0.12486e-18

.Probe v(1) v(5) v(6) v(7) v(8)
.Print v(1) v(5) v(6) v(7) v(8)
.End
```

Figure 20-13 shows the transient response of the low-loss line, along with E Element Laplace models using one, two, and four poles (see 6 on page 20-54). Note that the single-pole model shows none of the ringing of the higher order models. Also, all of the E models had to 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 takes care of the 25 ohm source and 400 ohm load voltage divider. All of the approximations give good delay estimations.

While the two-pole approximation gives reasonable agreement with the transient overshoot, the four-pole model gives almost perfect agreement. The actual circuit has six poles. Scaling was used to bring some of the very small numbers in the Laplace model above the 1e-28 limit of Star-Hspice. The SCALE parameter multiplies every parameter in the LAPLACE specification by the same value, in this case 1.0E-20.

Figure 20-13: Transient Response of the Low-Loss Line

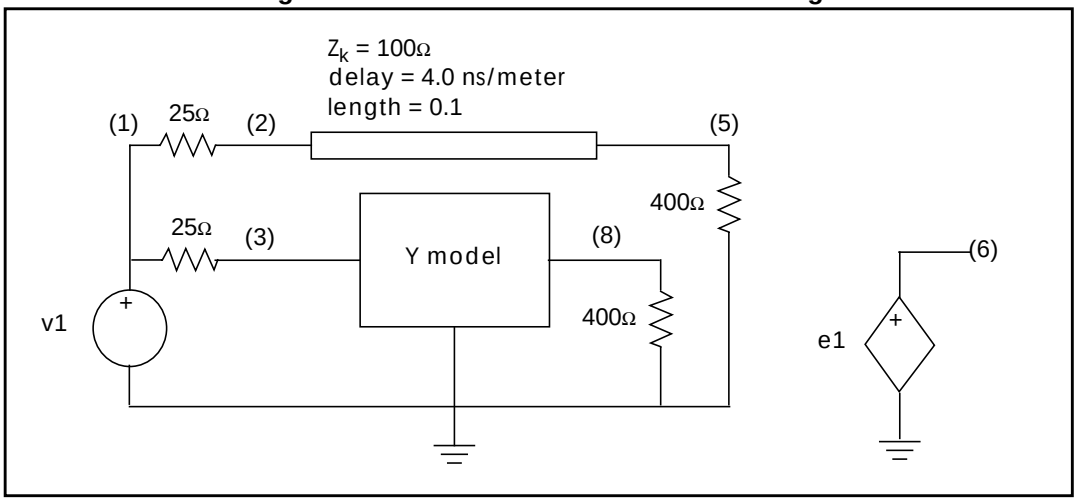


A low-loss line allows reflections between the load and source, while the loss of an RC line usually isolates the source from the load. So you can either incorporate the load into the AWE transfer function approximation or create a Star-Hspice model that allows source/load interaction. If you allow source/load interaction, the AWE expansions do not have to be done each time you change load impedances, allowing you to handle nonlinear loads and remove 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 allows the source/load interaction needed. The next example shows such a Y parameter model for a low-loss line.

Y Parameter Line Modeling

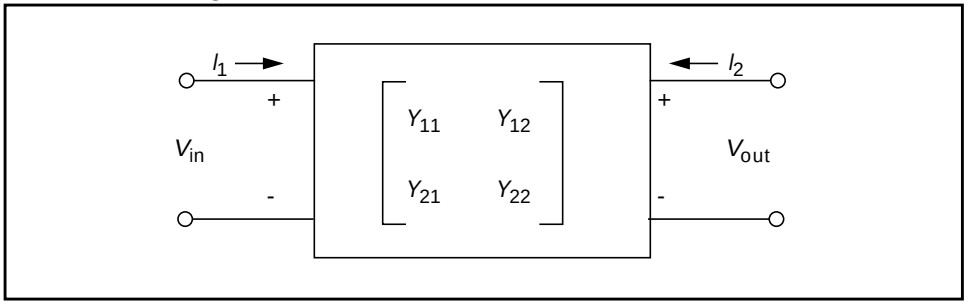
A model that is independent of load impedance is more complicated. You can still use AWE techniques, but you need a way for the load voltage and current must be able to interact with the source impedance. Given a transmission line of 100 ohms and 0.4 ns total delay, as shown in Figure 20-14, compare the response of the line using a Y parameter model and a single-pole model.

Figure 20-14: Line and Y Parameter Modeling



The voltage and current definitions for a Y parameter model are shown in Figure 20-15.

Figure 20-15: Y Matrix for the Two-Port Network



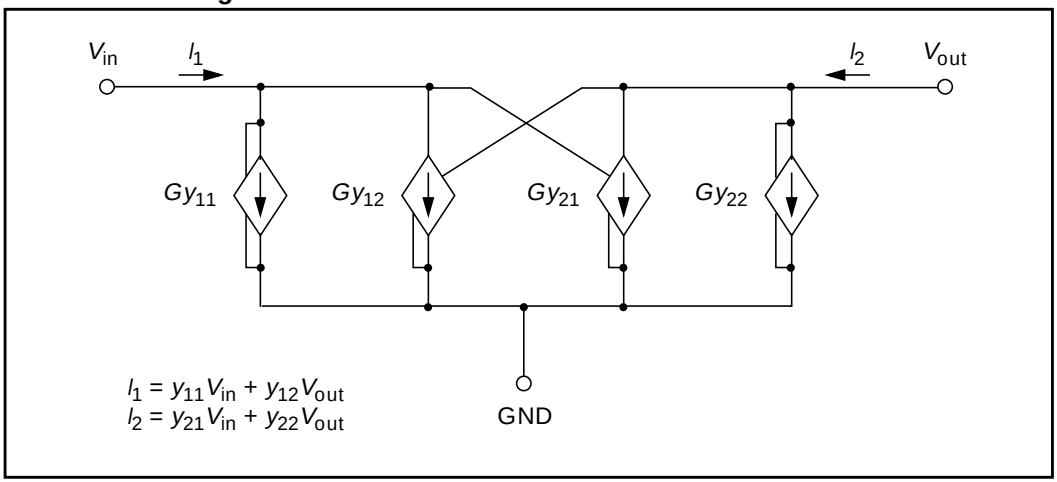
The general network in Figure 20-15 is described by the following equations, which can be translated into G Elements:

$$I_1 = Y_{11}V_{in} + Y_{12}V_{out}$$

$$I_2 = Y_{21}V_{in} + Y_{22}V_{out}$$

A schematic for a set of two-port Y parameters is shown in Figure 20-16. Note that the circuit is essentially composed of G Elements.

Figure 20-16: Schematic for the Y Parameter Network



The Laplace parameters for the Y parameter model are determined by a Pade expansion of the Y parameters of a transmission line, 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 and the line length (see 7 on page 20-54).

A Pade approximation contains polynomials in both the numerator and the denominator. Since a Pade approximation can model both poles and zeros and since *coth* and *csch* functions also contain both poles and zeros, a Pade approximation gives a better low order model than a series approximation.

A Pade expansion of *coth(p)* and *csch(p*), with second order numerator and third order denominator, is given below:

$$coth(p) \rightarrow \frac{\left(1 + \frac{2}{5} \cdot p^2\right)}{\left(p + \frac{1}{15} \cdot p^3\right)}$$

$$csch(p) \rightarrow \frac{\left(1 - \frac{1}{20} \cdot p^2\right)}{\left(p + \frac{7}{60} \cdot p^3\right)}$$

When you substitute ($s \cdot length \cdot \sqrt{LC}$) for *p*, you get 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_0 ($Z_o = \sqrt{L/C}$) in the matrix equation above, you get values you can use in a circuit file.

The circuit file shown below uses all of the above 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 would cause oscillations in the Star-Hspice response. To avoid this problem, use the same denominator in the *coth* and *csch* functions in the example. The simulation results may vary, depending on which denominator is used as the common denominator, because the coefficient of the third order term is changed (but by less than a factor of 2).

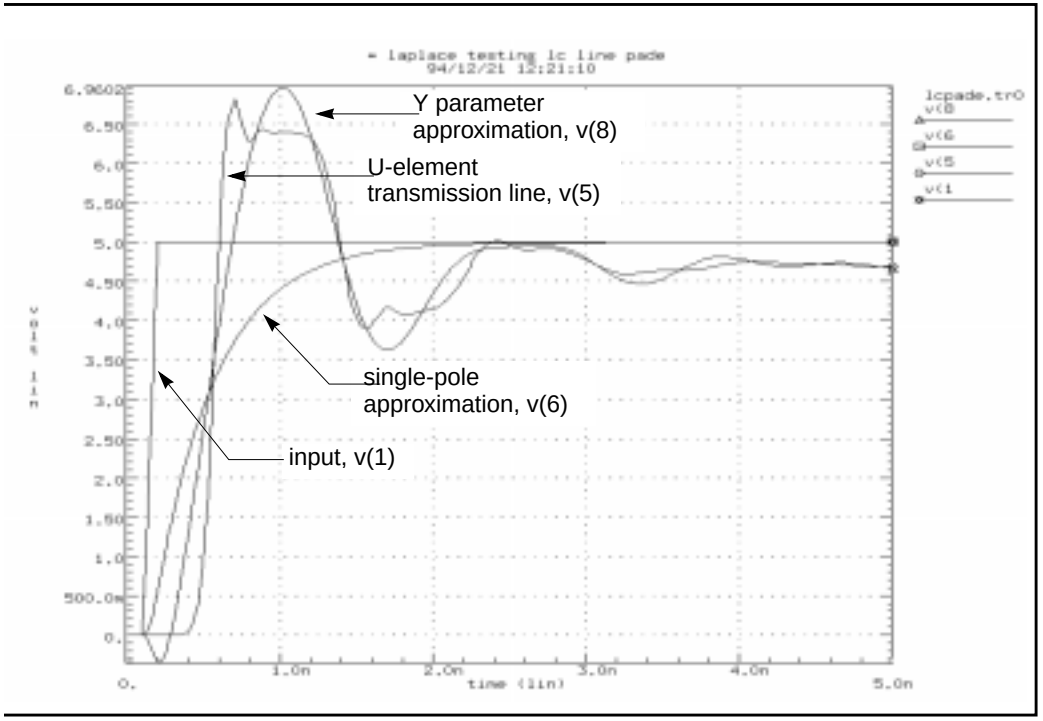
LC Line Circuit File

```
* Laplace testing LC line Pade
.Tran 0.02ns 5ns
.Options Post Accurate List Probe
v1 1 0 PWL 0ns 0 0.1ns 0 0.2ns 5
r1 1 2 25
r3 1 3 25
u1 2 0 5 0 wire1 L=0.1
r4 5 0 400
r8 8 0 400
e1 6 0 LAPLACE 1 0 1 / 1 0.4n
```

```
Gy11 3 0 LAPLACE 3 0 320016 0.0 2.048e-14 /
+ 0.0 0.0128 0.0 2.389e-22

Gy12 3 0 LAPLACE 8 0 -320016 0.0 2.56e-15 /
+ 0.0 0.0128 0.0 2.389e-22

Gy21 8 0 LAPLACE 3 0 -320016 0.0 2.56e-15 /
+ 0.0 0.0128 0.0 2.389e-22

Gy22 8 0 LAPLACE 8 0 320016 0.0 2.048e-14 /
+ 0.0 0.0128 0.0 2.389e-22

.model wire1 U Level=3 PLEV=1 ELEV=3 LLEV=0 MAXL=20
+ ZK=100 DELAY=4.0n

.Probe v(1) v(5) v(6) v(8)
.Print v(1) v(5) v(6) v(8)
.End
```

Figure 20-17 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 was not corrected for the load impedance, so the function produces an incorrect final voltage level. As expected, the Y parameter model gives the correct final voltage level. Although the Y parameter model gives a good approximation of the circuit delay, it contains too few poles to model the transient details fully. However, the Y parameter model does give excellent agreement with the overshoot and settling times.

Figure 20-17: 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 total CPU times for the two methods are shown in Simulation Time Summary. For larger circuits, the computation time saving can be much higher.

The input listings for each model type are shown below. Figure 20-18 and Figure 20-19 display the transient and frequency response comparisons resulting from the two modeling methods.

Circuit Model Input Listing

```
low_pass9a.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, 494-496.
*

.PARAM freq=100 tstop='2.0/freq'
*.PZ v(out) vin
.AC dec 50 .1k 100k
.OPTIONS dcstep=1e3 post probe unwrap
.PROBE ac vdb(out) vp(out)
.TRAN STEP='tstop/200' STOP=tstop
.PROBE v(out)
vin in GND sin(0,1,freq) ac 1
.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 opamp 2 4
eop2 3 0 opamp 6 4
.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
```

Pole/Zero Model Input Listing

```
ninth.sp 9th order low_pass filter.
.PARAM twopi=6.2831853072
.PARAM freq=100 tstop='2.0/freq'
.AC dec 50 .1k 100k
.OPTIONS dcstep=1e3 post probe unwrap
.PROBE ac vdb(outp) vp(outp)
.TRAN STEP='tstop/200' STOP=tstop
.PROBE v(outp)
vin in GND sin(0,1,freq) ac 1
Epole outp GND POLE in GND   417.6153
+    0. 3.8188k
+    0. 4.0352k
+    0. 4.7862k
+    0. 7.8903k / 1.0
+   '73.0669*twopi'     3.5400k
+   '289.3438*twopi'    3.4362k
+   '755.0697*twopi'    3.0945k
+   '1.5793k*twopi'     2.1105k
+   '2.1418k*twopi'     0.
repole outp GND 1e12
.END
```

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 20-18: Transient Responses of the Circuit and Pole/Zero Models

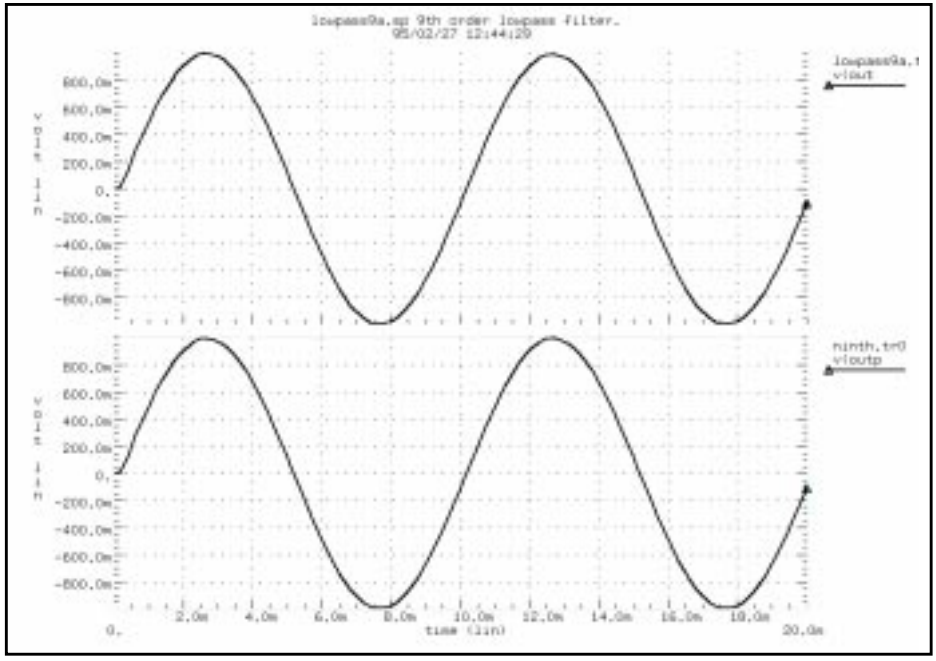
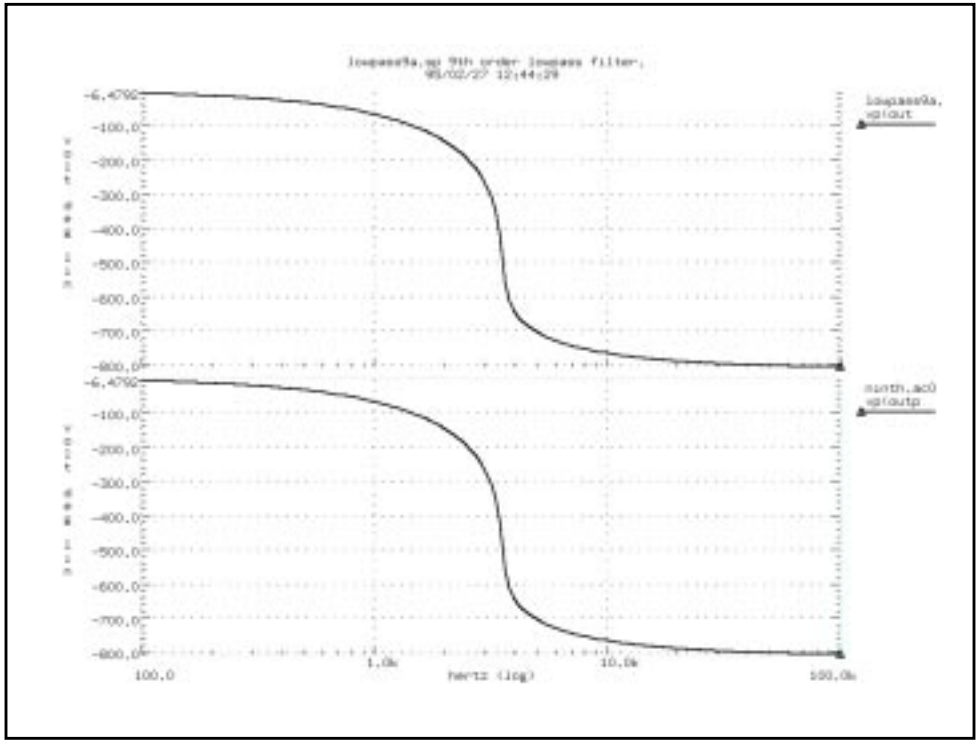


Figure 20-19: AC Analysis Responses of the Circuit and Pole/Zero Models



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 the capacitance.

Construct a filter from MOSFETs and capacitors where the filter characteristics are a function of the switching frequency of the MOSFETs.

In order to quickly determine the filter characteristics, use ideal switches (voltage controlled resistors) instead of MOSFETs. The resulting simulation speedup can be as great as 7 to 10 times faster than a circuit using MOSFETs.

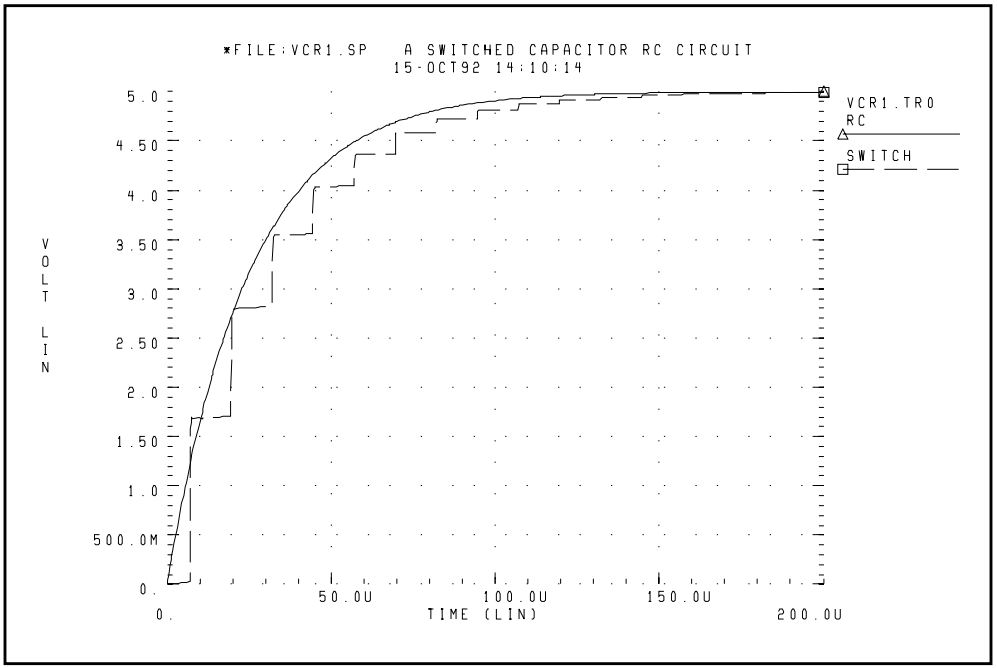
The model constructs an RC network using a resistor and a capacitor along with a switched capacitor equivalent network. Node RCOUT is the resistor/capacitor output, and VCROUT is the switched capacitor output.

The switches GVCR1 and GVCR2, together with the capacitance C3, model the resistor. The resistor value is calculated as:

$$\text{Res} = \frac{\text{Tswitch}}{\text{C3}}$$

where Tswitch is the period of the pulses PHI1 and PHI2.

Figure 20-20: VCR1.SP Switched Capacitor RC Circuit



Example

```
*FILE:VCR1.SP   A SWITCHED CAPACITOR RC CIRCUIT
.OPTIONS acct NOMOD POST
.IC V(SW1)=0 V(RCOUT)=0 V(VCROUT)=0
.TRAN 5U 200U
.GRAPH RC=V(RCOUT) SWITCH=V(VCROUT) (0,5)
VCC   VCC     GND     5V
C     RCOUT   GND     1NF
R     VCC     RCOUT   25K
C6    VCROUT GND     1NF

* equivalent circuit for 25k resistor r=12.5us/.5nf
VA    PHI1 GND    PULSE 0 5 1US .5US .5US 3US 12.5US
VB    PHI2 GND    PULSE 0 5 7US .5US .5US 3US 12.5US
GVCR1 VCC SW1 PHI1 GND LEVEL=1 MIN=100 MAX=1MEG 1.MEG
+ -.5MEG
GVCR2 SW1 VCROUT PHI2 GND LEVEL=1 MIN=100 MAX=1MEG
+ 1.MEG .5MEG
C3    SW1     GND    .5NF
.END
```

Switched Capacitor Filter Example

This example is a fifth-order elliptic switched capacitor filter with passband 0-1 kHz, loss less than 0.05 dB. It results from cascading models the switches with a resistance of 1 ohm when the switch is closed and 100 Megohm when it is open. The E Element models op-amps as an ideal op-amp. The transient response of the filter is provided for 1 kHz and 2 kHz sinusoidal input signal (see 8 on page 20-54).

Figure 20-21: Linear Section

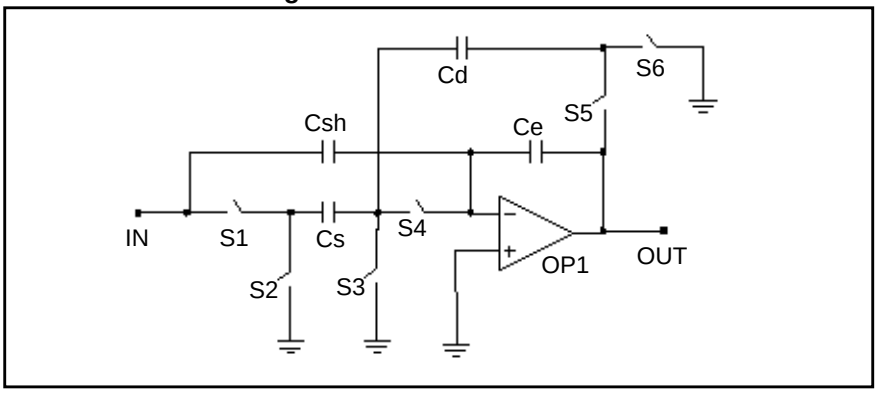


Figure 20-22: High_Q Biquad Section

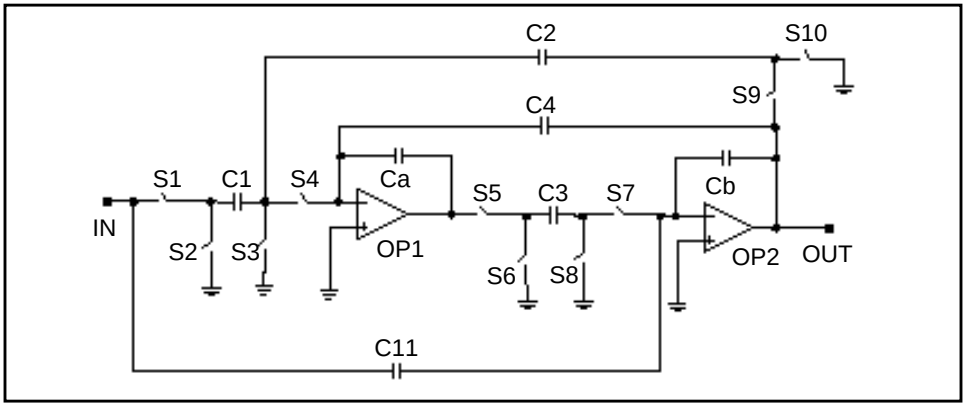
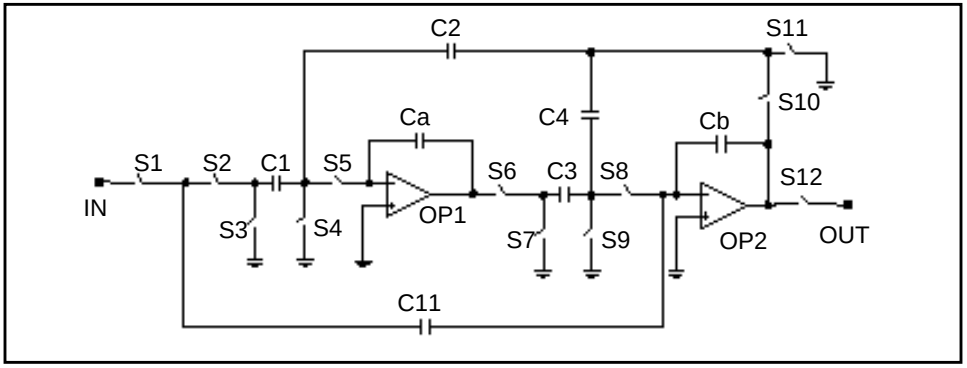


Figure 20-23: Low_Q Biquad Section



Star-Hspice Input File for Switched Capacitor Filter

```
SWCAP5.SP Fifth Order Elliptic Switched Capacitor
+ Filter.

.OPTIONS POST PROBE
.GLOBAL phi1 phi2
.TRAN 2u 3.2m UIC
*.GRAPH v(phi1) v(phi2) V(in)
.PROBE V(out)
*.PLOT v(in) v(phi1) v(phi2) v(out
*Iin   0    in SIN(0,1ma,1.0khz)
Iin   0    in SIN(0,1v,2khz)
Vphi1 phi1 0 PULSE(0,2 00u,.5u,.5u,7u,20u)
Vphi2 phi2 0 PULSE(0,2 10u,.5u,.5u,7u,20u)
Rsrc in   0 1k
Rload out 0 1k
Xsh   in   out1 sh
Xlin out1 out2 linear
Xhq   out2 out3 hqbiq
Xlq   out3 out lqbiq
```

Sample and Hold

```
.SUBCKT sh in out
Gs1 in 1 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Eop1 out 0 OPAMP 1 out
Ch    1 0 1.0pf
.ENDS
```

Linear Section

```
.SUBCKT linear in out
Gs1 in 1 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Gs2 1 0 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Cs   1 2 1.0pf
Gs3 2 0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Gs4 2 3 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Eop1 out 0 OPAMP 0 3
Ce   out 3 9.6725pf
Gs5 out 4 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Gs6 4   0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Cd   4   2 1.0pf
Csh in 3 0.5pf
.ENDS
```

High_Q Biquad Section

```
.SUBCKT hqbiq in out
Gs1 in 1 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Gs2 1 0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
C1   1 2 0.5pf
Gs3 2 0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Gs4 2 3 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Eop1 4 0 OPAMP 0 3
Ca   3 4 7.072pf
Gs5 4 5 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Gs6 5 0 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
C3   5 6 0.59075pf
Gs7 6 7 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Gs8 6 0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Eop2 out 0 OPAMP 0 7
Cb   7 out 4.3733pf
Gs9 out 8 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Gs10 8   0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
C4   out 3 1.6518pf
C2   8   2 0.9963pf
C11 7 in 0.5pf
.ENDS
```

Low_Q Biquad Section

```
.SUBCKT lqbiq in out
Gs1 in 1 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Gs2 1 2 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
C1   2 3 0.9963pf
Gs3 2 0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Gs4 3 0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Gs5 3 4 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Ca   4 5 8.833pf
Eop1 5 0 OPAMP 0 4
Gs6 5 6 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Gs7 6 0 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
C3   6 7 1.0558pf
Gs8 7 8 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Gs9 7 0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
Eop2 9 0 OPAMP 0 8
Cb   8 9 3.8643pf
Gs10 9 10 VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
Gs11 10 0 VCR PWL(1) phi1 0 0.5v,100meg 1.0v,1.0
C4   10 7 0.5pf
C2   10 3 0.5pf
C11 8 1 3.15425pf
Gs12 9 out VCR PWL(1) phi2 0 0.5v,100meg 1.0v,1.0
.ENDS
.END
```

Figure 20-24: Response to 1-kHz Sinusoidal Input

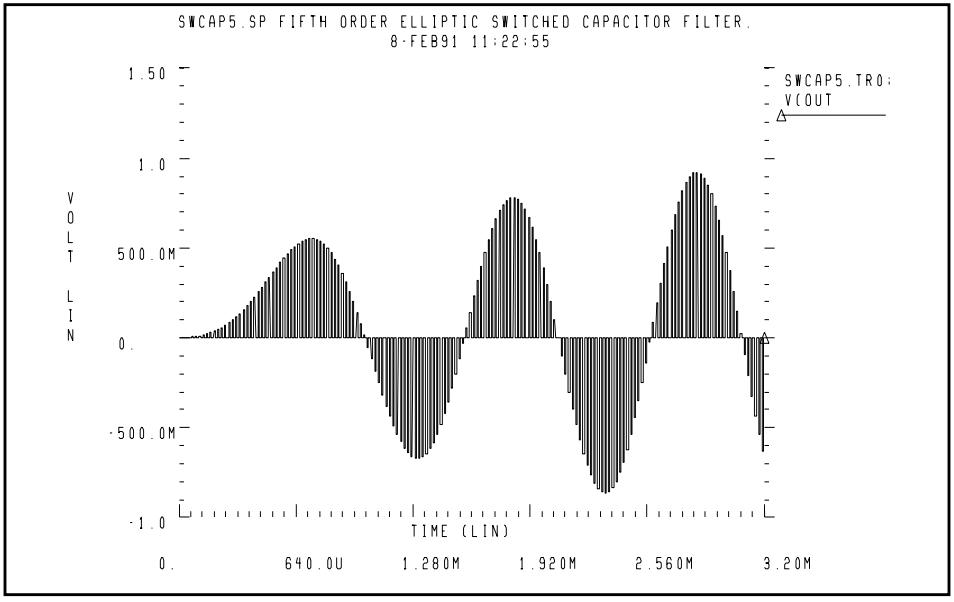
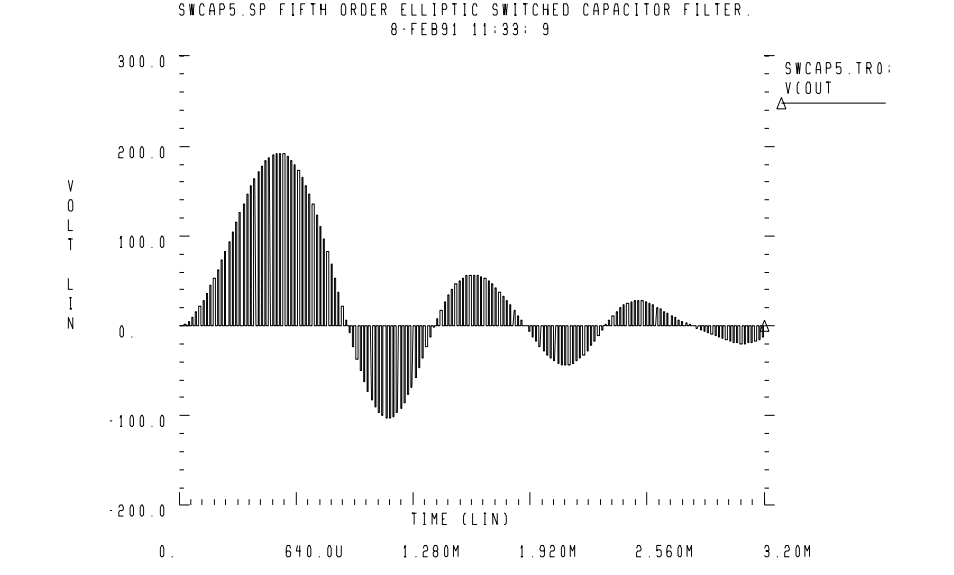


Figure 20-25: Response to 2-kHz Sinusoidal Input



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.

Avant!

Chapter 21
Timing Analysis Using Bisection

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. Then when a circuit timing failure occurs, you can identify a timing constraint that can lead to a design guideline. You must be able to perform an iterative analysis to define the violation specification.

Typical types of timing constraint violations include:

- Data setup time before clock
- Data hold time after clock
- Minimum pulse width required to allow a signal to propagate to the output
- Maximum toggle frequency of the component(s)

This chapter describes how to use the Star-Hspice bisection function in timing optimization. The general topic of optimization with Star-Hspice is covered in depth in Statistical Analysis and Optimization on page 13-1.

The following topics are covered in this chapter:

Understanding Bisection

Formerly, engineers built external drivers to submit multiple parameterized Star-Hspice jobs. Each job explored a region of the operating envelope of the circuit. The driver also had to provide part of the analysis by post-processing the Star-Hspice results to deduce the limiting conditions.

Characterizing circuits in this way is associated with small jobs, so individual analysis times are relatively small, compared with the overall job time. This method is inefficient, due to the overhead of submitting the job, reading and checking the netlist, and setting up the matrix. New methods increase efficiency in analyzing timing violations, to find the causes of timing failure. Bisection optimization method is an efficient cell characterization method in Star-Hspice.

Star-Hspice bisection methodology saves time in three ways:

- Reduction of multiple jobs to a single characterization job
- Removal of post-processing requirements
- Use of accuracy-driven iteration

Figure 21-1 illustrates a typical analysis of setup time constraints. A cell is driven by clock and data input waveforms. There are two input transitions, rise and fall, that occur at times T_1 and T_2. The result is an output transition, when V(out) goes from low to high. The following relationship between times T_1(data) and T_2 (clock) must be true in order for the V(out) transition to occur:

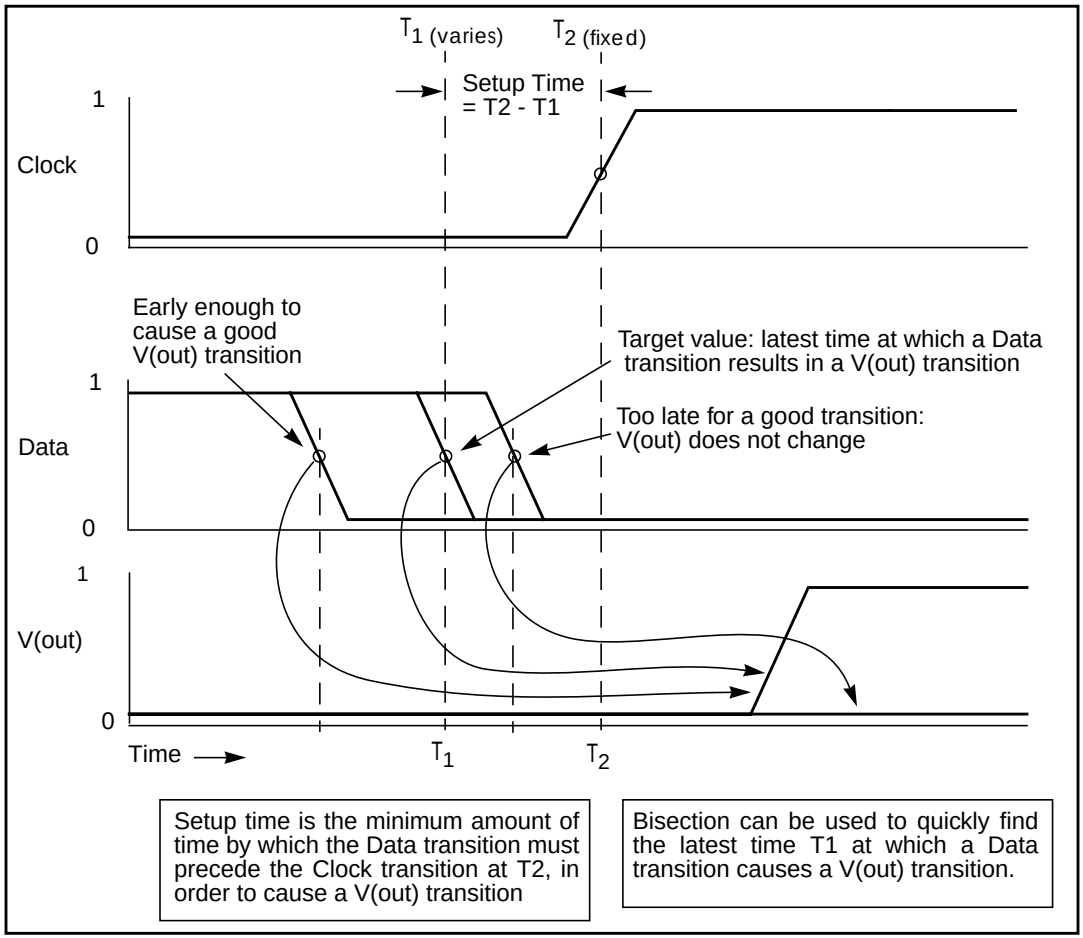
$T_2 > (T_1 + \text{setup time})$

The goal of the characterization, or violation analysis, is to determine the setup time. This is done by keeping T_2 fixed while repeating the simulation with different values of T_1 and observing which T_1 values produce the output transition and which values do not.

Previously, you had to run tight sweeps of the delay between the data setup and clock edge, looking for the value at which the transition fails to occur. 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 not accurate unless 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, while monitoring the outcome.

For example, even if you know that the desired transition occurs during a particular five-nanosecond period, searching for the setup time to within 0.1 nanoseconds over that five nanosecond period can require 50 simulations. Even after this, the error in the result can be as large as 0.05 nanoseconds.

Figure 21-1: Determining Setup Time with Bisection Violation Analysis



The Star-Hspice bisection feature greatly reduces the amount of work and computational time required to find an accurate solution to this type of problem. The following pages show examples of using this feature, to identify setup, hold, and minimum clock pulse width timing violations.

Understanding the Bisection Methodology

Bisection is a method of optimization that employs a binary search method to find the value of an input variable (target value) associated with a “goal” value of an output variable. The input and output variables may be of various types – for example, voltage, current, delay time or gain– related by some transfer function. In general, use a binary search to locate the output variable goal value within a search range of the input variable by iteratively halving that range to converge rapidly on the target value. At each iteration the “measured value” of the output variable is compared with the goal value. Bisection is employed in both the “pass/fail” method and the “bisection” method (see Using Bisection on page 21-5). The process is largely the same for either case.

The Star-Hspice bisection procedure involves two steps when solving the timing violation problem. First, the procedure detects whether the output transition occurred. Second, the procedure automatically varies the input parameter (T_1 in Figure 21-1) to find the value for which the transition barely occurs. The Star-Hspice measurement and optimization features handle these two steps.

Measurement

Use the Star-Hspice MAX measurement function to detect success or failure of an output transition. In the case of a low-to-high output transition, a MAX measurement produces zero on failure, or approximately the supply voltage V_{dd} 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, given a single measurement with a goal and known upper and lower boundary values for the input parameter. The characterization engineer should be able to specify acceptable upper and lower boundary values.

Using Bisection

To use bisection, the following is required:

- A user-specified pair of upper and lower boundary input variable values. For a solution to be found, one of these values must result in an output variable result |goal value| and the other in a result < |goal value|
- Specified goal value
- Error tolerance value. The bisection process stops when the difference between successive test values ≤ error tolerance. If the other criteria are met, see below.
- Related variables. Variables must be related by a monotonic transfer function, where a steadily progressing time (increase or decrease) results in a single occurrence of the "goal" value at the "target" input variable value

The error tolerance is included in a relation used as a process-termination criterion.

Figure 21-2 shows an example of the binary search process used by the bisection algorithm. This example is of the "pass/fail" type, and is appropriate for a setup-time analysis that tests for the presence of an output transition as shown in Figure 21-1. Here, a long setup time T_S *(=* T_2 - T_1*)* results in a VOUT transition (a "pass"), and a too-short setup time (where the latch has not stabilized the input data before the clock transition) results in a "fail." A "pass" time value, for example, might be defined as any setup time TS that produces a VOUT output "minimum high" logic output level of 2.7 V – the "goal" value. The "target" value is that setup time that *just* produces the V_{OUT} value of 2.7 V. Since finding the exact value is impractical, if not impossible, an error tolerance is specified to give a solution arbitrarily close to the target value. The bisection algorithm performs tests for each of the specified boundary values to determine the direction in which to pursue the target value after the first bisection. In this example, the upper boundary value is a "pass" value, and the lower boundary value is a "fail" value.

To start the binary search, a lower boundary and upper boundary are specified. The program tests the point midway between the lower and upper boundaries (see Figure 21-2).

If the initial value passes the test, the target value must be less than the tested value (in this case). The bisection algorithm moves the upper search limit to the value it just tested. If the test fails, the target value must greater than the tested value. The bisection algorithm moves the lower limit to the value it just tested.

Then the algorithm tests a value midway between the new limits. 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. The process stops when the difference between the latest test values is less than or equal to the user-specified error tolerance (normalized by multiplying by the initial boundary range).

Examining the Command Syntax

```
.MODEL <OptModelName> OPT METHOD=BISECTION ...
.MODEL <OptModelName> OPT METHOD=PASSFAIL ...
```

OptModel- The model to be used. Refer to “Statistical Analysis and Optimization” on page Chapter 131 for *Name* information on specification of optimization models in Star-Hspice.

METHOD Keyword to indicate which optimization method to use. For bisection, the method may be one of the following:

BISECTION

If the difference between the two latest test input values is within the error tolerance, and the latest measured value exceeds the goal, bisection succeeds, and stops. The process reports the optimized parameter that corresponded to the test value that satisfies this error tolerance, and this goal (passes).

PASSFAIL

If the difference between the two latest test input values is within the error tolerance, and one of the values ≥ goal (passes) and the other fails, bisection has succeeded and stops. The process reports the value the input parameter value associated with the “pass” measurement.

OPT Keyword to indicate optimization is to be performed

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. For 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 being optimized.

Note that the bisectional search is applied to only one parameter.

When the OPTLST option is set (.OPTION OPTLST=1), the process prints 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>
```

where x is the old parameter value and xnew is the new parameter value.

When .OPTION OPTLST=1, the process prints 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
```

(measfail = 0 for a test failure for the x value).

Example: transient analysis .TRAN statement:

```
.TRAN <TranStep> <TranTime> SWEEP OPTIMIZE=<OptParFun>
+ RESULTS=<MeasureNames> MODEL=<OptModelName>
```

Example: transient .MEASURE statement:

```
.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 */bisect/dff_top.sp* in the Star-Hspice *$installdir/demo/hspice* demonstration file directory. The files in Figure 21-2 and Figure 21-3 show the results of this demo. Setup time is not optimized directly, but is extracted from its relationship with the DelayTime parameter (the time preceding the data signal), which is the parameter being optimized.

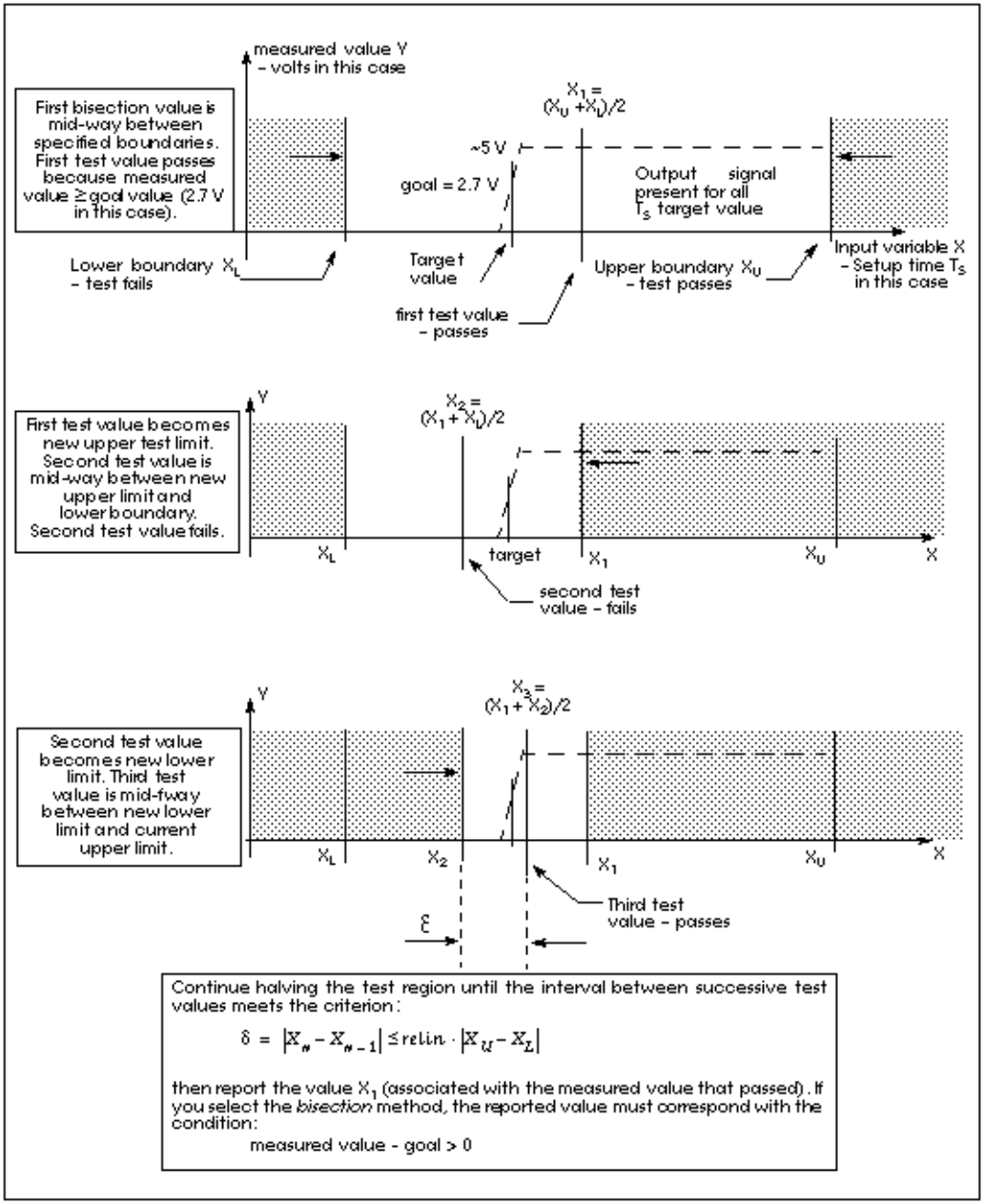
Input listing

```
File: $installdir/demo/hspice/bisect/dff_top.sp
* 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 0
+ 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 1n 8n Sweep   Optimize = 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 SetupTime   Trig v(Data)   Val = 'v(Vdd)/2'
+                                        Fall = 1
+                         Targ v(Clock)  Val = 'v(Vdd)/2'
+                                        Rise = 1

* Optimization Model
.MODEL OptMod Opt
+ Method = Bisection
.OPTIONS Post Brief NoMod

**********************************
* AvanLink to Cadence Composer by Avant!
* Hspice Netlist
* May 31 15:24:09 1994
**********************************

.MODEL nmos nmos LEVEL=2
.MODEL pmos pmos LEVEL=2
.Global vdd gnd

.SUBCKT XGATE control in n_control out
m0 in n_control out vdd pmos l=1.2u w=3.4u
m1 in control out gnd nmos l=1.2u w=3.4u
.ends

.SUBCKT INV in out wp=9.6u wn=4u l=1.2u
mb2 out in gnd gnd nmos l=l w=wn
mb1 out in vdd vdd pmos l=l w=wp
.ends

.SUBCKT DFF c d nc nq
Xi64 nc net46 c net36 XGATE
Xi66 nc net38 c net39 XGATE
Xi65 c nq nc net36 XGATE
Xi62 c d nc net39 XGATE
Xi60 net722 nq INV
Xi61 net46 net38 INV
Xi59 net36 net722 INV
Xi58 net39 net46 INV
c20 net36 gnd c=17.09f
c15 net39 gnd c=15.51f
c12 net46 gnd c=25.78f
c4 nq gnd c=25.28f
c3 net722 gnd c=19.48f
c16 net38 gnd c=16.48f
.ENDS
```

```
*-------------------------------------------------------
* Main Circuit Netlist:
*-------------------------------------------------------

v14 vdd gnd dc=5
c10 vdd gnd c=35.96f
c15 d_output gnd c=21.52f
c12 dff_nq gnd c=11.73f
c11 net31 gnd c=42.01f
c14 net27 gnd c=34.49f
c13 net25 gnd c=41.73f
c8 clock gnd c=5.94f
c7 data gnd c=7.93f
Xi3 net25 net31 net27 dff_nq DFF l=1u wn=3.8u wp=10u
Xi6 data net31 INV
Xi5 net25 net27 INV
Xi4 clock net25 INV
Xi2 dff_nq d_output INV wp=26.4u wn=10.6u
.END
```

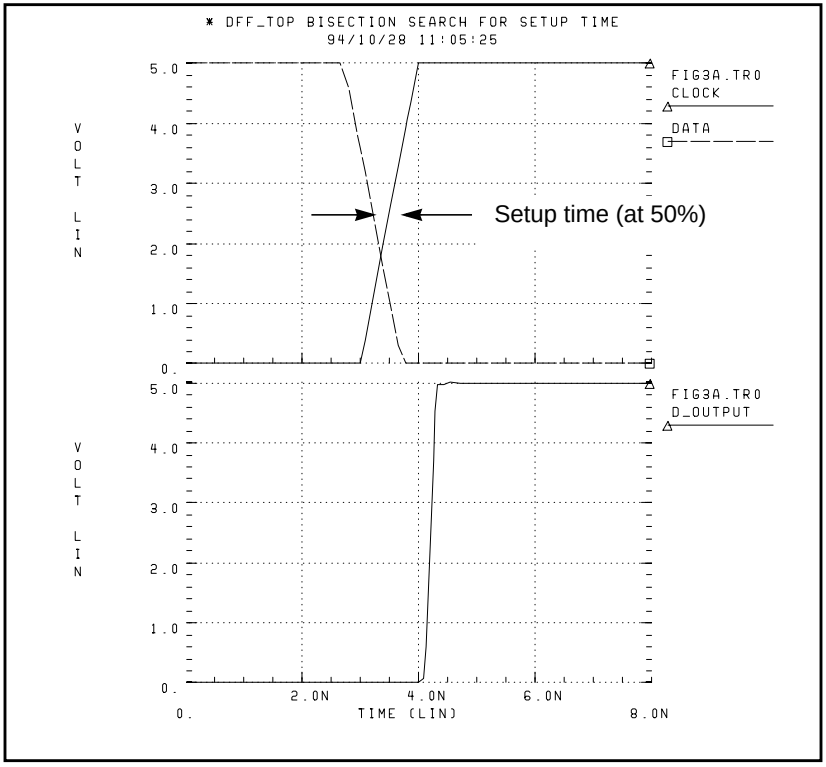
Figure 21-2: Bisection Example for Three Iterations



Results

The top plot in Figure 21-3 shows the relationship between the clock and data pulses that determine the setup time. The bottom plot is the output transition.

Figure 21-3: Transition at Minimum Setup Time



Find the actual value for the setup time in the “Optimization, Results” section of the Star-Hspice listing file:

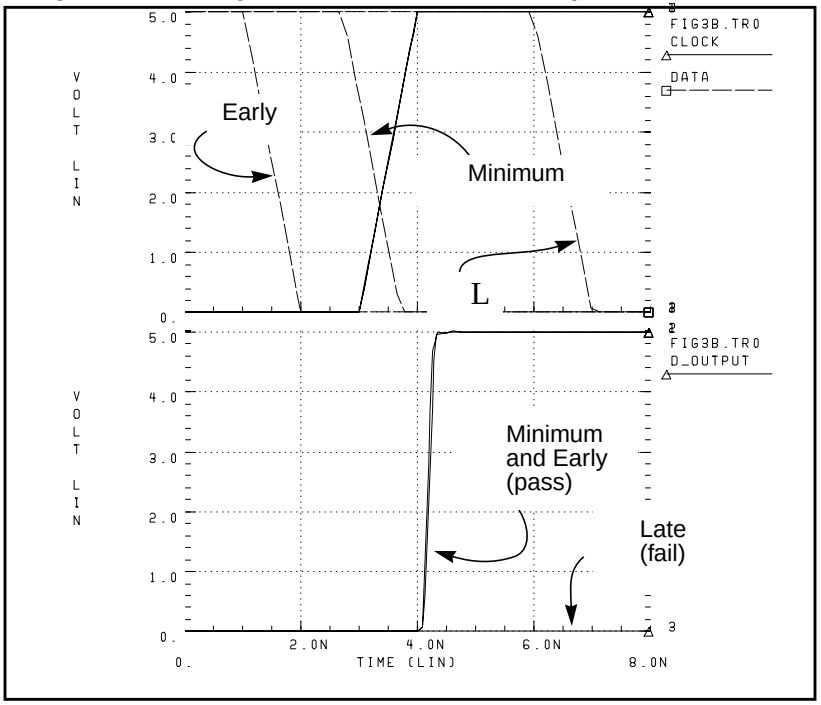
```
optimization completed, the condition
relin = 1.0000E-03 is satisfied
**** optimized parameters opt1
.PARAM DelayTime = 1.7188n
...
maxvout = 5.0049E+00    at= 4.5542E-09
from     =  .0000E+00    to= 8.0000E-09
setuptime= 2.8125E-10 targ= 3.5000E-09
+ trig= 3.2188E-09
```

This listing file excerpt shows that the optimal value for the setup time is 0.28125 nanoseconds.

The top plot in Figure 21-4 shows examples of early and late data transitions, as well as the transition at the minimum setup time. The bottom plot shows how the timing of the data transition affects the output transition. These results were produced with the following analysis statement:

```
* Sweep 3 values for DelayTime      Early   Optim    Late
.TRAN 1n 8n Sweep DelayTime Poi 3   0.0n    1.7188n  5.0n
```

Figure 21-4: 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 = 1.719E-09 *** $ Optimal
setuptime= 2.8120E-10 targ= 3.5000E-09   trig= 3.2188E-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 to allow the input pulse to propagate to the output of an inverter. The circuit for this example is */bisect/inv_a.sp*. in the *$installdir/demo/hspice* directory. The results of this demo are shown in Figure 21-5.

Input listing

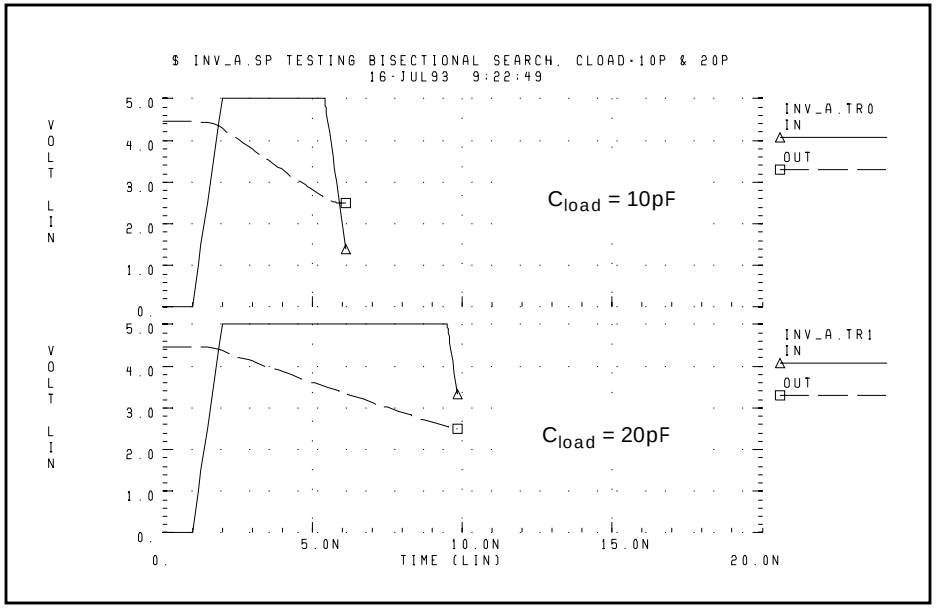
```
File: $installdir/demo/bisect/inv_a.sp
$ Inv_a.sp testing bisectional search, cload=10p & 20p
*
* Parameters
.PARAM Cload  =10p             $ See end of deck for Alter
.PARAM Tpw    =opt1(0,0,15n)   $ Used in Pulsed Voltage
                               $ Source, v1

* Transient simulation with PassFail Optimization
.TRAN .1n 20n Sweep    Optimize     = Opt1
+                      results      = Tprop
+                      Model        = Optmod
.MODEL OptMod Opt      Method       = PassFail
.MEASURE Tran Tprop    Trig v(in)   Val=2.5 Rise=1
+                      Targ v(out)  val=2.5 Rall=1
.OPTION nomod acct=3 post autostop
.GLOBAL 1

* The Circuit
vcc 1 0 5
vin in 0 pulse(0,5 1n 1n 1n Tpw 20n)
rin in 0 1e13
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 1 1 pch W=10u L=1u
.ENDS

* Models
.PARAM
+ mult1=1 xl=0.06u xwn=0.3u xwp=0.3u
+ tox=200 delvton=0 delvtop=0 rshn=50
+ rshp=150
```

```
.MODEL nch nmos
+ LEVEL=28
+ lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl xw=xwn tox=tox delvto=delvton rsh=rshn
+ ld=0.06u wd=0.2u acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0 js=3e-04 jsw=9e-10
+ cj=3e-04 mj=.5 pb=.8 cjsw=3e-10 mjsw=.3 php=.8
+ fc=.5 capop=4 xqc=.4 meto=0.08u
+ tlev=1 cta=0 ctp=0 tlevc=0 nlev=0
+ trs=1.6e-03 bex=-1.5 tcv=1.4e-03
* dc model
+ x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x2m=0
+ vfb0=-.5 phi0=0.65 k1=.9 k2=.1 eta0=0
+ muz=500 u00=.075 x3ms=15 u1=.02 x3u1=0
+ b1=.28 b2=.22 x33m=0.000000e+00
+ alpha=1.5 vcr=20 n0=1.6 wfac=15 wfacu=0.25
+ lvfb=0 lk1=.025 lk2=.05 lalpha=5
.Model pch pmos
+ LEVEL=28
+ lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl xw=xwp tox=tox delvto=delvtop rsh=rshp
+ ld=0.08u wd=0.2u acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0 rsh=rshp js=3e-04 jsw=9e-10
+ cj=3e-04 mj=.5 pb=.8 cjsw=3e-10 mjsw=.3 php=.8
+ fc=.5 capop=4 xqc=.4 meto=0.08u
+ tlev=1 cta=0 ctp=0 tlevc=0 nlev=0
+ trs=1.6e-03 bex=-1.5 tcv=-1.7e-03
* dc model
+ x2e=0 x3e=0 x2u1=0 x2ms=0 x2u0=0 x2m=5
+ vfb0=-.1 phi0=0.65 k1=.35 k2=0 eta0=0
+ muz=200 u00=.175 x3ms=8 u1=0 x3u1=0.0
+ b1=.25 b2=.25 x33m=0.0 alpha=0 vcr=20
+ n0=1.3 wfac=12.5 wfacu=.2 lvfb=0 lk1=-.05
*
* Alter for second load value
*
.ALTER $ repeat optimization for 20p load
.PARAM Cload=20p
.END
```

Results

Figure 21-5 shows the results of the pass/fail search for two different capacitive loads.

Figure 21-5: Results of Bisectional Pass/Fail Search



Avant!

Chapter 22
Running Demonstration Files

This chapter contains examples of basic file construction techniques, advanced features, and simulation tricks. Several Star-Hspice input files are listed and described.

The following topics are covered in this chapter:

Using the Demo Directory Tree

The last section of this chapter is a listing of demonstration files, which are designed as good training examples.These examples are included with most Star-Hspice distributions in the *demo* directory tree, where *$installdir* is the installation directory environment variable:

$installdir/demo/hspice	/alge	algebraic output
	/apps	general applications
	/behave	analog behavioral components
	/bench	standard benchmarks
	/bjt	bipolar components
	/cchar	cell characterization prototypes
	/ciropt	circuit level optimization
	/ddl	Discrete Device Library
	/devopt	device level optimization
	/fft	Fourier analysis
	/filters	filters
	/mag	transformers, magnetic core components
	/mos	MOS components
	/rad	radiation effects (photocurrent)
	/sources	dependent and independent sources
	/tline	filters and transmission lines

Running the Two-Bit Adder Demo

This two-bit adder shows techniques to improve circuit simulation efficiency, accuracy, and productivity. The adder in demonstration file *$installdir/demo/hspice/apps/mos2bit.sp* is composed of two-input NAND gates defined by the sub-circuit NAND. CMOS devices include length, width, and output loading parameters. Descriptive names enhance the readability of this circuit.

The sub-circuit ONEBIT defines the two half adders with carry in and carry out. The two-bit adder is created by two calls to ONEBIT. Independent piecewise linear voltage sources provide input stimuli. Complex waveforms are created by the “R” repeat function.

Figure 22-1: One-bit Adder sub-circuit

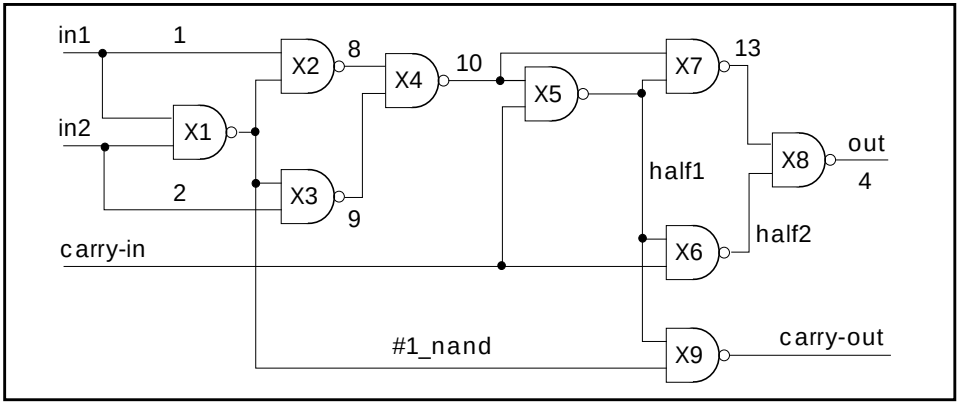


Figure 22-2: Two-bit Adder Circuit

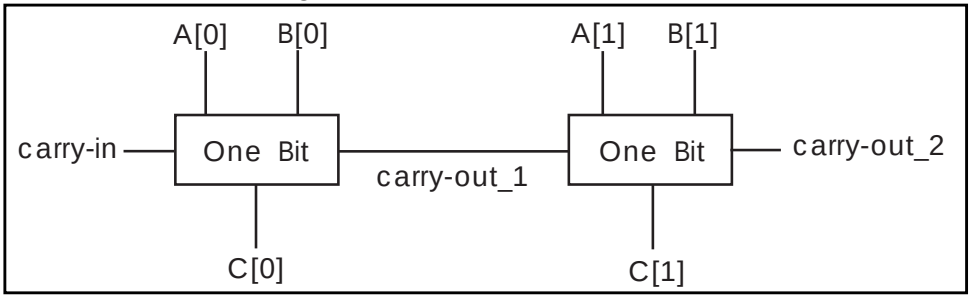
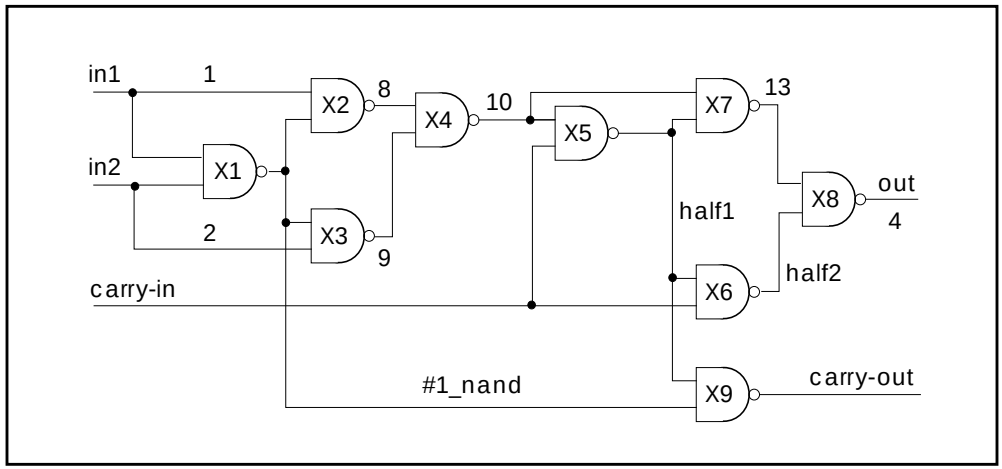


Figure 22-3: 1-bit NAND Gate Binary Adder



MOS Two-Bit Adder Input File

```
*FILE: MOS2BIT.SP - ADDER - 2 BIT ALL-NAND-GATE BINARY ADDER
.OPTIONS ACCT NOMOD FAST autostop scale=1u gmindc=100n
.param lmin=1.25 hi=2.8v lo=.4v vdd=4.5
.global vdd
.TRAN .5NS 60NS
.graph TRAN V(c[0]) V(carry-out_1) V(c[1]) V(carry-out_2)
+ par('V(carry-in)/6 + 1.5')
+ par('V(a[0])/6     + 2.0')
+ par('V(b[0])/6     + 2.5') (0,5)
.MEAS PROP-DELAY TRIG V(carry-in) TD=10NS VAL='vdd*.5'
+                                  RISE=1
+               TARG V(c[1])     TD=10NS VAL='vdd*.5'
+                                  RISE=3
*
.MEAS PULSE-WIDTH TRIG V(carry-out_1) VAL='vdd*.5' RISE=1
+                  TARG V(carry-out_1) VAL='vdd*.5' FALL=1
*
.MEAS FALL-TIME    TRIG V(c[1]) TD=32NS VAL='vdd*.9' FALL=1
+                  TARG V(c[1]) TD=32NS VAL='vdd*.1' FALL=1
vdd vdd gnd DC vdd
X1    A[0] B[0] carry-in     C[0] carry-out_1 ONEBIT
X2    A[1] B[1] carry-out_1 C[1] carry-out_2 ONEBIT
```

```
sub-circuit Definitions
.subckt NAND in1 in2 out   wp=10 wn=5
   M1 out in1 vdd vdd P W=wp L=lmin    ad=0
   M2 out in2 vdd vdd P W=wp L=lmin    ad=0
   M3 out in1 mid gnd N W=wn L=lmin    as=0
   M4 mid in2 gnd gnd N W=wn L=lmin    ad=0
   CLOAD out gnd 'wp*5.7f'
.ends
* switch model equivalent of the NAND. Gives a 10 times
* speedup over the MOS version.
.subckt NANDx in1 in2 out   wp=10 wn=5
  G1 out vdd vdd in1 LEVEL=1 MIN=1200 MAX=1MEG 1.MEG -.5MEG
  G2 out vdd vdd in2 LEVEL=1 MIN=1200 MAX=1MEG 1.MEG -.5MEG
  G3 out mid in1 gnd LEVEL=1 MIN=1200 MAX=1MEG 1.MEG -.5MEG
  G4 mid gnd in2 gnd LEVEL=1 MIN=1200 MAX=1MEG 1.MEG -.5MEG
   cout out gnd 300f
.ends
.subckt ONEBIT   in1  in2  carry-in  out   carry-out
   X1 in1      in2      #1_nand   NAND
   X2 in1      #1_nand   8         NAND
   X3 in2      #1_nand  9         NAND
   X4 8        9        10        NAND
   X5 carry-in 10       half1     NAND
   X6 carry-in half1    half2     NAND
   X7 10       half1    13        NAND
   X8 half2    13       out       NAND
   X9 half1    #1_nand  carry-out NAND
.ENDS ONEBIT
```

Stimuli

```
V1 carry-in gnd PWL(0NS,lo 1NS,hi 7.5NS,hi 8.5NS,lo 15NS lo R
V2 A[0] gnd PWL (0NS,hi 1NS,lo 15.0NS,lo 16.0NS,hi 30NS hi R
V3 A[1] gnd PWL (0NS,hi 1NS,lo 15.0NS,lo 16.0NS,hi 30NS hi R
V4 B[0] gnd PWL (0NS,hi 1NS,lo 30.0NS,lo 31.0NS,hi 60NS hi
V5 B[1] gnd PWL (0NS,hi 1NS,lo 30.0NS,lo 31.0NS,hi 60NS hi
```

Models

```
.MODEL N NMOS LEVEL=3 VTO=0.7 UO=500    KAPPA=.25 KP=30U
+ ETA=.01    THETA=.04 VMAX=2E5 NSUB=9E16 TOX=400 GAMMA=1.5
+ PB=0.6     JS=.1M    XJ=0.5U LD=0.1U   NFS=1E11 NSS=2E10
+ RSH=80     CJ=.3M    MJ=0.5   CJSW=.1N MJSW=0.3
+ acm=2 capop=4
.MODEL P PMOS LEVEL=3 VTO=-0.8 UO=150    KAPPA=.25 KP=15U
+ ETA=.015 THETA=.04 VMAX=5E4 NSUB=1.8E16 TOX=400
+ GAMMA=.672 PB=0.6    JS=.1M    XJ=0.5U LD=0.15U
+ NFS=1E11 NSS=2E10  RSH=80   CJ=.3M   MJ=0.5  CJSW=.1N
+ MJSW=0.3 acm=2 capop=4
.END
```

Running the MOS I-V and C-V Plotting Demo

It is often necessary to review the basic transistor characteristics to diagnose a simulation or modeling problem. This demonstration file, *$installdir/demo/hspice/mos/mosivcv.sp*, is a template file that can be used with any MOS model. The example shows the easy input file creation and the complete graphical results display. The following features aid model evaluations:

SCALE=1u	Sets the element units to microns from meters since users generally think in microns rather than meters
DCCAP	Forces the voltage variable capacitors to be evaluated during a DC sweep
node names	Makes the circuit easy to understand. Up to 16 characters can be used in the symbolic name
.GRAPH	.GRAPH statements create high resolution plots. A graph model can be added to set additional characteristics

This template provides the ability to get plots of internal variables such as:

i(mn1)	i1, i2, i3, or i4 can specify the true branch currents for each transistor node
LV18(mn6)	Total gate capacitance (C-V plot)
LX7(mn1)	Gate transconductance GM. (LX8 specifies GDS, and LX9 specifies GMB)

Figure 22-4: MOS IDS Plot

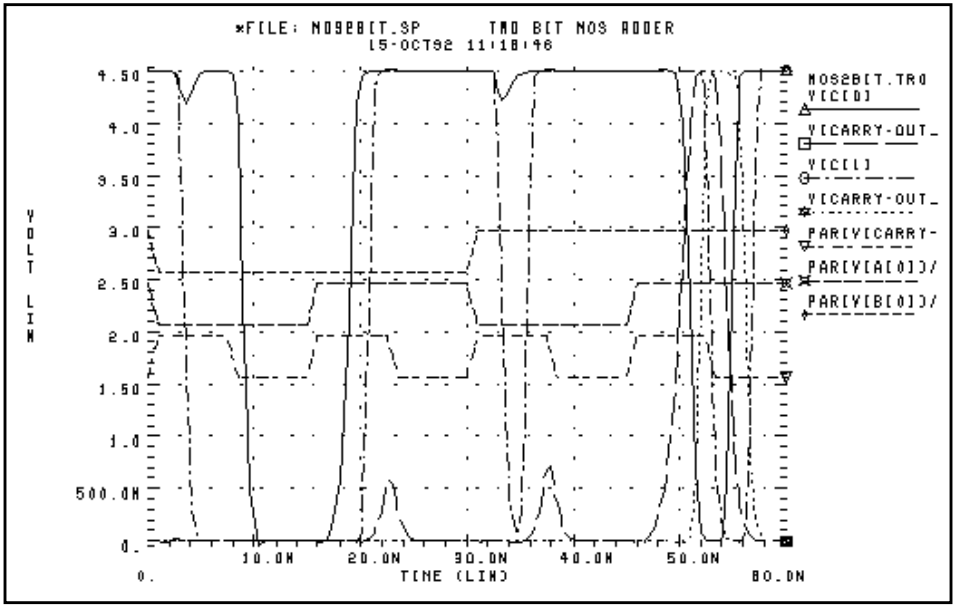


Figure 22-5: MOS VGS Plot

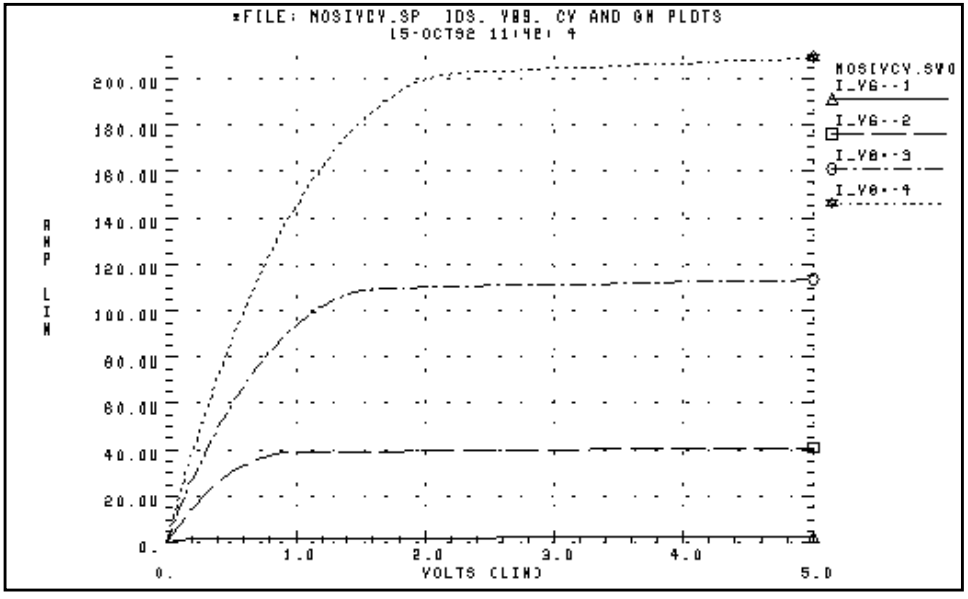


Figure 22-6: MOS GM Plot

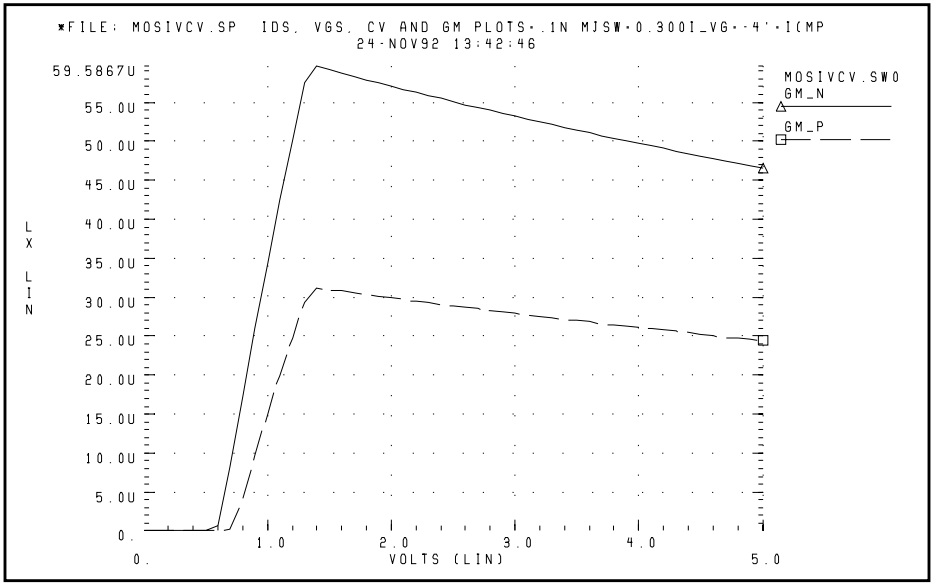
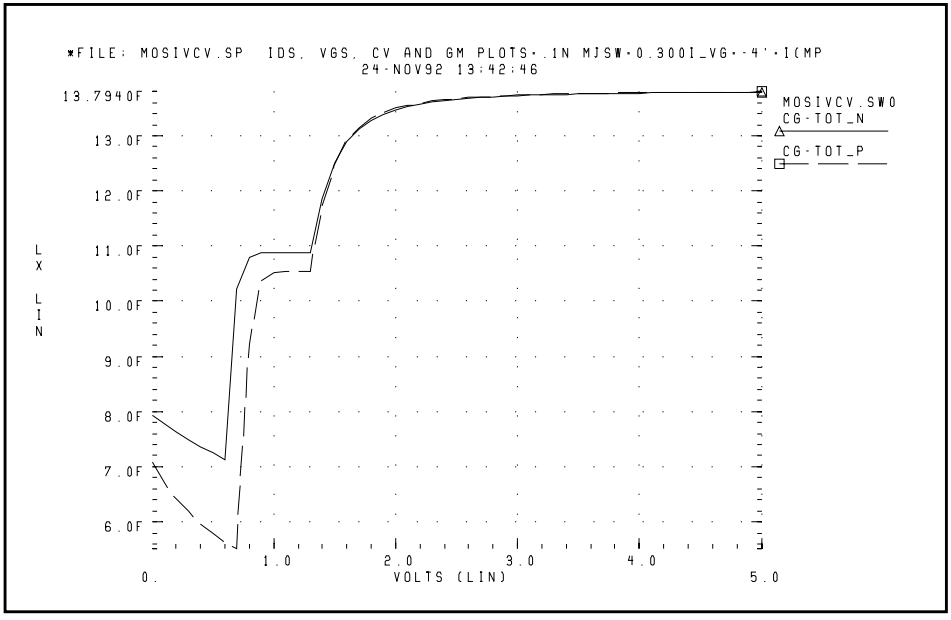


Figure 22-7: MOS C-V Plot



MOS I-V and C-V Plot Example Input File

```
*FILE: MOSIVCV.SP IDS, VGS, CV AND GM PLOTS

.OPTIONS SCALE=1U DCCAP
.DC VDDN 0 5.0 .1 $VBBN 0 -3 -3      sweep supplies
.PARAM ww=8 LL=2

$ ids-vds curves
.GRAPH 'I_VG=1' =I(MN1) 'I_VG=2' =I(MN2) 'I_VG=3' =I(MN3)
+ 'I_VG=4' =I(MN4)
.GRAPH 'I_VG=-1'=I(MP1) 'I_VG=-2'=I(MP2) 'I_VG=-3'=I(MP3)
+ 'I_VG=-4'=I(MP4)
$ ids-VGs curves
.GRAPH 'I_VD=.5'=I(MN6) 'I_VD=-.5'=I(MP6)
$ gate caps (cgs+cgd+cgb)
.GRAPH 'CG-TOT_N'=LX18(MN6) 'CG-TOT_P'= LX18(MP6)
$ gm
.GRAPH 'GM_N'=LX7(MN6) 'GM_P'=LX7(MP6)

VDDN vdd_n gnd 5.0
VBBN vbb_n gnd 0.0
EPD vdd_p gnd vdd_n gnd -1   $ reflect vdd for P devices
EPB vbb_p gnd vbb_n gnd -1   $ reflect vbb for P devices

V1 vg1 gnd 1
V2 vg2 gnd 2
V3 vg3 gnd 3
V4 vg4 gnd 4
V5 vddlow_n gnd .5
V-1 vg-1 gnd -1
V-2 vg-2 gnd -2
V-3 vg-3 gnd -3
V-4 vg-4 gnd -4
V-5 vddlow_p gnd -.5

MN1 vdd_n vg1 gnd vbb_n N W=ww L=LL
MN2 vdd_n vg2 gnd vbb_n N W=ww L=LL
MN3 vdd_n vg3 gnd vbb_n N W=ww L=LL
MN4 vdd_n vg4 gnd vbb_n N W=ww L=LL

MP1 gnd vg-1 vdd_p vbb_p P W=ww L=LL
MP2 gnd vg-2 vdd_p vbb_p P W=ww L=LL
MP3 gnd vg-3 vdd_p vbb_p P W=ww L=LL
MP4 gnd vg-4 vdd_p vbb_p P W=ww L=LL

MN6 vddlow_n vdd_n gnd vbb_n N W=ww L=LL
MP6 gnd vdd_p vddlow_p vbb_p P W=ww L=LL
```

```
.MODEL      N NMOS LEVEL=3   VTO=0.7 UO=500    KAPPA=.25
+ KP=30U    ETA=.01 THETA=.04 VMAX=2E5 NSUB=9E16 TOX=400
+ GAMMA=1.5 PB=0.6 JS=.1M    XJ=0.5U LD=0.1U   NFS=1E11
+ NSS=2E10 RSH=80 CJ=.3M    MJ=0.5   CJSW=.1N MJSW=0.3
+ acm=2     capop=4
*
.MODEL      P   PMOS LEVEL=3   VTO=-0.8 UO=150   KAPPA=.25
+ KP=15U   ETA=.015 THETA=.04 VMAX=5E4 NSUB=1.8E16 TOX=400
+ GAMMA=.67 PB=0.6   JS=.1M    XJ=0.5U LD=0.15U NFS=1E11
+ NSS=2E10 RSH=80   CJ=.3M    MJ=0.5   CJSW=.1N MJSW=0.3
+ acm=2     capop=4
.END
```

Running the CMOS Output Driver Demo

ASIC designers face the problem of integrating high performance IC parts onto a printed circuit board (PCB). The output driver circuit is most critical to the overall system performance. The demonstration file *$installdir/demo/hspice/apps/asic1.sp* shows the 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 a FR-4 printed circuit board
- Capacitive load at end of copper wire

The Star-Hspice strategy is to:

- Create a five-lump transmission line model for the copper wire
- Create single lumped models for leadframe loads

Figure 22-8: Noise Bounce

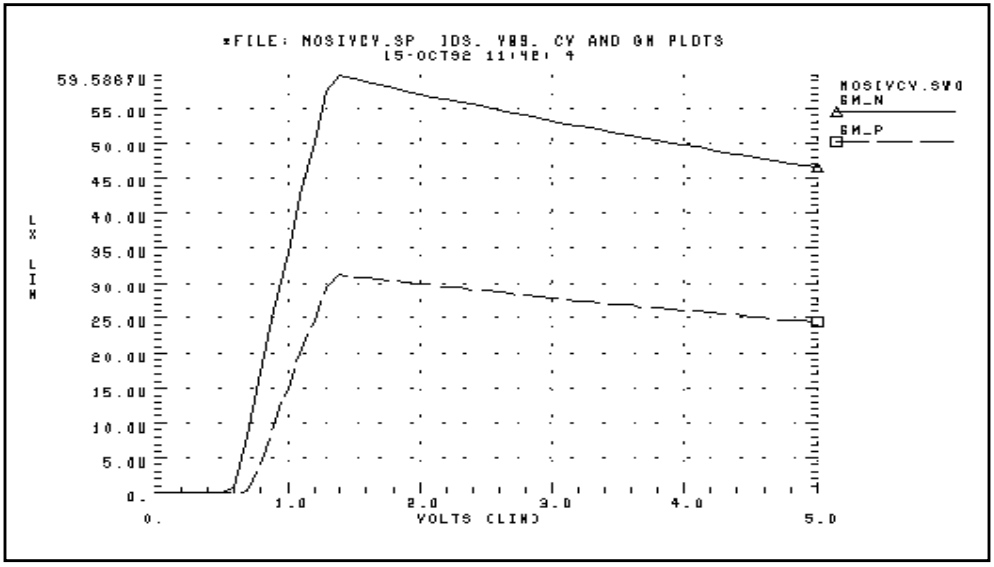


Figure 22-9: Asic1.sp Demo Local Supply Voltage

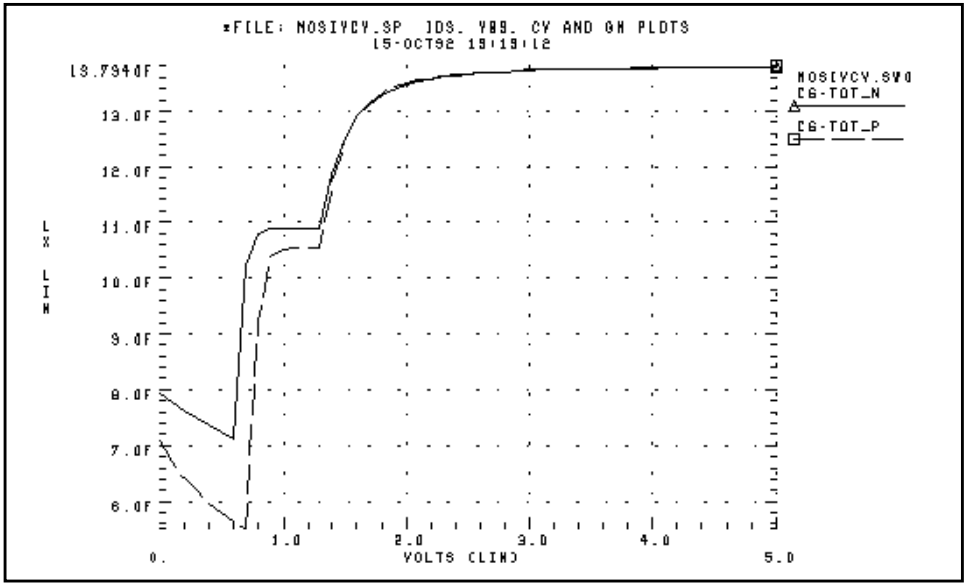


Figure 22-10: Asic1.sp Demo Local Supply Current

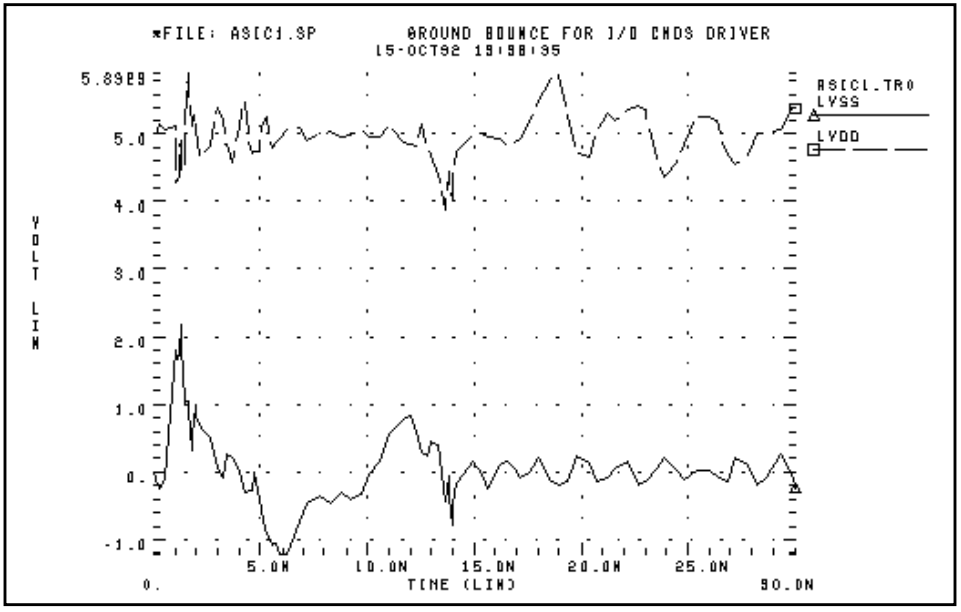
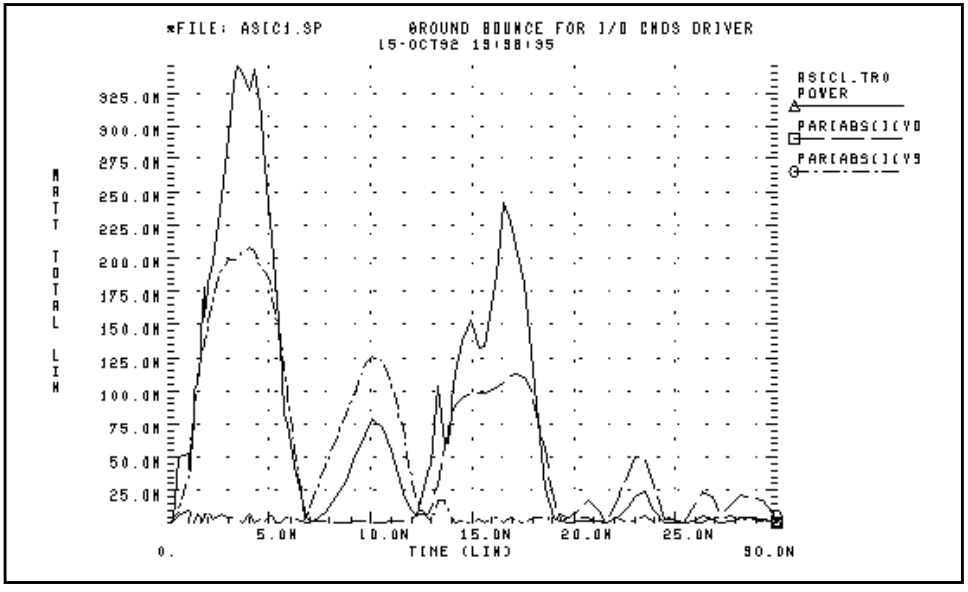


Figure 22-11: Asic1.sp Demo Input and Output Signals



CMOS Output Driver Example Input File

```
* FILE: ASIC1.SP
* SIMULATE AN OUTPUT DRIVER DRIVING 6 INCHES OF 6MIL PRINTED
*    CIRCUIT BOARD COPPER WITH 25PF OF LOAD CAPACITANCE
* MEASURE PEAK TO PEAK GROUND VOLTAGE
* MEASURE MAXIMUM GROUND CURRENT
* MEASURE MAXIMUM SUPPLY CURRENT
GROUND BOUNCE FOR I/O CMOS DRIVER 1200/1.2 & 800/1.2 MICRONS
.OPTIONS POST=2 RELVAR=.05
.TRAN .25N 30N
.MEASURE IVDD_MAX MAX PAR('ABS(I(VD))')
.MEASURE IVSS_MAX MAX PAR('ABS(I(VS))')
.MEASURE PEAK_GNDV PP V(LVSS)
.MEASURE PEAK_IVD   PP PAR(' ABS(I(VD)*V(VDD,OUT)) ')
.MEASURE PEAK_IVS   PP PAR(' ABS(I(VS)*V(VSS,OUT)) ')
.MEASURE RMS_POWER RMS POWER
.MEASURE FALL_TIME TRIG V(IN) RISE=1 VAL=2.5V
+                   TARG V(OUT) FALL=1 VAL=2.5V
.MEASURE RISE_TIME TRIG V(IN) FALL=1 VAL=2.5V
+                   TARG V(OUT) RISE=1 VAL=2.5V
.MEASURE TLINE_DLY TRIG V(OUT) RISE=1 VAL=2.5V
+                   TARG V(OUT2) RISE=1 VAL=2.5V
```

Input Signals

```
VIN IN LGND PWL(0N 0V, 2N 5V, 12N 5, 14N 0)
* OUTPUT DRIVER
MP1 LOUT IN LVDD LVDD P W=1400U L=1.2U
MN1 LOUT IN LVSS LVSS N W=800U L=1.2U
xout LOUT OUT LEADFRAME
*POWER AND GROUND LINE PARASITICS
Vd VDD GND 5V
xdd vdd lvdd leadframe
Vs VSS gnd 0v
xss vss lvss leadframe
*OUTPUT LOADING – 3 INCH FR-4 PC BOARD + 5PF LOAD +
*3 INCH FR-4 + 5PF LOAD
XLOAD1 OUT OUT1 GND LUMP5 LEN=3 WID=.006
CLOAD1 OUT1 GND 5PF
XLOAD2 OUT1 OUT2 GND LUMP5 LEN=3 WID=.006
CLOAD2 OUT2 GND 5PF
.macro leadframe in out
rframe in mid .01
lframe   mid out 10n
cframe   mid gnd .5p
.ends
```

```
*Transmission Line Parameter Definitions
.param rho=.6mho/sq cap=.55nf/in**2 ind=60ph/sq
*The 5-lump macro defines a parameterized transmission line
.macro lump5 in out ref len_lump5=1 wid_lump5=.1
.prot
.param reseff='len_lump5*rho/wid_lump5*5'
+       capeff='len_lump5*wid_lump5*cap/5'
+       indeff='len_lump5*ind/wid_lump5*5'
r1 in 1   reseff
c1   1 ref capeff
l1   1 2   indeff
r2   2 3   reseff
c2   3 ref capeff
l2   3 4   indeff
r3   4 5   reseff
c3   5 ref capeff
l3   5 6   indeff
r4   6 7   reseff
c4   7 ref capeff
l4   7 8   indeff
r5   8 9   reseff
c5   9 ref capeff
l5   9 out indeff
.unprot
.ends
```

Model Section

```
.MODEL N NMOS LEVEL=3 VTO=0.7 UO=500 KAPPA=.25 ETA=.03
+ THETA=.04 VMAX=2E5 NSUB=9E16 TOX=200E-10 GAMMA=1.5 PB=0.6
+ JS=.1M XJ=0.5U LD=0.0 NFS=1E11 NSS=2E10 capop=4
.MODEL P PMOS LEVEL=3 VTO=-0.8 UO=150 KAPPA=.25 ETA=.03
+ THETA=.04 VMAX=5E4 NSUB=1.8E16 TOX=200E-10 GAMMA=.672
+ PB=0.6 JS=.1M XJ=0.5U LD=0.0 NFS=1E11 NSS=2E10 capop=4
.end
IVDD_MAX        = 0.1141        AT= 1.7226E-08
            FROM= 0.0000E+00    TO= 3.0000E-08
IVSS_MAX        = 0.2086        AT= 3.7743E-09
            FROM= 0.0000E+00    TO= 3.0000E-08
PEAK_GNDV = 3.221     FROM= 0.0000E+00 TO= 3.0000E-08
PEAK_IVD   = 0.2929    FROM= 0.0000E+00 TO= 3.0000E-08
PEAK_IVS   = 0.3968    FROM= 0.0000E+00 TO= 3.0000E-08
RMS_POWER = 0.1233    FROM= 0.0000E+00 TO= 3.0000E-08
FALL_TIME = 1.2366E-09 TARG= 1.9478E-09 TRIG= 7.1121E-10
RISE_TIME = 9.4211E-10 TARG= 1.4116E-08 TRIG= 1.3173E-08
TLINE_DLY = 1.6718E-09 TARG= 1.5787E-08 TRIG= 1.4116E-08
```

Running the 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. Star-Hspice has first-order and second-order temperature coefficients in many of the critical model parameters to assure accurate simulations. There are two methods to optimize these temperature coefficients.

The first method uses the DC sweep variable TEMP. All of the Star-Hspice analysis sweeps allow two sweep variables; one of these must be the optimize variable to do an optimization. Sweeping TEMP limits the component to a linear element such as resistor, inductor, or capacitor. The second method uses multiple components at different temperatures.

The following example, demo file *$installdir/demo/hspice/ciropt/opttemp.sp*, simulates three circuits of a voltage source and 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 nonlinear components, such as bipolar transistors, diodes, and FETs.

Some simulation shortcuts are used in this example. In the internal output templates for resistors, LV1 (resistor) is the conductance (reciprocal resistance) at the desired temperature, allowing the optimization to be done in the resistance domain. To optimize more complex elements, use the current or voltage domain with measured sweep data. Also, the error function is expecting a sweep on at least two points, requiring the data statement to have two duplicate points.

Optimized Temperature Coefficients Example Input File

```
*FILE OPTTEMP.SP   OPTIMIZE RESISTOR TC1 AND TC2
v-25 1 0 1v
v0   2 0 1v
v+25 3 0 1v
r-25 1 0 rmod dtemp=-25
r0   2 0 rmod dtemp=0
r+25 3 0 rmod dtemp=25
.model rmod R res=1k tc1r=tc1r_opt tc2r=tc2r_opt
```

Optimization Section

```
.model optmod opt
.dc data=RES_TEMP optimize=opt1
+           results=r@temp1,r@temp2,r@temp3
+           model=optmod
.param tc1r_opt=opt1(.001,-.1,.1)
.param tc2r_opt=opt1(1u,-1m,1m)
.meas r@temp1 err2 par(R_meas_t1) par('1.0 / lv1(r-25)')
.meas r@temp2 err2 par(R_meas_t2) par('1.0 / lv1(r0) ')
.meas r@temp3 err2 par(R_meas_t3) par('1.0 / lv1(r+25) ')
* * Output section *
.dc data=RES_TEMP
.print 'r1_diff'=par('1.0/lv1(r-25)')
+      'r2_diff'=par('1.0/lv1(r0) ')
+      'r3_diff'=par('1.0/lv1(r+25)')
.data RES_TEMP R_meas_t1 R_meas_t2 R_meas_t3
950 1000 1010
950 1000 1010
.enddata
.end
```

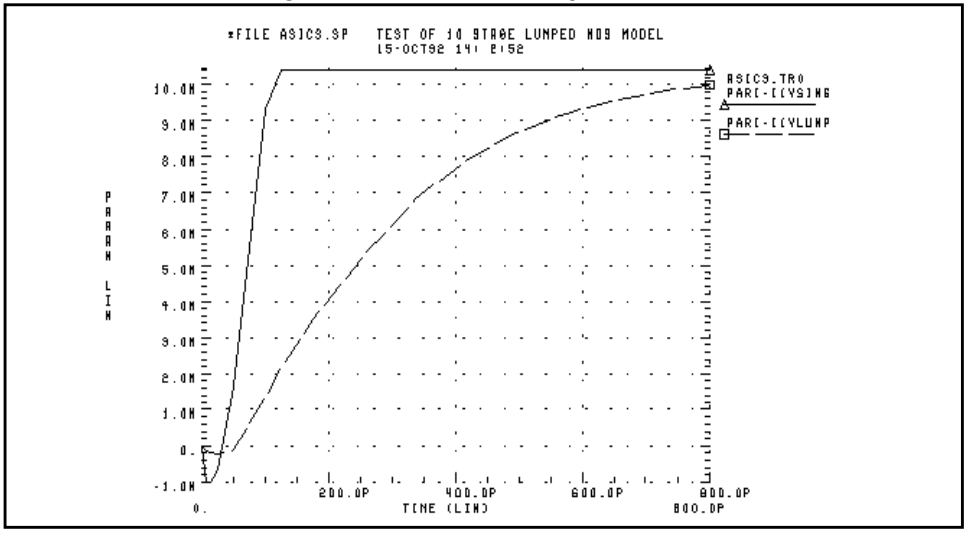
Simulating Electrical Measurements

In this example, Star-Hspice simulates the electrical measurements used to characterize devices for data sheet information. The demonstration file for this example is *$installdir/demo/hspice/ddl /t2n2222.sp*. The example automatically includes DDL models by reference using the DDLPATH environment variable, or through the .OPTION SEARCH='path'. It also combines an AC circuit and measurement with a transient circuit and measurement.

The AC circuit measures the maximum Hfe, the small signal common emitter gain. Star-Hspice uses the .MEASURE WHEN statement to calculate the unity gain frequency and the phase, at the specified frequency. In the "Transient Measurements" section of the input file, a segmented transient statement speeds up the simulation, and compresses the output graph. Measurements include:

- TURN ON from 90% of input rising to 90% of output falling
- OUTPUT FALL from 90% to 10% of output falling
- TURN OFF from 10% of input falling to 10% of output rising
- OUTPUT RISE from 10% to 90% of output rising

Figure 22-12: T2N2222 Optimization



T2N2222 Optimization Example Input File

```
* FILE: T2N2222.SP
** assume beta=200 ft250meg at ic=20ma and vce=20v
** for 2n2222
.OPTION   nopage autostop search=' '
*** ft measurement
* the net command is automatically reversing the sign of
* the power supply current for the network calculations
.NET I(vce) IBASE ROUT=50 RIN=50
VCE C 0 vce
IBASE 0 b   AC=1 DC=ibase
xqft c b 0 t2n2222
.ac dec 10 1 1000meg
.graph s21(m) h21(m)
.measure 'phase @h21=0db' WHEN h21(db)=0
.measure 'h21_max' max h21(m)
.measure 'phase @h21=0deg' FIND h21(p) WHEN h21(db)=0
.param ibase=1e-4 vce=20 tauf=5.5e-10
```

Transient Measurements

```
** vccf power supply for forward reverse step recovery time
** vccr power supply for inverse reverse step recovery time
**   VPLUSF positive voltage for forward pulse generator
**   VPLUSr positive voltage for reverse pulse generator
**   Vminusf positive voltage for forward pulse generator
**   Vminusr positive voltage for reverse pulse generator
**   rloadf load resistor for forward
**   rloadr load resistor for reverse
.param vccf=30v
.param VPLUSF=9.9v
.param VMINUSF=-0.5v
.param rloadf=200
.TRAN 1N 75N 25N 200N 1N 300N 25N 1200N
.measure 'turn-on time 'trig par('v(inf)-0.9*vplusf') val=0
+ rise=1 targ par('v(outf)-0.9*vccf') val=0 fall=1
.measure 'fall time 'trig par('v(outf)-0.9*vccf') val=0
+ fall=1 targ par('v(outf)-0.1*vccf') val=0 fall=1
.measure 'turn-off time' trig par('v(inf)-0.1*vplusf')
+ val=0 fall=1 targ par('v(outf)-0.1*vccf') val=0 rise=1
.measure 'rise time 'trig par('v(outf)-0.1*vccf') val=0
+ rise=1 targ par('v(outf)-0.9*vccf') val=0 rise=1
```

```
.graph V(INF) V(OUTF)
VCCF    VCCF   0      vccf
RLOADF  VCCF   OUTF   RLOADF
RINF    INF    VBASEF 1000
RPARF   INF    0      58
XSCOPf  OUTF   0      SCOPE
VINF    INF    0      PL    VMINUSF 0S   VMINUSF 5NS
+ VPLUSF 7NS VPLUSF 207NS VMINUSF 209NS
* CCX0F    VBASEF OUTF   CCX0F
* CEX0F    VBASEF 0      CEX0F
XQF      OUTF VBASEF   0    t2n2222

.MACRO SCOPE VLOAD VREF
RIN    VLOAD VREF 100K
CIN    VLOAD VREF   12P
.EOM
.END
```

Modeling Wide-channel MOS Transistors

Selecting an appropriate model for I/O cell transistors improves simulation accuracy. For wide-channel devices, model the transistor as a group of transistors connected in parallel with appropriate RC delay networks, rather than as one transistor, because the polysilicon gate introduces delay. When scaling to higher speed technologies, the area of the polysilicon gate decreases, reducing the gate capacitance. However, if you scale the gate oxide thickness, it increases the capacitance per unit area, increasing the RC product. The following example illustrates the effect on delay due to this scaling. 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

$$Rpoly = \frac{W}{L} \cdot 40 \qquad poly = \frac{Esio \cdot nsi}{tox} \cdot L \cdot W$$

$$Rpoly = \frac{100}{5} \cdot 40 = 800, \quad Co = \frac{3.9 \cdot 8.86}{800} \cdot 100 \cdot 5 = 215\ fF$$

RC = 138 ps

For a transistor with

- channel width = 100 microns
- channel length = 1.2 microns
- gate oxide thickness = 250 Angstroms

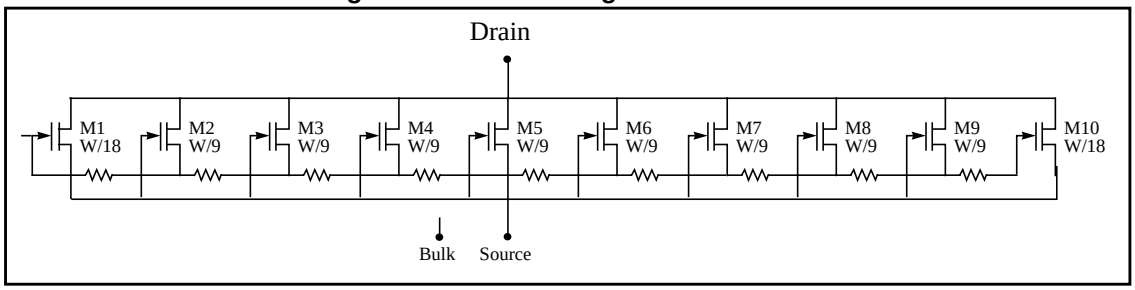
$$Rpoly = \frac{channel\ \ width}{channel\ \ length} \cdot 40$$

$$Co = \frac{3.9 \cdot 8.86}{Tox} \cdot channel\ \ width \cdot channel\ \ length$$

RC = 546 ps

You can model the RC delay introduced in modern CMOS technologies, using a nine-stage ladder model.

Figure 22-13: Nine-stage Ladder Model

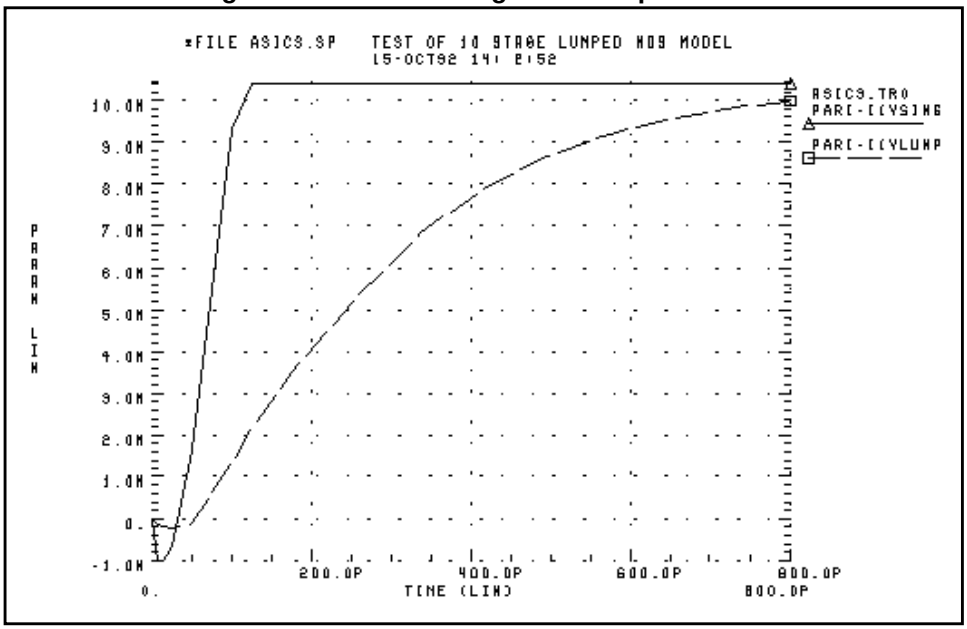


In this example, the nine-stage ladder model is in a data file, *$installdir/demo/hspice/apps /asic3.sp*, which Star-Hspice optimized (with actual measured data of a wide channel transistor as the target data). Optimization produced a nine-stage ladder model that matches timing characteristics of the physical data. The simulation results for the nine-stage ladder model and the one-stage model are compared, using the nine-stage ladder model as the reference. The one-stage model results are about 10% faster than the actual physical data indicates.

Example of Nine-Stage Ladder Model

```
* FILE: ASIC3.SP Test of 9 Stage Ladder Model
.subckt lrgtp drain gate source bulk
m1 drain gate source bulk p w='wt/18' l=lt
m2 drain g1 source bulk p w='wt/9' l=lt
m3 drain g2 source bulk p w='wt/9' l=lt
m4 drain g3 source bulk p w='wt/9' l=lt
m5 drain g4 source bulk p w='wt/9' l=lt
m6 drain g5 source bulk p w='wt/9' l=lt
m7 drain g6 source bulk p w='wt/9' l=lt
m8 drain g7 source bulk p w='wt/9' l=lt
m9 drain g8 source bulk p w='wt/9' l=lt
m10 drain g9 source bulk p w='wt/18' l=lt
r1 gate g1 'wt/lt*rpoly/9'
r2 g1 g2 'wt/lt*rpoly/9'
r3 g2 g3 'wt/lt*rpoly/9'
r4 g3 g4 'wt/lt*rpoly/9'
r5 g4 g5 'wt/lt*rpoly/9'
r6 g5 g6 'wt/lt*rpoly/9'
r7 g6 g7 'wt/lt*rpoly/9'
r8 g7 g8 'wt/lt*rpoly/9'
r9 g8 g9 'wt/lt*rpoly/9'
.ends lrgtp
.end pro
.end
```

Figure 22-14: Asic3 Single vs. Lumped Model



Examining Demonstration Input Files

Table 22-1: Demonstration Input Files *(Sheet 1 of 17)*

File Name	Description
Algebraic Output Variable Examples *$installdir/demo/hspice/alge*	
alg.sp	demonstration of algebraic parameters
alg_fil.sp	magnitude response of behavioral filter model
alg_vco.sp	voltage controlled oscillator
alg_vf.sp	voltage-to-frequency converter behavioral model
xalg1.sp	QA of parameters
xalg2.sp	QA of parameters
Applications of General Interest *$installdir/demo/hspice/apps*	
alm124.sp	AC, noise, transient op-amp analysis
alter2.sp	.ALTER examples
ampg.sp	pole/zero analysis of a G source amplifier
asic1.sp	ground bounce for I/O CMOS driver
asic3.sp	ten-stage lumped MOS model
bjt2bit.sp	BJT two-bit adder
bjt4bit.sp	four-bit, all NAND gate binary adder
bjtdiff.sp	BJT diff amp with every analysis type
bjtschmt.sp	bipolar Schmidt trigger
bjtsense.sp	bipolar sense amplifier
cellchar.sp	ASIC inverter cell characterization

Table 22-1: Demonstration Input Files *(Sheet 2 of 17)*

File Name	Description
crystal.sp	crystal oscillator circuit
gaasamp.sp	simple GaAsFET amplifier
grouptim.sp	group time delay example
inv.sp	sweep MOSFET -3 sigma to +3 sigma, use .MEASURE output
mcdiff.sp	CMOS differential amplifier
mondc_a.sp	Monte Carlo of MOS diffusion and photolithographic effects
mondc_b.sp	Monte Carlo DC analysis
mont1.sp	Monte Carlo Gaussian, uniform, and limit function
mos2bit.sp	two-bit MOS adder
pll.sp	phase locked loop
sclopass.sp	switched capacitor low-pass filter
worst.sp	worst case skew models using .ALTER
xbjt2bit.sp	BJT NAND gate two-bit binary adder
Behavioral Applications	*$installdir/demo/hspice/behave*
acl.sp	acl gate
amp_mod.sp	amplitude modulator with pulse waveform carrier
behave.sp	AND/NAND gates using G, E Elements
calg2.sp	voltage variable capacitance
det_dff.sp	double edge triggered flip-flop
diff.sp	differentiator circuit
diode.sp	behavioral diode using a PWL VCCS

Table 22-1: Demonstration Input Files *(Sheet 3 of 17)*

File Name	Description
dlatch.sp	CMOS D-latch using behaviorals
galg1.sp	sampling a sine wave
idealop.sp	ninth-order low-pass filter
integ.sp	integrator circuit
invb_op.sp	optimization of CMOS macromodel inverter
ivx.sp	characterization of PMOS and NMOS as a switch
op_amp.sp	op-amp from Chua and Lin
pd.sp	phase detector modeled by switches
pdb.sp	phase detector using behavioral NAND gates
pwl10.sp	operational amplifier used as voltage follower
pwl2.sp	PPW-VCCS with gain of 1 amp/volt
pwl4.sp	eight-input NAND gate
pwl7.sp	modeling inverter by a PWL VCVS
pwl8.sp	smoothing the triangle waveform by PWL CCCS
ring5bm.sp	five-stage ring oscillator – macromodel CMOS inverter
ringb.sp	ring oscillator using behavioral model
sampling.sp	sampling a sine wave
scr.sp	silicon controlled rectifier modelled with PWL CCVS
swcap5.sp	fifth-order elliptic switched capacitor filter
switch.sp	test for PWL switch element
swrc.sp	switched capacitor RC circuit

Table 22-1: Demonstration Input Files *(Sheet 4 of 17)*

File Name	Description
triode.sp	triode model family of curves using behavioral elements
triodex.sp	triode model family of curves using behavioral elements
tunnel.sp	modeling tunnel diode characteristic by PWL VCCS
vcob.sp	voltage controlled oscillator using PWL functions
Benchmarks	*$installdir/demo/hspice/bench*
bigmos1.sp	large MOS simulation
demo.sp	quick demo file to test installation
m2bit.sp	72-transistor two-bit adder – typical cell simulation
m2bitf.sp	fast simulation example
m2bitsw.sp	fast simulation example – same as m2bitf.sp but using behavioral elements
senseamp.sp	bipolar analog test case
Timing Analysis	*$installdir/demo/hspice/bisect*
fig3a.sp	DFF bisection search for setup time
fig3b.sp	DFF early, optimum, and late setup times
inv_a.sp	inverter bisection pass-fail
BJT and Diode Devices	*$installdir/demo/hspice/bjt*
bjtbeta.sp	plot BJT beta
bjtft.sp	plot BJT FT using s-parameters
bjtgm.sp	plot BJT Gm, Gpi
dpntun.sp	junction tunnel diode
snaphsp.sp	convert SNAP to Star-Hspice

Table 22-1: Demonstration Input Files *(Sheet 5 of 17)*

File Name	Description
tun.sp	tunnel oxide diode
Cell Characterization	*$installdir/demo/hspice/cchar*
dff.sp	DFF bisection search for setup time
inv3.sp	inverter characterization
inva.sp	inverter characterization
invb.sp	inverter characterization
load1.sp	inverter sweep, delay versus fanout
setupbsc.sp	setup characterization
setupold.sp	setup characterization
setuppas.sp	setup characterization
sigma.sp	sweep MOSFET -3 sigma to +3 sigma, use measure output
tdgtl.a2d	Viewsim A2D Star-Hspice input file
tdgtl.d2a	Viewsim D2A Star-Hspice input file
tdgtl.sp	two-bit adder using D2A Elements
Circuit Optimization	*$installdir/demo/hspice/ciropt*
ampgain.sp	set unity gain frequency of BJT diff pair
ampopt.sp	optimize area, power, speed of MOS amp
asic2.sp	optimize speed, power of CMOS output buffer
asic6.sp	find best width of CMOS input buffer
delayopt.sp	optimize group delay of LCR circuit
lpopt.sp	match lossy filter to ideal filter
opttemp.sp	find first and second temperature coefficients of resistor

Table 22-1: Demonstration Input Files *(Sheet 6 of 17)*

File Name	Description
rcopt.sp	optimize speed, power for RC circuit
DDL	*$installdir/demo/hspice/ddl*
ad8bit.sp	eight-bit A/D flash converter
alf155.sp	National JFET op-amp characterization
alf156.sp	National JFET op-amp characterization
alf157.sp	National JFET op-amp characterization
alf255.sp	National JFET op-amp characterization
alf347.sp	National JFET op-amp characterization
alf351.sp	National wide bandwidth JFET input op-amp characterization
alf353.sp	National wide bandwidth dual JFET input op-amp char.
alf355.sp	Motorola JFET op-amp characterization
alf356.sp	Motorola JFET op-amp characterization
alf357.sp	Motorola JFET op-amp characterization
alf3741.sp	
alm101a.sp	
alm107.sp	National op-amp characterization
alm108.sp	National op-amp characterization
alm108a.sp	National op-amp characterization
alm118.sp	National op-amp characterization
alm124.sp	National low power quad op-amp characterization
alm124a.sp	National low power quad op-amp characterization

Table 22-1: Demonstration Input Files *(Sheet 7 of 17)*

File Name	Description
alm158.sp	National op-amp characterization
alm158a.sp	National op-amp characterization
alm201.sp	LM201 op-amp characterization
alm201a.sp	LM201 op-amp characterization
alm207.sp	National op-amp characterization
alm208.sp	National op-amp characterization
alm208a.sp	National op-amp characterization
alm224.sp	National op-amp characterization
alm258.sp	National op-amp characterization
alm258a.sp	National op-amp characterization
alm301a.sp	National op-amp characterization
alm307.sp	National op-amp characterization
alm308.sp	National op-amp characterization
alm308a.sp	National op-amp characterization
alm318.sp	National op-amp characterization
alm324.sp	National op-amp characterization
alm358.sp	National op-amp characterization
alm358a.sp	National op-amp characterization
alm725.sp	National op-amp characterization
alm741.sp	National op-amp characterization
alm747.sp	National op-amp characterization

Table 22-1: Demonstration Input Files *(Sheet 8 of 17)*

File Name	Description
alm747c.sp	National op-amp characterization
alm1458.sp	National dual op-amp characterization
alm1558.sp	National dual op-amp characterization
alm2902.sp	National op-amp characterization
alm2904.sp	National op-amp characterization
amc1458.sp	Motorola internally compensated high performance op-amp characterization
amc1536.sp	Motorola internally compensated high voltage op-amp characterization
amc1741.sp	Motorola internally compensated high performance op-amp characterization
amc1747.sp	Motorola internally compensated high performance op-amp characterization
ane5534.sp	TI low noise, high speed op-amp characterization
anjm4558.sp	TI dual op-amp characterization
anjm4559.sp	TI dual op-amp characterization
anjm4560.sp	TI dual op-amp characterization
aop04.sp	PMI op-amp characterization
aop07.sp	PMI ultra low offset voltage op-amp characterization
aop14.sp	PMI op-amp characterization
aop15b.sp	PMI precision JFET input op-amp characterization
aop16b.sp	PMI precision JFET input op-amp characterization
at094cns.sp	TI op-amp characterization

Table 22-1: Demonstration Input Files *(Sheet 9 of 17)*

File Name	Description
atl071c.sp	TI low noise op-amp characterization
atl072c.sp	TI low noise op-amp characterization
atl074c.sp	TI low noise op-amp characterization
atl081c.sp	TI JFET op-amp characterization
atl082c.sp	TI JFET op-amp characterization
atl084c.sp	TI JFET op-amp characterization
atl092cp.sp	TI op-amp characterization
atl094cn.sp	TI op-amp characterization
aupc358.sp	NEC general dual op-amp characterization
aupc1251.sp	NEC general dual op-amp characterization
j2n3330.sp	JFET 2n3330 I-V characteristics
mirf340.sp	IRF340 I-V characteristics
t2n2222.sp	BJT 2n2222 characterization
Device Optimization	*$installdir/demo/hspice/devopt*
beta.sp	LEVEL=2 beta optimization
bjtopt.sp	s-parameter optimization of 2n6604 BJT
bjtopt1.sp	2n2222 DC optimization
bjtopt2.sp	2n2222 Hfe optimization
d.sp	diode, multiple temperatures
dcopt1.sp	1n3019 diode I-V and C-V optimization
gaas.sp	JFET optimization

Table 22-1: Demonstration Input Files *(Sheet 10 of 17)*

File Name	Description
jopt.sp	300u/1u GaAs FET DC optimization
jopt2.sp	JFET optimization
joptac.sp	300u/1u GaAs FET 40 MHz–20 GHz s-parameter optimization
l3.sp	MOS LEVEL 3 optimization
l3a.sp	MOS LEVEL 3 optimization
l28.sp	LEVEL=28 optimization
ml2opt.sp	MOS LEVEL=2 I-V optimization
ml3opt.sp	MOS LEVEL=3 I-V optimization
ml6opt.sp	MOS LEVEL=6 I-V optimization
ml13opt.sp	MOS LEVEL=13 I-V optimization
opt_bjt.sp	2n3947 forward and reverse Gummel optimization
Fourier Analysis	*$installdir/demo/hspice/fft*
am.sp	FFT analysis, AM source
bart.sp	FFT analysis, Bartlett window
black.sp	FFT analysis, Blackman window
dist.sp	FFT analysis, second harmonic distortion
exam1.sp	FFT analysis, AM source
exam3.sp	FFT analysis, high frequency signal detection test
exam4.sp	FFT analysis, small-signal harmonic distortion test
exp.sp	FFT analysis, exponential source
fft.sp	FFT analysis, transient, sweeping a resistor

Table 22-1: Demonstration Input Files *(Sheet 11 of 17)*

File Name	Description
fft1.sp	FFT analysis, transient
fft2.sp	FFT analysis on the product of three waveforms
fft3.sp	FFT analysis, transient, sweeping frequency
fft4.sp	FFT analysis, transient, Monte Carlo Gaussian distribution
fft5.sp	FFT analysis, data-driven transient analysis
fft6.sp	FFT analysis, sinusoidal source
gauss.sp	FFT analysis, Gaussian window
hamm.sp	FFT analysis, Hamming window
hann.sp	FFT analysis, Hanning window
harris.sp	FFT analysis, Blackman-Harris window
intermod.sp	FFT analysis, intermodulation distortion
kaiser.sp	FFT analysis, Kaiser window
mod.sp	FFT analysis, modulated pulse
pulse.sp	FFT analysis, pulse source
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

Table 22-1: Demonstration Input Files *(Sheet 12 of 17)*

File Name	Description
win.sp	FFT analysis, window test
window.sp	FFT analysis, window test
winreal.sp	FFT analysis, window test
Filters	*$installdir/demo/hspice/filters*
fbp_1.sp	bandpass LCR filter measurement
fbp_2.sp	bandpass LCR filter pole/zero
fbpnet.sp	bandpass LCR filter s-parameters
fbprlc.sp	LCR AC analysis for resonance
fhp4th.sp	high-pass LCR fourth-order Butterworth filter
fkerwin.sp	pole/zero analysis of Kerwin's circuit
flp5th.sp	low-pass fifth-order filter
flp9th.sp	low-pass ninth-order FNDR with ideal op-amps
micro1.sp	test of microstrip
micro2.sp	test of microstrip
tcoax.sp	test of RG58/AU coax
trans1m.sp	FR-4 printed circuit lumped transmission line
Magnetics	*$installdir/demo/hspice/mag*
aircore.sp	air core transformer circuit
bhloop.sp	b-h loop nonlinear magnetic core transformer
mag2.sp	three primary, two secondary magnetic core transformer
magcore.sp	magnetic core transformer circuit

Table 22-1: Demonstration Input Files *(Sheet 13 of 17)*

File Name	Description
royerosc.sp	Royer magnetic core oscillator
MOSFET Devices	*$installdir/demo/hspice/mos*
bsim3.sp	LEVEL=47 BSIM3 model
cap13.sp	plot MOS capacitances LEVEL=13 model
cap_b.sp	capacitances for LEVEL=13 model
cap_m.sp	capacitance for LEVEL=13 model
capop0.sp	plot MOS capacitances LEVEL=2
capop1.sp	plot MOS capacitances LEVEL=2
capop2.sp	plot MOS capacitances LEVEL=2
capop4.sp	plot MOS capacitances LEVEL=6
chrgpump.sp	charge conservation test LEVEL=3
iiplot.sp	impact ionization current plot
ml6fex.sp	plot temperature effects LEVEL=6
ml13fex.sp	plot temperature effects LEVEL=13
ml13ft.sp	s-parameters for LEVEL=13
ml13iv.sp	plot I-V for LEVEL=13
ml27iv.sp	plot I-V for LEVEL=27 SOSFET
mosiv.sp	plot I-V for user include file
mosivcv.sp	plot I-V and C-V for LEVEL=3
qpulse.sp	charge conservation test LEVEL=6
qswitch.sp	charge conservation test LEVEL=6

Table 22-1: Demonstration Input Files *(Sheet 14 of 17)*

File Name	Description
selector.sp	automatic width and length model selector
tgam2.sp	LEVEL=6 gamma model
tmos34.sp	MOS LEVEL=34 EPFL, test DC
Radiation Effects	*$installdir/demo/hspice/rad*
brad1.sp	bipolar radiation effects example
brad2.sp	bipolar radiation effects example
brad3.sp	bipolar radiation effects example
brad4.sp	bipolar radiation effects example
brad5.sp	bipolar radiation effects example
brad6.sp	bipolar radiation effects example
drad1.sp	diode radiation effects example
drad2.sp	diode radiation effects example
drad4.sp	diode radiation effects example
drad5.sp	diode radiation effects example
drad6.sp	diode radiation effects example
dradarb2.sp	diode radiation effects example
jex1.sp	JFET radiation effects example
jex2.sp	JFET radiation effects example
jprad1.sp	JFET radiation effects example
jprad2.sp	JFET radiation effects example
jprad4.sp	JFET radiation effects example

Table 22-1: Demonstration Input Files *(Sheet 15 of 17)*

File Name	Description
jrad1.sp	JFET radiation effects example
jrad2.sp	JFET radiation effects example
jrad3.sp	JFET radiation effects example
jrad4.sp	JFET radiation effects example
jrad5.sp	JFET radiation effects example
jrad6.sp	JFET radiation effects example
mrad1.sp	MOSFET radiation effects example
mrad2.sp	MOSFET radiation effects example
mrad3.sp	MOSFET radiation effects example
mrad3p.sp	MOSFET radiation effects example
mrad3px.sp	MOSFET radiation effects example
rad1.sp	total MOSFET dose example
rad2.sp	diode photocurrent test circuit
rad3.sp	diode photocurrent test circuit RLEV=3
rad4.sp	diode photocurrent test circuit
rad5.sp	BJT photocurrent test circuit with an NPN transistor
rad6.sp	BJT secondary photocurrent effect which varies with R1
rad7.sp	BJT RLEV=6 example (semi-empirical model)
rad8.sp	JFET RLEV=1 example with Wirth-Rogers square pulse
rad9.sp	JFET stepwise increasing radiation source
rad10.sp	GaAs RLEV=5 example (semi-empirical model)

Table 22-1: Demonstration Input Files *(Sheet 16 of 17)*

File Name	Description
rad11.sp	NMOS E-mode LEVEL=8 with Wirth-Rogers square pulse
rad12.sp	NMOS 0.5x resistive voltage divider
rad13.sp	three-input NMOS NAND gate with non-EPI, EPI, and SOS examples
rad14.sp	GaAs differential amplifier circuit
rad14dc.sp	n-channel JFET DC I-V curves
Sources	*$installdir/demo/hspice/sources*
amsrc.sp	amplitude modulation
exp.sp	exponential independent source
pulse.sp	test of pulse
pwl.sp	repeated piecewise linear source
pwl10.sp	op-amp voltage follower
rtest.sp	voltage controlled resistor inverter chain
sffm.sp	single frequency FM modulation source
sin.sp	sinusoidal source waveform
vcr1.sp	switched capacitor network using G-switch
Transmission Lines	*$installdir/demo/hspice/tline*
fr4.sp	microstrip test FR-4 PC board material
fr4o.sp	optimizing model for microstrip FR-4 PC board material
fr4x.sp	FR4 microstrip test
hd.sp	ground bounce for I/O CMOS driver
rcsnubts.sp	ground bounce for I/O CMOS driver at snubber output

Table 22-1: Demonstration Input Files *(Sheet 17 of 17)*

File Name	Description
rcsnubtt.sp	ground bounce for I/O CMOS driver
strip1.sp	two series microstrips (8 mil and 16 mil wide)
strip2.sp	two microstrips coupled together
t14p.sp	1400 mil by 140 mil, 50 ohm tline on FR-4 50 MHz - 10.05 GHz
t14xx.sp	1400 mil by 140 mil, 50 ohm tline on FR-4 optimization
t1400.sp	1400 mil by 140 mil, 50 ohm tline on FR-4 optimization
tcoax.sp	RG58/AU coax with 50 ohm termination
tfr4.sp	microstrip test
tfr4o.sp	microstrip test
tl.sp	series source coupled and shunt terminated transmission lines
transmis.sp	algebraics and lumped transmission lines
twin2.sp	twinlead model
xfr4.sp	microstrip test sub-circuit expanded
xfr4a.sp	microstrip test sub-circuit expanded, larger ground resistance
xfr4b.sp	microstrip test
xulump.sp	test 5-, 20-, and 100-lump U models

Avant!

Appendix A
FAQ/Troubleshooting

This appendix contains frequently asked questions (FAQ) and their solutions for the following topics:

- Analysis
- Documentation
- Environment Variables
- Error Messages
- Input
- Installation Issues
- Licensing/Access Issues
- Limitations
- Miscellaneous
- Models
- MS Windows/PC Issues
- Netlist/Options
- Output
- W Element/Field Solver
- Waveform Viewing

Solution numbers at the end of each question and answer refer to the Avant! internal bug system solution number.

Analysis

Q: How do I make .TRAN and .TRAN SWEEP results agree?

A: If you get different results from transient simulations than when doing a parameterized transient sweep (and you are using transmission line elements), you should explicitly set the option RISETIME. For example:

```
.OPTION risetime=<rise time of fastest component>
```

Solution: 28

* *

Q: Why do I get "*Internal timestep too small*" when I use UIC in the .TRAN statement?

A: If the UIC is used in the .TRAN statement, the DC operating point analysis is bypassed and a transient analysis is started using node voltages specified in an .IC statement.

During a simulation, DC operating point analysis is an important step. Most of the DC analysis algorithms, control options, initilizations, and convergence also apply to transient analysis. As a result, although a transient analysis might provide a convergent DC solution, the transient analysis itself can still fail to converge if UIC is used. In transient analysis, the error message "internal timestep too small" indicates that the circuit failed to converge. The convergence failure might be due to stated initial conditions that are not close enough to the actual DC operating point values.

Therefore, the best solution is to comment out the UIC from the .TRAN statement and rerun the simulation. At that point, the DC operating point analysis is performed and the correct values are used in the transient analysis.

Solution: 383

* *

Q: Is there a way to change the seed that pseudo-randomly generates the initial conditions in the Monte Carlo simulation?

A: Yes, you can set .OPTION seed=<*value*> where *value* is between 1 and 259200. This changes the random number generator starting seed.

Solution: 408

* *

Documentation

Q: Whom should I contact regarding Star-Hspice manuals and other documents?

A: To purchase the latest *Star-Hspice* documents, call Ana Wagem at (510) 413-8383 or email:

```
ana_wageman@avanticorp.com
```

Solution: 41

* *

Q: Are the demo files in the *Star-Hspice User Guide* available on-line?

A: No, they are not on-line. However, they are in the directory where you install the Star-Hspice and AvanWaves executables:

1. From the directory where you install Star-Hspice (for example: `/hspice`), point to:

   `/hspice/99.4/demo/`

2. Two sub-directories appear:
   - `/hspice` directory, which contains several sub-directories, such as alge, apps, bench, bisect, bjt, etc. In each sub-directory, there are several demo netlists relative to the title of that sub-directory.
   - `/awaves` directory contains a sub-directory, named “tutorial” which contains AC, DC, and TRAN sub-directories. Each sub-directory contains one demo file relative to its title.

Solution: 376

* *

Q: How do I get a copy of the *Star-Hspice* documentation?

A: First, check the Star-Hspice installation directory for a docs folder. If the manual isn't there, check the Star-Hspice CD for the documents.

- For UNIX, they should be in a *200*.*_hspice_docs.tar.gz* file, or something similar.
- For Windows, rerun the installation. One of the options during the installation is to choose what items to install (Star-Hspice, AvanWaves, and documentation).

If none of these options work, send an email to:

```
hspice_nw@avanticorp.com
```

An ftp account can be set up to download the electronic documents.

For hardcopies, email:

```
hspice_nw@avanticorp.com
```

for the necessary contact information.

Solution: 460

* *

Q: Are any demo files included with the PC version of the software?

A: Yes, during installation you have the option to install demo files.

Solution: 503

* *

Environment Variables

Q: What does the environment variable META_QUEUE do?

A: If META_QUEUE is enabled, it allows licenses to be queued. For example, a customer has five Star-Hspice floating licenses; if all five licenses are checked out and META_QUEUE is enabled, then the next job submitted waits in queue until a license is available. If META_QUEUE is not enabled in this situation, you simply get an error message that no licenses are available.

Note: If you have more than one Star-Hspice token (INCREMENT line) and the version dates are different, only the first token in your license file is queued. FLEXlm queues for the first INCREMENT line that satisfies the request. If you have two INCREMENT lines with different versions, there are two license pools created on the server. When you issue the queuing request, the server tries to satisfy the request, and, if not possible, queues for the first INCREMENT line that would satisfy the request. Once it queues for that particular INCREMENT line, the server waits for it to become free. It does not continually look for any other line that might satisfy this request. This is normal operation for FLEXlm.

Solution: 367

* *

Error Messages

Q: I am getting “inconsistent license key” error messages when starting up lmgrd. Why? What should I do?

A: This happens when the encryption code that avantd calculates is different from the encryption key used to generate the license key. The following pieces of information from the license key are used: the host id of the server, the software expiration date, the increment name, and the software version.

If the user changes any of these fields, you will get an inconsistent license key error message. This can also happen when you are using different versions of avantd and lmgrd; however, it is very unlikely that this will cause a problem.

Avant! does not recommend using lmreread to start a new license. Shut the license server down using:

```
lmdown -c /path/to/license/file
```

and then start the new license server in the normal manner.

Solution: 134

* *

Q: I am getting a TCL error: “`Cannot find config.status`” when running the $installdir/bin/config script to configure Star-Hspice/AvanWaves. Why?

A: Your installation is probably fine. No matter what products you choose to install, the configuration script tries to display the config.status file. Unfortunately, not all installation options generate a “config.status” file.

You can verify that the configuration was successful by reading the automatically generated logfile.This is placed in /tmp/avanti_####/ config.log, where #### are numbers specific to the date and time of the installation.

Solution: 137

* *

Q: I am getting the following error message:

```
** error**: iob_loads2:9612: Number of input voltage
sources in IOB = 0
```

What does this mean?

A: Certain connections in IBIS elements must be connected to ideal sources (e.g., output buffers must have their input node connected to an ideal source). For more information on which nodes must be connected, see Chapter 7, "Using IBIS Models", in the *True-Hspice Device Models Reference Manual*.

Solution: 145

* *

Q: Why is the simulate button grayed out in HSPUI, even if a simulation is not being performed?

A: This is usually caused by a computer crash during a simulation. To ungray the simulate button, type in a command prompt:

```
"del %tmp%\job.run"
```

Alternatively, you search for the file `job.run`, and delete it.

Solution: 293

* *

Q: Why do I get the error "inductor/voltage source loop"?

A: This is normally caused by:

1. A voltage source with the inductor connected to the same point.
2. A voltage source with an inductor connected directly across its nodes.
3. Two or more inductors connected in a loop with nothing to limit the current.

The best way to solve this type of error is to connect a small series resistance (1 nano ohm or smaller usually works).

Solution: 464

* *

Q: I get the error: “Time: command terminated abnormally.” Why?

A: This means Star-Hspice stops unexpectedly, which can be due to a number of problems (such as convergence problems or a numerical overflow). Look at the *.lis and *.st0 files for clues to what happened. Also, try to find out if the simulation concluded using a different version of Star-Hspice or on other platforms. If it did, it could be a bug. Otherwise, it is most likely a problem with the designs or model cards.

The Unix TIME utility outputs the message, “Time: command terminated abnormally”, not Star-Hspice itself. The TIME utility reports various system use information on the processes it monitors. When a monitored process stops unexpectedly, TIME issues this warning.

Solution: 520

* *

Q: Why am I getting the following error in my license.log file when I try to start the avantd daemon:

```
ATTENTION ... “avantd” vendor daemon CAN NOT co-exist
with monarc lockfile * error (-1) with “monarc”, program
aborted... Please correct problem and restart daemons
```

A: This is caused by an earlier Avanti vendor daemon. Look in the /usr/tmp directory for a file named either lockmonarcd or lock.monarcd. Delete the lockfile and restart lmgrd. The avantd daemon can now start.

Solution: 528

* *

Q: Why do I get a Star-Hspice and AvanWaves “configure unsuccessful” error?

A: If $installdir is previously set, when you install a new version of Star-Hspice the path for the configuration will not be correct (it will try to install over your previous installation). This causes a “configure unsuccessful” error. To fix this, set the correct path to your install directory in the configure utility.

Solution: 561

* *

Q: Why do I get the error message: "**error**: system file descriptors may be used up"?

A: This is an error reported by your operating system. It happens when Star-Hspice tries to open more files than is permitted. If you have a large number of .ALTER statements or if you read in lots of files (.LIB or .INCLUDE), you will have to raise your system's limit.

With sh/ksh/bash, you can use "ulimit" to display or change the limits for the current shell and all the processes you launch from this shell. Use:

```
$ ulimit -n
```

to display the current limit. For example, on my system, I get 64; this means that any process I launch from this shell cannot open more than 64 files at once. To raise this limit to 256, for instance, type:

```
$ ulimit -n 256
```

There is a "hard" limit that you cannot exceed (although the root can change the hard limit). To see what the hard limit is on your system, type:

```
$ ulimit -Hn
```

On csh/tcsh, use the "limit" built-in do this (type `limit` to see a list of limits). For example, on my system:

```
cputime         unlimited
filesize        unlimited
datasize        2097148 kbytes
stacksize       8192 kbytes
coredumpsize    unlimited
descriptors     64
memorysize      unlimited
```

To increase the number of open files (file descriptors) to 256, type:

```
$ limit descriptors 256
```

To view the hard limits, type:

```
$ limit -h
```

Solution: 613

* *

Q: Why do I get this message: "**error** reference 0:nch not found" even when NCH appears in my library?

A: Star-Hspice has a feature called "automatic model selection," which is probably used in your library. You can identify models that use this feature if the model name is followed by a period and a suffix, as in:

```
.MODEL nch.1 nmos ...
```

It is likely that you have multiple .model nch.* statements that define the model's parameters for different sizes of the transistor. These models use LMIN, LMAX, WMIN and WMAX to determine which model applies to a given transistor dimensions.

For example, if model nch.1 includes LMIN=0.15u LMAX=0.25u WMIN=0.15u WMAX=1.0u, then the model is only applied to transistors with lengths between 0.15um and 0.25um and widths between 0.15um and 1.0um.

If you are getting a "reference not found" error, then your transistor dimensions (L and W) do not fall in the ranges specified by any of the models in your library.

Solution: 642

* *

Input

Q: Why do I get a message: “input file contains no data”?

A: This occurs with either no .END or no return after the .END statement in the netlist. Ensure the netlist has an .END as the last statement, and do not forget to insert a return.

Solution: 44

* *

Q: Why does an I/O Buffer used as an input have a much smaller input current than an input buffer?

A: The power clamp and ground clamp nodes need to be included in the calling card when using an I/O buffer as an input buffer.

Solution: 455

* *

Installation Issues

Q: Whom should I contact regarding Star-Hspice installation?

A: To contact the Star-Hspice technical support team, submit a question to:

```
hspice_nw@avanticorp.com
```

To ensure Avant! has a thorough understanding of your question, provide all pertinent details.

Solution: 41

* *

Q: I am having trouble installing Star-Hspice on my Windows NT machine. Do you have any suggestions?

A: A common solution is setting Windows NT environment variables ($installdir, LM_LICENSE_FILE and PATH) in *ControlPanel->System*; these are not set by default.

Solution: 56

* *

Q: I recently upgraded or installed a newer (post-97.4) Star-Hspice version; however, now I cannot simulate out of Analog Artist from Cadence. It is issuing an error message about not being able to find “permit.hsp.” Or perhaps Analog Artist is spawning the incorrect version of Star-Hspice. What do I do?

A: Most installations of Cadence Composer/Analog Artist create a script that is run to bring up the framework. Within this script, they are setting the $installdir environment variable. Edit this script and point the $installdir variable to the new Star-Hspice installation directory (the directory that contains the “bin” directory, which in turn contains the Star-Hspice and AvanWaves scripts).

Solution: 136

* *

Q: I am having problems installing my new node-locked Star-Hspice license. The error I am receiving is: “invalid hostid”. Star-Hspice is reporting a hostid of 0, but I have the dongle installed! What should I do?

A: Node-locked versions of Star-Hspice on the PC require the use of a parallel port dongle. Under WinNT, you must install a driver to be able to use the dongle. The installation program is “setupx86.exe” in the %installdir% directory. Under Win98/95, it is not necessary to run the “setupx86.exe” sentinel driver.

Solution: 146

* *

Licensing/Access Issues

Q: Whom should I contact regarding Star-Hspice license files?

A: To request a new license file, send email to:

```
license@avanticorp.com
```

To ensure Avant! has a thorough understanding of your question, provide all pertinent details.

Solution: 41

* *

Q: My older FLEX license does not work with my new Star-Hspice. What should I do?

A: FLEXlm v5.0 needs a 4-digit year in the expiration date; you should get a new license.

Solution: 53

* *

Q: How can I run older Star-Hspice versions without having to source the `cshrc.meta` file of the older version?

A: You have to edit the "versions" file in the latest version of Star-Hspice, for example, `..installdir../99.2/bin/versions`, and include the paths of the older versions in the `versions` file. Once that is set, type:

```
hspice
```

at the command line. After you enter the input and output file names, you are prompted for a version to run. Select the number (1,2,3...etc) that represents the version you would like to run.

On PC's, modify the `...installdir\99.2\versions` file in the same manner and then you can select the version you would like to run by modifying the last line on the HSPUI window.

Solution: 57

* *

Q: I am getting “inconsistent license key” error messages when starting up lmgrd.

A: See this solution in Error Messages on page A-7.

Solution: 134

* *

Q: I am getting a TCL error: “Cannot find config.status” when running the $installdir/bin/config script to configure Star-Hspice/AvanWaves.

A: See this solution in Error Messages on page A-7.

Solution: 137

* *

Q: How can I reserve licenses for a particular user, group, or host?

A: Flexlm allows you to reserve licenses using a configuration file. The configuration file is a plain ASCII file containing comments and statements. Lines starting with a # are considered comments. You may create groups of users using the statement:

```
GROUP <groupname> <user> [...]
```

You may reserve a specific license to either a user, a group, a host or a display using one of:

```
RESERVE <reserved_num> <feature> USER <username>
RESERVE <reserved_num> <feature> GROUP <groupname>
RESERVE <reserved_num> <feature> HOST <hostname>
RESERVE <reserved_num> <feature> DISPLAY <dispname>
```

Reserving a license for a specific HOST does not prevent users from setting their DISPLAY environment variable to their local DISPLAY and running the tool. In fact, reserving a license for a HOST is very similar to using a node-locked license.

Reserving a license for a DISPLAY ensures that the DISPLAY environment variable is set to what you have specified.

Solution: 654

* *

Limitations

Q: What are the known limitations of Metaencryption?

A: The known limitations are:

- The filenames of the encrypted files must have the same name as the subcircuit definition they contain (with a .inc extension).
- The encrypted files must *not* be used with a .LIB or .INCLUDE in the netlist. Instead, they must be included through the automatic search path (.OPTION search=...) by referencing the subckt name in a subckt instance.
- No more than 80 characters are allowed in a line; semi colons (;) are not allowed.
- For Star-Hspice 1998.4 and earlier versions, dummy models should be inserted before using the LEVEL 49 model.
- For every .PROTECT you should have an .UNPROTECT statement.
- It is always good to have .PROTECT and .UNPROTECT in the following order:

Recommended	Not Recommended
.SUBCKT	.SUBCKT
.PROTECT	.PROTECT
~~	~~~
.UNPROTECT	.ENDS
.ENDS	.UNPROTECT

Solution: 40

* *

Miscellaneous

Q: Should I include the gate edge of the source and drain when specifying the PD and PS (periphery of the drain and source respectively, used for calculating parasitic diodes)?

A: For ACM=0 or 2, you do include the gate edge of the source and drain when calculating PS and PD. For ACM=3, you do not include the source or drain edges when specifying PS and PD. PS and PD are not used for ACM=1.

Solution: 133

* *

Q: Is there a difference between parameters defined on the .SUBCKT line and those defined in the subcircuit body?

A: Parameters defined on the .subckt line are in the same namespace as parameters defined within the body of the subcircuit.

The following two subcircuits are equivalent:

```
.SUBCKT sub1 in out l=10u w=5u
.ENDS
```

Or:

```
.SUBCKT sub2 in out
.PARAM l=10u w=5u
.ENDS
```

Note: Both forms of the subcircuit accept the parameters “l” and “w” on element instance lines.

Solution: 139

* *

Q: How can I simplify repetitive complex measurements?

A: Encapsulate the functionality of your measurements/probes in subcircuits. Subcircuits do not have to contain elements. They can contain complex measurement routines to make measurements less tedious. If you make judicious choices for the instance names, parsing the output becomes relatively easy. For example, calculate AC transfer functions for arbitrary nodes in your netlist:

```
.PARAM mag(a,b) = 'sqrt(a*a+b*b)'
.SUBCKT Xfer in out
.PROBE AC xfer
par('mag(vr(in)*vr(out)+vi(in)*vi(out),
+ vi(in)*vr(out)-vi(out)*vr(in))/
+ mag(vr(out),vi(out))')
*xfer = mag(in/out) (note the voltages at in and
+ out are complex)
.ENDS
```

Solution: 142

* *

Models

Q: Why doesn't `.OPTION SCALE` work in my simulation?

A: Check to see if your MOSFET and model card are wrapped inside a .SUBCKT call. For example:

```
.SUBCKT pmos d g s b w=1 l=1 m=1 mp d g s b p w=w l=l
+ m=m
.MODEL p pmos LEVEL=.......
.ENDS
```

If so, check the model parameters for any dependencies on width or length. These parameters, in this case, are not scaled. .OPTION SCALE affects width and length only when passed into the model card as physical length and width. The model parameter uses width and length before scaling.

Solution: 209

* *

Q: What is the process to get proprietary MOSFET models?

A: The following MOSFET model level numbers are vendor proprietary and require a special license to use:

AMD	AMD	LEVEL 32
Dallas Semi	DALLAS	LEVEL 19
Hitachi	HIT	LEVEL 43
IBM	IBM	LEVEL 48
Lucent	ATT	LEVEL 45
Motorola	MOT	LEVEL 31
	MOTSSIM	LEVEL 29
National	NAT	LEVEL 33
SGS Thomson	SGSTHOM	LEVEL 46

Sharp	SHARP	LEVEL 36
TI	TI	LEVEL 37
		LEVEL 41
Vitesse	VITESSE	LEVEL 6

As of July 12, 2000, these are the contact people:

AMD	Barbara Vervenne	(512) 602-5783
Dallas Semi	Roy Hensley	(214) 450-3858
Hitachi	Len Mizrah	(408) 456-2081
IBM	Cal Bittner	(902) 769-6528
Lucent	Roberta Kulp	(610) 712-2614
Motorola	Ralph Sokel	(512) 933-5264
National	Chen Young Tao	(408) 721-5665
SGS Thomson	Claude Frank	33 476 92 63 47 (Tel) 33 476 08 96 52 (Fax)
Sharp	Sigenobu Kiyoi	81 7 4365 4273
TI	Jeff Brunson	(972) 480-2481
Vitesse	Aubrey Grey	(805) 388-7517

For a customer to receive a license from Avant! (the vendors do not cut the licenses), he/she must contact the vendor and request that an authorization letter is submitted to Avant! Once Avant! receives the letter, a license file can be processed.

Solution: 263

* *

Q: Why doesn't BJT LV5 - FT agree with net analysis results?

A: FT is a simple estimate for a cut-off frequency of the internal transistor:

```
ft=abs(gm-go)/(2*pi*max( 1.0d-18, C_be+C_bc+C_bx ) )
```

The parameters for the equation are from the equivalent circuit for the BJT (see Chapter 4, "Using BJT Models", in the *True-Hspice Device Models Reference Manual*).

FT does not include parasitic elements (Rb, Rc, Re for example), and does not include the external circuit. It just gives a simple estimate for cut-off frequency of the internal transistor and it cannot replace results of actual simulation.

For another discussion on this, see "Semiconductor Device Modeling with SPICE" by Massobrio and Antognetti, McGraw Hill, 1993, page 99.

Solution: 560

* *

Q: What is the importance of the following excerpt from the attached .lis file:

```
*****************************************************
**warning BSIM3v3 VERSION parameter does not appear in
**the model card!!!
**Failure to explicitly set VERSION may result in
**inaccurate results!
*****************************************************
```

A: Avant! implemented the BSIM3v3 models released from UC Berkeley in the forms of LEVEL 49 and LEVEL 53. Since implementation, UC Berkeley has released patches and Avant! has implemented them as 3.1 (default for 98.2), 3.2 (default for 98.2 and 98.4) and 3.22 (default for 99.2, 99.4 and 2000.2).

If your models were developed for the VERSION = 3.1 release and this parameter is not present in your models/libraries, then Star-Hspice defaults to 3.22 (in the Star-Hspice 2000.2 release), which might give you unexpected results.

Solution: 562

* *

MS Windows/PC Issues

Q: My Star-Hspice license will not work on my PC. Why?

A: Problems often occur on PC's with truncated lines in license files; repair the license or contact the Star-Hspice technical support team by submitting a question to:

```
hspice_nw@avanticorp.com
```

To ensure Avant! has a thorough understanding of your question, provide all pertinent details.

Solution: 55

* *

Q: I am having trouble installing Star-Hspice on my Windows NT machine.

A: Windows NT environment variables (installdir, LM_LICENSE_FILE and PATH) need to be set in *ControlPanel->System*; these are not set by default.

Solution: 56

* *

Q: How can I run older Star-Hspice versions without having to source the “cshrc.meta” file of the older version?

A: See this solution in Licensing/Access Issues on page A-15.

Solution: 57

* *

Q: Why do I get the message, “Bad date found in the running machine,” when starting Star-Hspice on Win NT?

A: To get a license to check out, make sure the dates on the Windows NT machine are consistent with the current time (i.e., no future creation or modification times). The files with bad dates must be changed.

However, Windows NT does not have a utility that can easily change file dates. To change these dates requires a command such as the UNIX *touch* command, which Windows NT does not have. A third-party software package `does` have this *touch* command. “UnixDos Toolkit 4.3d” has numerous UNIX commands for the Window NT environment. You can download it from software sites. After installing this utility, you can *touch* the files and change their dates to the current date. However *touch* does not work with folders. To update the folder’s dates, just copy the folder’s contents to a new folder (example: folder xxx to folder xxx1), delete the old folder, and rename the folder to the original name (back to xxx).

Star-Hspice works when you change files and directories in the WINNT folder to the current date. Other files, not in the WINNT folder, with bad (future) dates, do not cause any problems. For example, MS Office files and folders can have bad dates, but Star-Hspice still runs.

Solution: 341

* *

Q: In Windows, the path to license.dat is correct, but still Star-Hspice cannot find it. What do I do?

A: If you use Windows Explorer, it will, by default, not show the extension of a file. So if, by mistake, your license file has an additional extension besides .dat, then you will see it in the Explorer view, but Star-Hspice cannot find it.

To remedy the problem, open a “DOS prompt” and go to the directory where you have your license file and type (without quotes) “dir”. Find your license file, and see if it has an additional extension such as .txt or .flx. To fix it, type “rename license.dat.??? license.dat”. This should rename the file to `license.dat`.

Solution: 439

* *

Q: How do I load network files into AvanWaves on a PC?

A: To view network drives in AvanWaves, you must know the path to the drive. To access a file on server in your home directory, use the path:

```
\\Server\home\user
```

Type this in the "open" dialog box in AvanWaves. AvanWaves remembers this path throughout the current session.

Note: You must log into the server, and have read permission.

Solution: 448

* *

Q: How do you run Star-Hspice from a command prompt in Windows?

A: Use the form:

```
hspice -i input.sp -o output.lis
```

This opens the Star-Hspice output window (shows the status of the .sw0 file) and runs the input file. When Star-Hspice finishes, it closes the window and updates the command prompt.

Solution: 457

* *

Q: How do I get a copy of the Star-Hspice manuals (for Windows)?

A: See this solution in Documentation on page A-4.

Solution: 460

* *

Q: If I downloaded Star-Hspice from Windows NT, will it work on all Windows operating systems?

A: Yes, Star-Hspice can be run on Windows 95/98 and NT 3.5 and 4.0.

Solution: 502

* *

Q: Are there any demo files included with the PC version of the software?

A: Yes, during installation you have the option to install demos.

Solution: 503

* *

Netlist/Options

Q: How do I determine the actual values used for simulation?

A: To determine the actual options used for a simulation, include the following line in your netlist:

```
.OPTION OPTS
```

This forces Star-Hspice to print out the options and corresponding values used in the simulation to stdout. Emphasis is redirected to a .lis file.

Note: If you are looking for the analysis-specific options, you must perform the type of analysis you are interested in within your netlist.

If you would like to find the actual element parameters used in the simulation, then you can use the element templates as described in Element Template Output on page 8-36.

Solution: 143

* *

Q: My netlists, which contain both IBIS elements and .ALTER statements, do not always complete successfully or the answers are odd.

A: This was corrected in Star-Hspice v1999.4 release and after.

Solution: 147

* *

Q: Can I use more than one interface option in my netlist?

A: No, the second interface option overwrites the first. If you first set:

```
.OPTION CSDF
```

and later in the netlist set:

```
.OPTION POST
```

Star-Hspice first writes all the outputs in Viewlogic format and later the outputs are overwritten in AvanWaves format. You cannot see the output in Viewlogic but you can see them in AvanWaves.

Solution: 208

* *

Q: Why doesn't .OPTION SCALE work in my simulation?

A: See this solution in Models on page A-20.

Solution: 209

* *

Q: Why does my extracted netlist fail in Star-Hspice?

A: To use an extracted netlist, do a global search for the colon symbol (:). Replace this character globally with some unique pattern that is not likely to be in the netlist already. "_c_" is a possible replacement for the colon symbol.

Solution: 342

* *

Q: What is the magnitude difference between the NORM and UNORM options when using windowing in the .FFT statement?

A: If you use the option UNORM instead of the default NORM setting in your .FFT statement, you will see a difference in the signal magnitude. The reason for this is because of the equations that govern the different windows. The equations are given in Table 19-1 on page 19-3.

The difference in magnitude is due to "Coherent Power Gain." The definition for Coherent Power Gain is that it is a measure of the reduction in signal power due to the window function suppressing a coherent signal at the end of the measurement interval. This value is really given for each window in the equation column (it is not obvious in all cases). Here is a table to show what the coherent power gain is for each case.

Window	Coherent Power Gain	Coherent Power Gain (dB)
Rectangular	1.0	0.0
Bartlett	0.5	-6.021
Hanning	0.5	-6.021
Hamming	0.54	-5.352
Blackman	0.42323	-7.468
Blackman - Harris	0.35875	-8.904
Gaussian (a=3)	0.72	-2.853
Kaiser - Bessel (a=3)	0.4	-7.959

Solution: 351

* *

Q: How can I find out what components are connected to the same node?

A: The option you probably need is: .OPTION NODE. This option creates a node cross-reference table that can be printed. The table lists each node and all the elements connected to it. The terminal is indicated by a code, separated from the element name with a colon (:). The codes are:

+	Diode anode
-	Diode cathode
B	BJT base
B	MOSFET or JFET bulk
C	BJT collector
D	MOSFET or JFET drain
E	BJT emitter
G	MOSFET or JFET gate
S	BJT substrate or MOSFET, JFET source

Solution: 360

* *

Q: In PETL, what is the significance of GRIDFACTOR?

A: The program uses the predefined number of segments according to the specified accuracy mode. If you want higher accuracy than you can specify using the accuracy mode, you can double or triple the number of pre-defined segments by setting GRIDFACTOR to 2 or 3.

Solution: 392

* *

Q: Which integration method (TRAP or GEAR) is more accurate?

A: The TRAP (trapezoidal) method is faster and more accurate than GEAR. The GEAR method is used to remove the oscillation that can occur due to the TRAP algorithm. Circuits that are nonconvergent with TRAP will often converge with GEAR. The GEAR method is suitable for the circuits that are inductive in nature, such as switching regulators.

Solution: 409

* *

Q: Why doesn't losstangent have any affect on my W Element?

A: For the dielectric loss matrix to be computed and included in the simulation, ".FSOPTIONS computegd" has to be set in the SPICE deck.

Solution: 505

* *

Q: I have 64 .ALTER statements but only get 36 .mtx files. Why?

A: By default, Star-Hspice only generates 36 unique output files for .tr*, .mt* etc. After 36 .ALTER statements, Star-Hspice starts from the beginning and writes over *.mt0 on up. To change this you can use either:

```
.OPTION ALT999
```

or:

```
.OPTION ALT9999
```

The first option lets Star-Hspice write 999 files before going back to the 0th file and alt9999 lets you write 9999 output files.

Solution: 511

* *

Q: What does the MU option do? The manual just says it is a trapezoidal integration parameter.

A: The MU option acts as both a flag and a parameter. The equations for trapezoidal integration and backward-Euler integration are very similar to each other. The difference between these two is the parameter MU. By setting MU to 0.5 (the default), Star-Hspice uses trapezoidal integration. If MU=0, then Star-Hspice uses backward-Euler integration. If you set MU to a value between 0 and 0.5, then you get a blend of both methods.

Here is the order of the three integration methods available in Star-Hspice (trapezoidal, backward-Euler, Gear) from most accurate to most stable:

Trapezoidal	(most accurate)
	.
Backward-Euler	.
	.
Gear	(most stable)

Solution: 556

* *

Output

Q: How can I reduce the size of output files?

A: By adding .OPTION PROBE to the netlist, only those nodes explicitly referenced with .PROBE, .PRINT, .PLOT, or .GRAPH have output values displayed in one of the output files.

Solution: 43

* *

Q: The output of my simulation is not what I expected at all. Also, the solution shows instability or extreme sensitivity to parameter or conductance values.

A: Try .OPTION PIVOT=1 (or .OPTION PIVOT=11). The default pivoting algorithm in Star-Hspice does not pick a pivot element. This can lead to instability when there are large conduction ratios (typically when small conductances exist).

Solution: 135

* *

Q: I would like .GRAPH to write only to a file, and not send the job to the printer automatically.

A: You can accomplish this by editing your $installdir/meta.cfg. Comment out all the blocks that refer to “PLOTSETUP” and “PRTDEFAULT.” Then enter:

```
PRTDEFAULT = PS
PLOTSETUP PS PSCRIPT
PLOTFILE = /tmp/hspice.ps
END
```

Solution: 138

* *

Q: Can I use more than one interface option in my netlist (and how does this affect my output)?

A: See this solution in Netlist/Options on page A-27.

Solution: 208

* *

Q: Can PRINTDATA in PETL only have the YES and NO option? When I set PRINTDATA=YES, it did not create RLGC file.

A: If RLGCFILE is specified, it prints to that file; otherwise, it prints to the standard error output, which can be redirected with the standard output using >& in UNIX. Specify the .PRINTDATA=YES in .FSOPTIONS and RLGCFILE in .MODEL cards.

Solution: 390

* *

W Element/Field Solver

Q: How do I define an RLGC for a W Element without R and G?

A: Since the R and G matrix elements are optional, they can be left empty or 0 when defining RLGC parameters in W Element.

Solution: 458

* *

Q: Why is the field solver not giving me accurate results?

A: To receive more accurate results when using the field solver, use Star-Hspice 1999.4 or later. There have been many improvements to the W Element and the field solver.

Solution: 459

* *

Q: Why doesn't losstangent have any affect on my W Element?

A: See this solution in Netlist/Options on page A-27.

Solution: 505

* *

Waveform Viewing

Q: How can one view a waveform on an old waveform viewer (e.g., HSPLOT) or using third-party tools (such as Viewdraw)?

A: Add the following card to SPICE deck:

```
.OPTION POST_VERSION = 9007
```

Solution: 42

* *

Q: I am performing analysis at different operating temperatures. However, my waveform/measurements are not changing when I change the temperature of the simulation. What is going on?

A: Most elements' temperature coefficients default to zero. You must specify non-zero coefficients to derate components in simulation. Do this either on the element instance line or in the .MODEL card for the elements.

Solution: 155

* *

Q: Can I use more than one interface option in my netlist (how does this affect my waveform viewing)?

A: See this solution in Netlist/Options on page A-27.

Solution: 208

* *

Q: How do I create eye diagrams with Star-Hspice?

A: Here is an example of an eye diagram circuit in the Star-Hspice distribution:

```
<installdir>/demo/awaves/demo/eyediag*
```

As a quick reference, an eye diagram simply imitates the behavior of an oscilloscope (the waveform is displayed starting from the left end of the window and moving to the right at a given rate). When the electron beam reaches the right end of the window, it is moved quickly to the left end of the window where another "cycle" can be displayed. This type of display is quite useful in observing cyclic patterns that vary slightly from one cycle to another.

To get these results in Star-Hspice, you need to include these two statements in your netlist:

```
.PARAM width=5ns phase=0ns
.PROBE TRAN TIME2=par('TIME+phase-int((TIME+phase)/
+ width)*width')
```

This creates a new TIME2 output variable. You may adjust the width of the window (in ns) with the parameter "width". You can also use the *phase* parameter to adjust the phase of the waveform. To get these results in AvanWaves:

- Open the resulting .tr# file
- In the "Results Browser" window, under "Types:" select "Params"
- Under "Curves:" select "time2"
- Under "Current X-Axis," click on "Apply"
- Under "Types:" again, select "Voltages," "Currents" or any other type of output variable you want to display
- Double click on the desired signals you want to display
- In the viewing window, click on the right mouse button; select "Monotonic Plot ..."

Solution: 611

* *

Avant!

Appendix B
Interfaces For Design Environments

Star-Hspice simulation is supplemented by the following design interface products:

- AvanLink to Cadence Composer™ and Cadence Analog Artist™ interfaces Star-Hspice with Cadence Design Systems' Design Framework II to support data interchange and cross-probing with Analog Artist and Composer.
- AvanLink to Mentor Design Architect™ interfaces Star-Hspice with Mentor Graphics' Falcon Framework to support cross-probing with Design Viewpoint Editor and data interchange with Design Architect (DA) and Design Viewpoint Editor (DVE).

Overviews of these interface products are presented in the following sections:

AvanLink to Cadence Composer and Analog Artist

AvanLink to Cadence Composer and Analog Artist provides a netlister, a symbol library, cross-probing, and backannotation capabilities. AvanLink is an Avant! interface product that links the Star-Hspice circuit simulator with the following products from Cadence:

- Composer, Versions 4.2.1a, 4.2.2, 4.3.2, and 4.3.3
- Analog Artist, Versions 4.2.1a, 4.2.2, 4.3.2, and 4.3.3

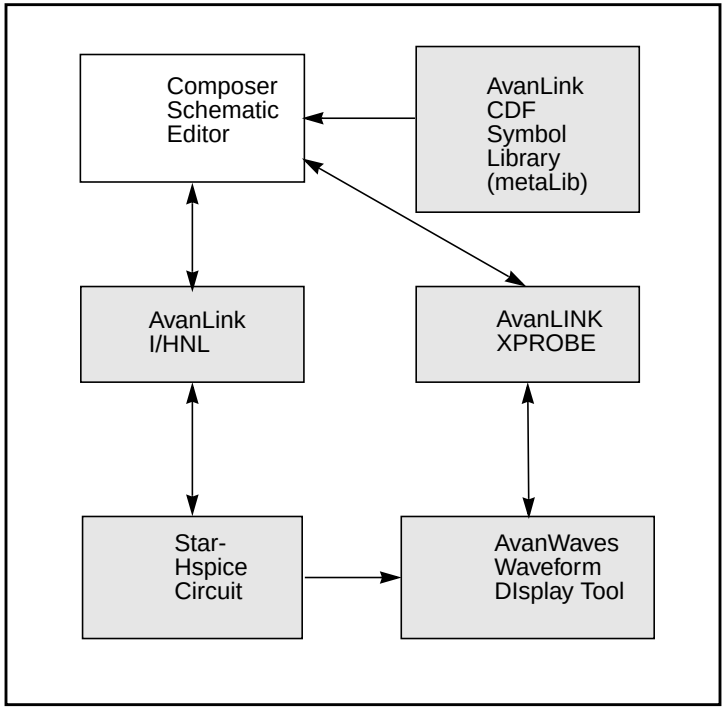
Features

AvanLink has the following functionality:

- Provides hierarchical Netlisting (HNL), with incremental capability (IHNL), through the CadenceLink netlister and the Cadence simulation environment
- Provides Avant!'s *metaLib* class-based library to support all of the Star-Hspice elements. The *metaLib* library uses Cadence's standard Component Description Format (CDF).
- Supports cross-probing for both voltage and current from both Composer and Analog Artist schematics to Avant!'s AvanWaves waveform display tool
- Supports hierarchical parameter passing and algebraic function operations
- Preserves Star-Hspice hierarchical path names through a transparent net name mapping function
- Supports the Parameter Storage Format (PSF) output program for displaying waveforms with Analog Artist
- Provides an automatic library translation program for the Cadence *sample* and *analogLib* libraries and user-defined libraries
- Supports backannotation of operating points and element parameters back to the schematic
- Supports graphical setup of simulation initial conditions

Figure B-1 shows how AvanLink fits into the design process.

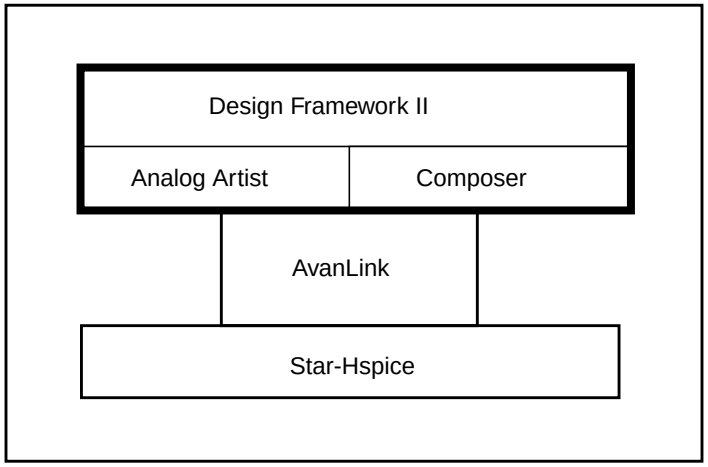
Figure B-1: AvanLink Architecture for Design Framework II with Composer



Environment

AvanLink provides an integrated environment that allows configuration of design, simulation, and display tools to support the chosen design flow. Develop the design using Composer, simulate it using Star-Hspice, and then view it using AvanWaves. Access AvanLink by selecting options on menus, or by using specific keys assigned to these options. Figure B-2 shows the operating environment for AvanLink.

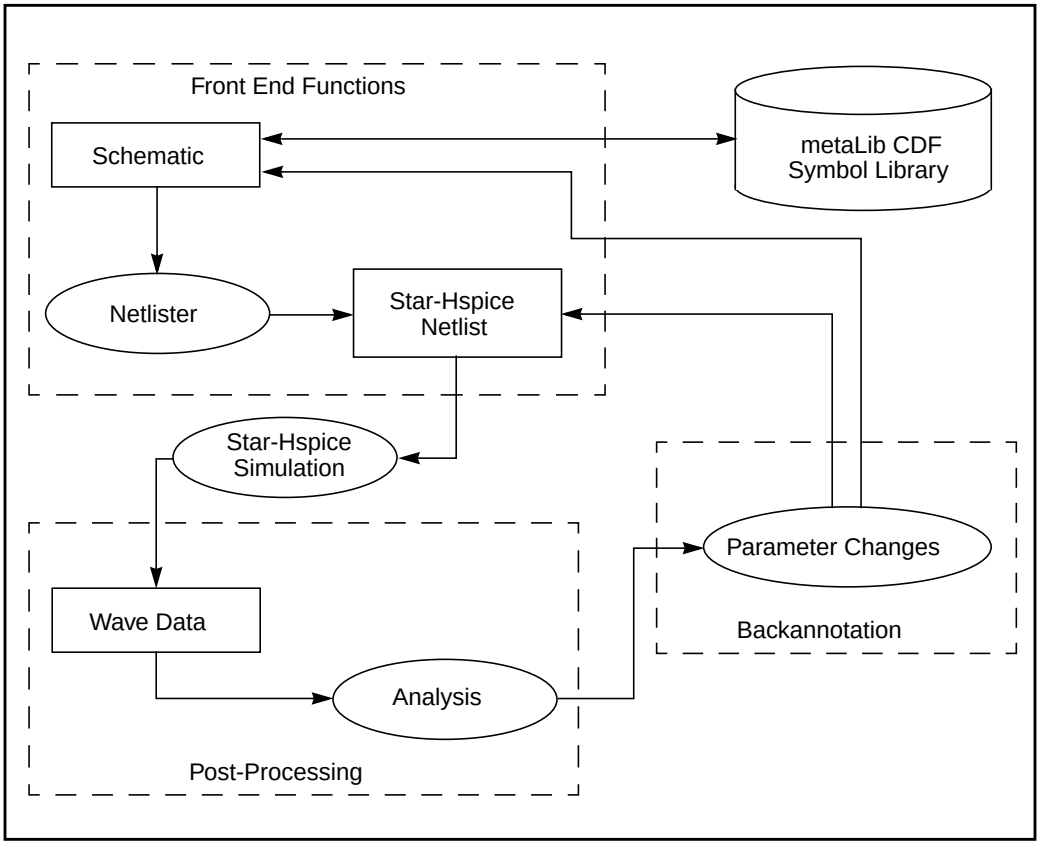
Figure B-2: Star-Hspice AvanLink in a Cadence Composer Environment



AvanLink Design Flow

Figure B-3 is an overview of the process involved in creating a design with AvanLink.

Figure B-3: AvanLink Design Flow

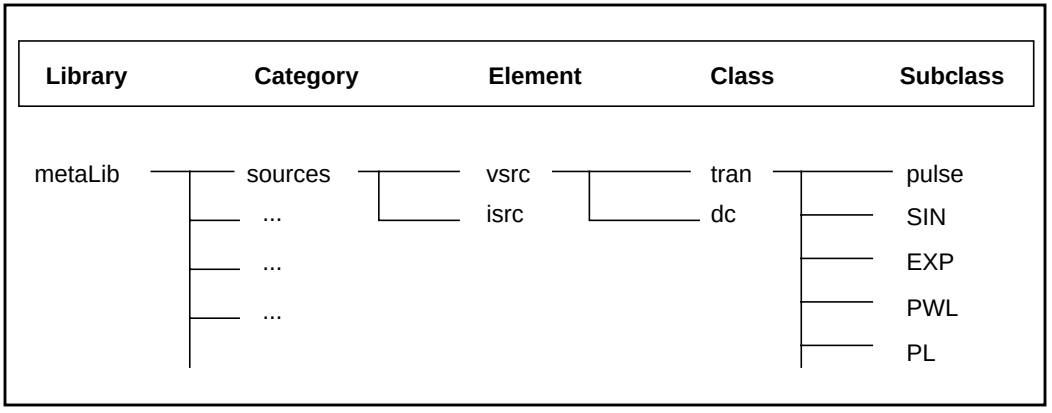


Schematic Entry and Library Operations

AvanLink helps you design your schematic using the *metaLib* Component Description Format (CDF) library. This is a class-based library that includes all the Star-Hspice elements. Each element is described by a set of parameters that are organized in classes and subclasses. An example of a *metaLib* class-based CDF library structure for a source element is shown in Figure B-4.

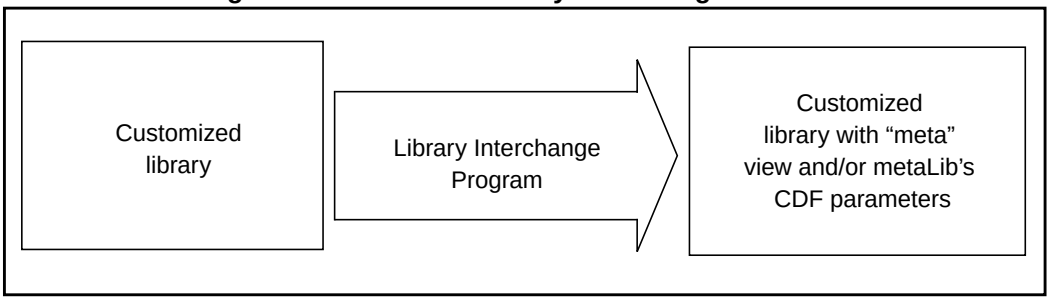
As each element in the design is instantiated, a form is opened, automatically displaying the CDF parameters for that element. Modifications can be made to these parameters as necessary. AvanLink guarantees that Star-Hspice generates an accurate and syntactically correct netlist for simulation.

Figure B-4: Example metaLib Library Structure for a Source Element



Use the library interchange program supplied by AvanLink to convert personal or customized symbol libraries to incorporate *metaLib* CDF parameters. Figure B-5 diagrams this procedure.

Figure B-5: AvanLink Library Interchange Function



Generating a Netlist

AvanLink uses the Cadence Open Simulation System (OSS) to integrate Star-Hspice into the Cadence Simulation Environment (SE). The AvanLink netlister includes customized functions, in addition to those in the OSS Hierarchical Netlister, in order to provide the following features:

- Preservation of the pin names in the design
- Maintenance of the original design signal names
- Preservation of design hierarchy
- Time savings when dealing with large designs that require few modifications

Simulation

The simulation is run in a user-designated simulation directory. The Cadence simulation environment initializes this directory and copies the basic Star-Hspice control files into it. You can modify these files to add personal Star-Hspice commands as desired. All Star-Hspice analysis types, including advanced analyses such as Optimization and Monte Carlo, can be specified with the control file.

Waveform Display

Analyze and display output waveforms using the AvanWaves program. The cross-probing feature allows probing a net in the schematic window and viewing its signal in the AvanWaves display window. Cross-probing is supported for signals generated by transient, DC, or AC analysis. The cross-probing feature also allows pins to be probed for branch currents.

AvanLink also generates waveform format (PSF) for displaying in the Analog Artist environment and supports cross-probing and backannotation of the Analog Artist display.

AvanLink for Design Architect

AvanLink for Design Architect is an Avant! interface product that links the Star-Hspice circuit simulator with the following products from Mentor Graphics:

- Design Architect
- Design Viewpoint Editor (DVE)

AvanLink-DA provides a netlister, a symbol library, and cross-probing and backannotation capabilities.

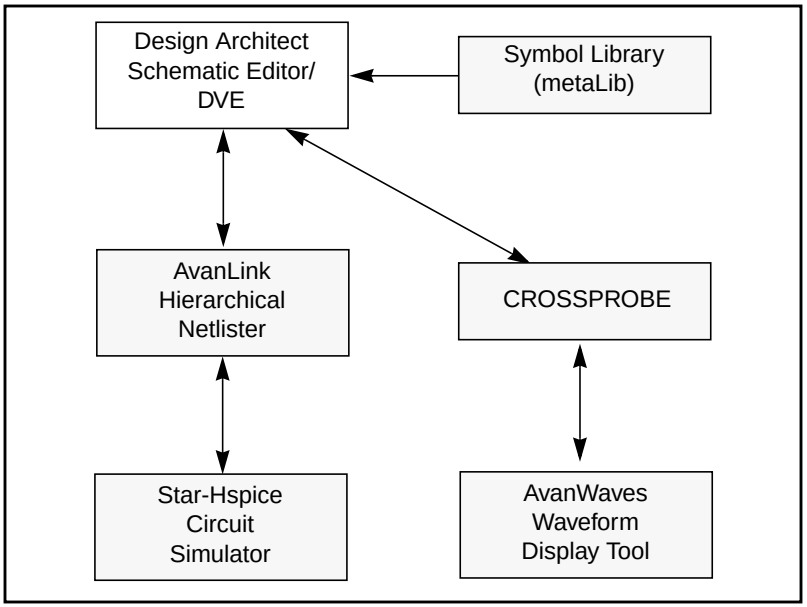
Features

AvanLink-DA features include the following:

- Hierarchical netlisting, provided through the AvanLink-DA netlister and simulation environment
- Avant!'s *metaLib* class-based library, supporting all of the Star-Hspice elements, with advanced Star-Hspice properties and element syntax
- Cross-probing from Design Viewpoint Editor schematics to Avant!'s waveform display tool, AvanWaves
- Hierarchical parameter passing and algebraic function operations
- Facility to preserve Star-Hspice hierarchical path names through a transparent net name mapping function
- Utilities for migrating customers' component libraries to AvanLink-DA
- Backannotation of operating point voltages back to the Mentor Graphics Design Viewpoint Editor
- Ability to set up and probe terminal current values
- Ability to specify initial condition (IC) and nodeset values
- Support for element parameter backannotation onto the schematic within DVE
- Support for Star-Hspice legacy libraries

Figure B-6 shows how AvanLink-DA fits into the design process.

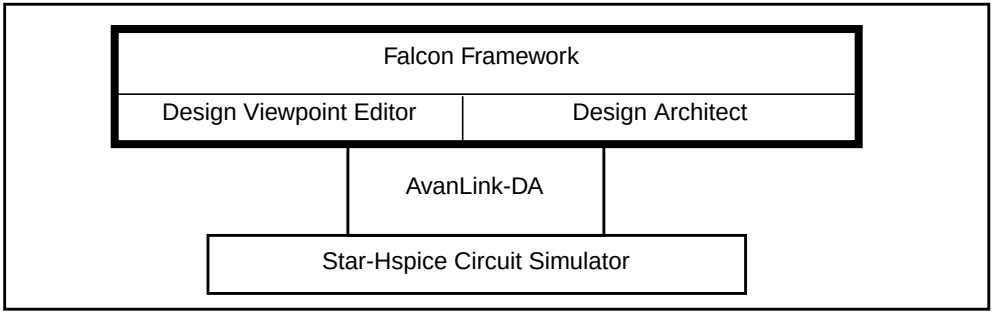
Figure B-6: AvanLink for Falcon Framework with Design Architect



Environment

AvanLink-DA provides an integrated environment for circuit design, simulation, and waveform display. Develop a design using the Design Architect, simulate it using Star-Hspice, and then view the results using AvanWaves. Access AvanLink-DA by selecting options on menus or by clicking buttons on the palette in the Mentor Graphics Design Architect window corresponding to these options. Figure B-7 shows the operating environments for AvanLink-DA.

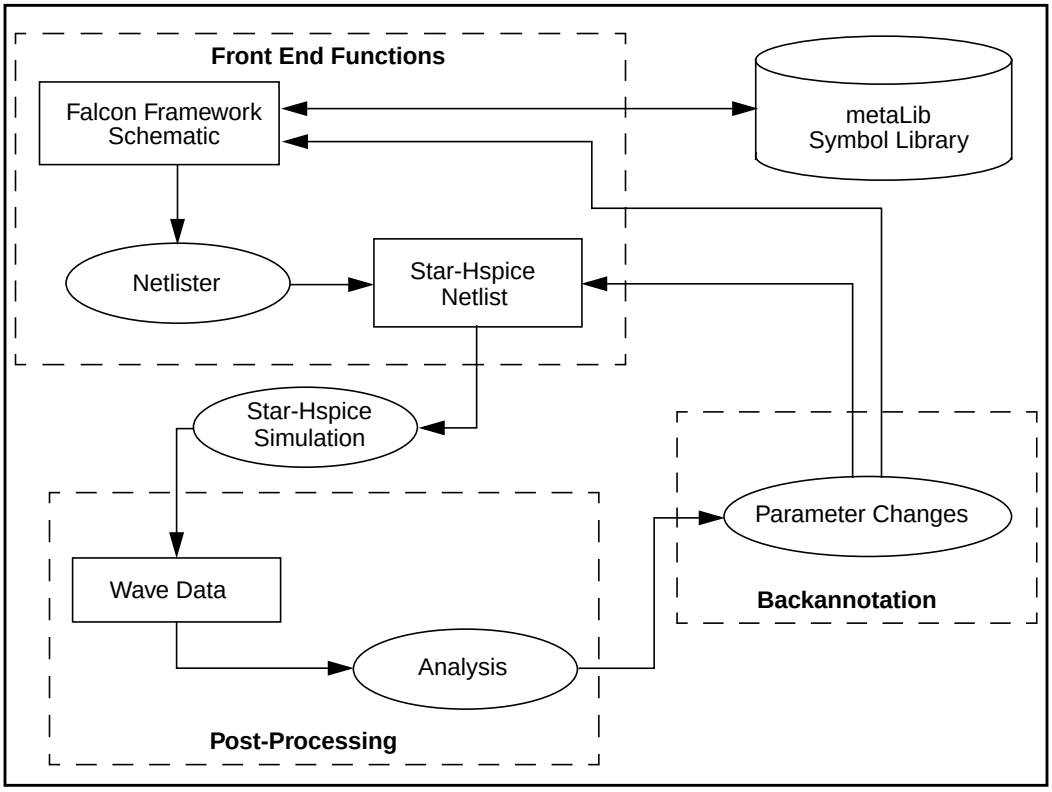
Figure B-7: AvanLink-DA in a Falcon Framework Environment



AvanLink-DA Design Flow

Figure B-8 provides an overview of the processes involved in creating and simulating a design with AvanLink-DA.

Figure B-8: Design Flow for AvanLink for Design Architect



Schematic Entry and Library Operations

AvanLink-DA helps schematic design using *metaLib*. Avant!'s *metaLib* is a class-based library that includes all of the Star-Hspice elements. Each element is described by a set of parameters that are organized in classes and subclasses. An example of a *metaLib* class-based library structure for a voltage source element is shown in Figure B-9.

After each design element is instantiated, you can display and edit the element's parameters. Modify these parameters as necessary. AvanLink-DA guarantees that Star-Hspice generates an accurate and syntactically correct netlist for simulation.

The library migration program supplied by AvanLink-DA to convert personal or customized symbol libraries to incorporate *metaLib* attributes. Figure B-10 diagrams this procedure.

Figure B-9: Example metaLib Library Structure for a Source Element

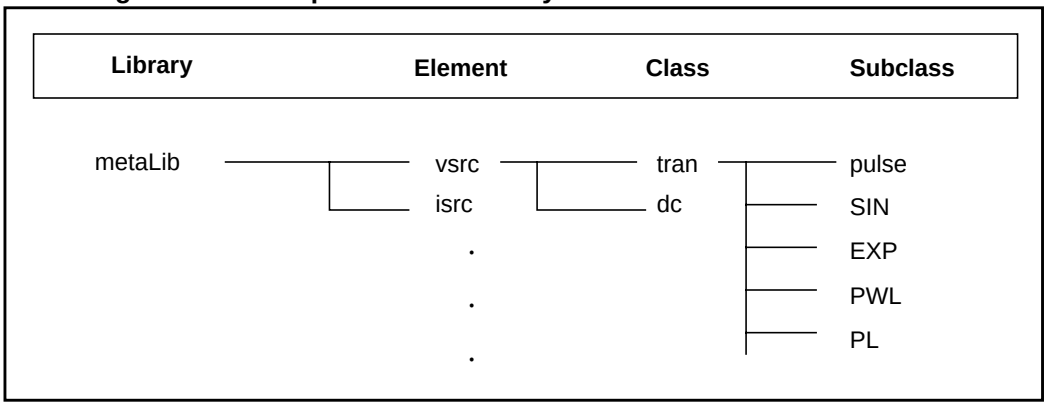
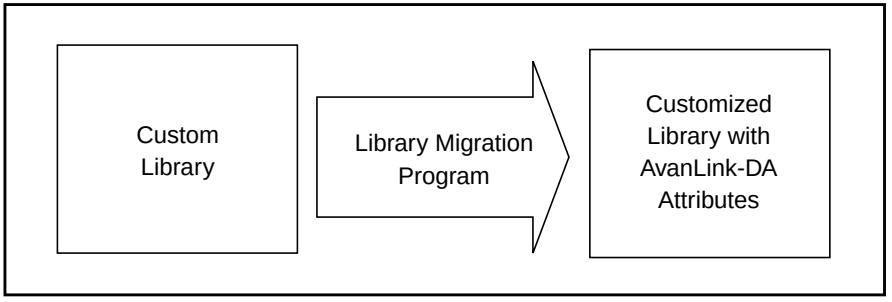


Figure B-10: AvanLink-DA Library Migration Function



Generating a Netlist

The AvanLink-DA Netlister provides the following features:

- Creation of a complete hierarchical Star-Hspice netlist
- Preservation of the original design pin names
- Maintenance of the original design signal names
- Preservation of the design hierarchy

Simulation

Run the simulation in a self-designated simulation run directory. The AvanLink-DA simulation environment initializes this directory and copies the basic Star-Hspice control files into it. Modify these files to add personal Star-Hspice commands as desired. Specify all Star-Hspice analysis types, including advanced analyses such as Optimization and Monte Carlo, using the control file.

Waveform Display

Display output waveforms using the waveform display tool, AvanWaves. The cross-probing feature allows you to probe on a net in the schematic window inside DVE and view its signal in the AvanWaves display window. Cross-probing is supported for signals generated by transient, DC, or AC analysis. The cross-probing feature also allows pins to be probed for branch currents.

Viewlogic Links

Users of Viewlogic have access to Star-Hspice through the Viewlogic Powerview framework. These tools (provided by Viewlogic) include a netlister, a Star-Hspice option CSDF for waveform display in viewtrace, and cross-probing. Additionally, Viewlogic provides a Star-Hspice/Madssim mixed-signal simulation solution.

Avant!

Appendix C
Performing Library Encryption

Library encryption allows you to distribute your own proprietary Star-Hspice custom models, parameters, and circuits to other people without revealing your company's sensitive information. Recipients of an encrypted library can run simulations that use your libraries, but Star-Hspice does not print encrypted parameters, encrypted circuit netlists, or internal node voltages. Your library user sees the devices and circuits as "black boxes," which provide terminal functions only.

Use the library encryption scheme primarily to distribute circuit blocks with embedded transistor models, such as ASIC library cells and I/O buffers. Star-Hspice uses sub-circuit calls to read encrypted information. To distribute device libraries only, create a unique sub-circuit file for each device.

This chapter describes Avant!'s Library Encryptor and how to use it to protect your intellectual property. The following topics are covered in the chapter:

Understanding Library Encryption

The library encryption algorithm is based on that of a five-rotor Enigma machine. The encryption process allows the user to specify which portions of sub-circuits are encrypted. The libraries are encrypted using a key value that Star-Hspice reconstructs for decryption.

Controlling the Encryption Process

To control the beginning and end of the encryption process, insert .PROTECT and .UNPROTECT statements around text to be encrypted in an Star-Hspice sub-circuit. The encryption process produces an ASCII text file in which all text that follows .PROTECT and precedes .UNPROTECT is encrypted.

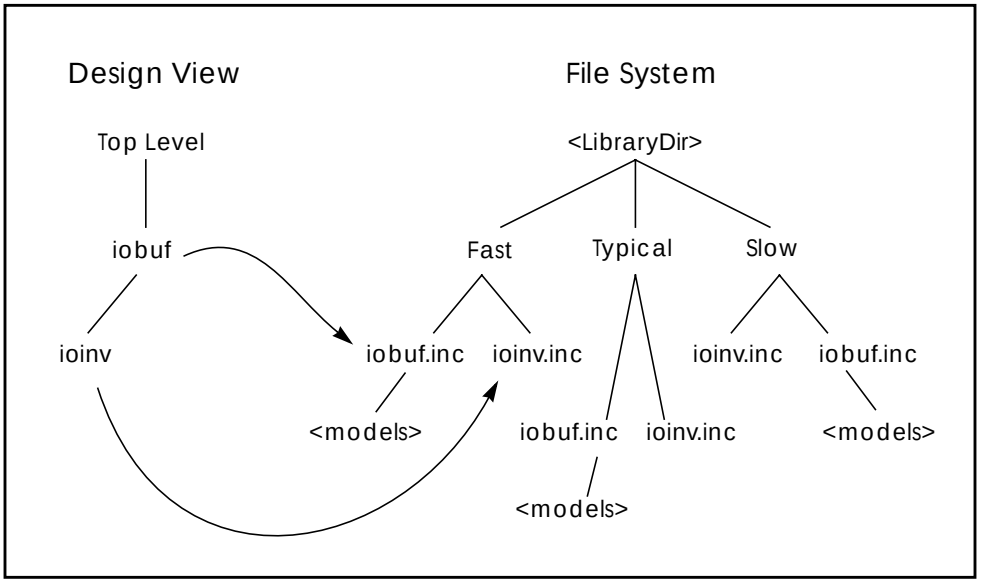
Note: The Star-Hspice .PROTECT and .UNPROTECT statements often are abbreviated to .PROT and .UNPROT, respectively. Either form may be used in Star-Hspice input files.

Library Structure

The requirements for encrypted libraries of sub-circuits are the same as the requirements for regular sub-circuit libraries. sub-circuit library structure requirements are described in Subcircuit Library Structure on page 3-56. Refer to an encrypted sub-circuit by using its sub-circuit name in a sub-circuit element line of the Star-Hspice netlist.

The following example provides the description of an encrypted I/O buffer library sub-circuit. This sub-circuit is constructed of several sub-circuits and model statements that you need to protect with encryption. Figure C-1 shows the organization of sub-circuits and models in libraries used in this example.

Figure C-1: Encrypted Library Structure



The following input file fragment from the main circuit level selects the *Fast* library and 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 file *<LibraryDir>/Fast/iobuf.inc* contains:

```
.Subckt iobuf Pin1 Pin2 Cload=1pF
*
* iobuf.inc - model 2001 improved iobuf
*
.PROTECT
cPin1 Pin1 0 1pF     $ Users can't change this!
x1 Pin1 Pin2 ioinv   $ Italics here means encrypted
.Model pMod pmos Level=28 Vto=...  $ <FastModels>
.Model nMod nmos Level=28 Vto=...  $ <FastModels>
.UNPROTECT
cPin2 Pin2 0 Cload   $ give you some control
.Ends
```

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

```
.Subckt ioinv Pin1 Pin2
.PROTECT
mp1 Vcc Pin1 Pin2 Vcc pMod  ..   $ Italics=Protected
mn1 Pin2                         Pin1 Gnd Gnd nMod...
$ Italics=Protected
.UNPROTECT
.Ends
```

After encryption, the basic layout of the sub-circuits is the same. However, the text between .PROTECT and .UNPROTECT statements is unreadable, except by Star-Hspice.

The protection statements also suppress printouts of encrypted model information from Star-Hspice. Only Star-Hspice knows how to decrypt the model.

Knowing the Encryption Guidelines

In general, there are no differences between using the encrypted models and using regular models. However, you *must test your sub-circuits before encryption*. You will not be able to see what has gone wrong after encryption because of the protection offered by Star-Hspice.

Use any legal Star-Hspice statement inside your sub-circuits to be encrypted. Refer to .SUBCKT or .MACRO Statement on page 3-13 for further information on sub-circuit construction. You must take care when structuring your libraries. If your library scheme requires that you change the name of a sub-circuit, you must encrypt that circuit again.

Placing .PROTECT and .UNPROTECT statements lets your customers see portions of your sub-circuits. If you protect only device model statements in your sub-circuits, your users can set device sizes or substitute different sub-circuits for lead frames, protection circuits, and so on. This requires your users to know the circuits, but it reduces the library management overhead for everyone.

Note: If you are running any version of the encryptor prior to Star-Hspice Release H93A.03, there is a bug that prevents Star-Hspice from correctly decrypting a sub-circuit if that sub-circuit contains any semicolon (;) characters, even in comments.

In the following example, the sub-circuit *badsemi.dat* is encrypted into *badsemi.inc*.

```
* Sample semicolon bug
.SubCkt BadSemi A B
.PROT
* Semicolons (;) cause problems!
r1 A B 1k
.UNPROT
.Ends
```

Star-Hspice responds with the following message:

```
**reading include file=badsemi.inc
**error**:  .ends card missing at read-in
>error   ***difficulty in reading input
```

To solve this problem, remove the semicolon from *badsemi.dat* and encrypt the file again.

Prior to the 2001.2 version, Metaencrypt cannot encrypt files with lines longer than 80 characters. Metaencrypt version 2001.2 can encrypt files with lines longer than 80 characters but cannot correctly encrypt files with lines longer than 254 characters. Avant! strongly recommends that all files to be encrypted be limited to a 254-character line length and use version 2001.2 and above to encrypt any files with lines longer than 80 characters. Star-Hspice can recognize all correctly encrypted files.

You cannot gather the individually-encrypted files into a single file or include them directly in the Star-Hspice netlist. Place them in a separate directory pointed to by the .OPTION SEARCH = <*dir*> named <*sub*>*.inc* for correct decryption by Star-Hspice.

Note: If a data line is divided into more than one line, with + linking the lines, you cannot add .prot or .unpr among the 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
.unpr
+PARAMCHK=0 WINT = 0 LINT = -2E-08
```

Installing and Running the Encryptor

This section describes how to install and run the Encryptor.

Installing the Encryptor

If Star-Hspice is already installed on your system, place the Encryptor in the directory *$installdir/bin* to install it. Add the lines that allow the Encryptor to operate to your *permit.hsp* file in the *$installdir/bin* directory.

If Star-Hspice is not installed on your system, first install Star-Hspice according to the installation guide and *Star-Hspice Release Notes* included in your Star-Hspice package, and then follow the instructions in the previous paragraph.

Note: If you are running a floating license server, you must stop and restart the server to see the changes to the permit file.

Running the Encryptor

The Encryptor requires three parameters for each sub-circuit encrypted: *<InFileName>*, *<OutFileName>*, and the key type specifier, *Freelib*. Enter the following line to encrypt a file.

```
metaencrypt -i <InfileName> -o <OutFileName> -t Freelib
```

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

To encrypt many files in a directory, use the following shell script to encrypt the files as a group. This script produces a *.inc* encrypted file for each *.dat* file in the current directory. The procedure assumes that the unencrypted files are suffixed with *.dat*.

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

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

Understanding Metaencrypt Features

This section describes new features and enhancements to the metaencrypt functionality supported in version 2001.2 and later.

New 8-Byte Key Encryption

A new 8-byte key encryption feature based on a 56-bit DES is now available in metaencrypt with versions 2001.2 and above. You can insert data into an include file and encrypt this file using 8-byte key encryption. The encrypted data is in binary format.

Syntax

```
metaencrypt -i <infileName> -o <outfileName>
+ -t <keyname>
```

Note: The keyname can be a combination of English characters and numbers and should be limited to 8 bytes.

Example

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

Encryption Structure

Star-Hspice supports, .inc encryption when using 8-byte key encryption. Insert the data to encrypt, into an include file. Then encrypt this file.

In the following example, example.dat contains data to be encrypted.

```
metaencrypt -i example.dat -o example.inc -t h78Gbvni
*example.sp*
......
.inc example.inc $ example.inc contains encrypted data
......
.end
```

Use the following rules when employing 8-byte key encryption:

- 8-byte key encryption only supports .inc encryption
- 8-byte key encryption does NOT support .lib, .load or .option search encryption in the 2001.2 version
- .prot and .unpr should NOT be included with the data while performing 8-byte key encryption
- if keyname=ddl1, ddl2, custom, freelib, then metaencrypt will user a former algorithm to perform the encryption
- if keyname is an 8-byte string (combination of characters and numbers), then metaencrypt will perform 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.

Supporting the .sp File Encryption

You can encrypt a .sp file in Star-Hspice. The data between .prot and .unpr in a .sp file can be encrypted so that Star-Hspice will recognize it.

Note: When performing the .sp encryption, the encrypted data should not contain .inc, .lib or .load to include another file.

Example

```
*sample.sp*
......
.inc sample1.inc
.prot
.... $ data to be encrypted
.... $ do not include .inc .lib .load in encrypted data
.unpr
.inc sample2.inc
......
.end
```

Supporting .lib File Encryption

Any important information can be placed into a .lib file and can be encrypted.

You can place parallel .lib statements into one library file. Each .lib can then be encrypted separately. However, .prot and .unpr must be placed between each pair of .lib and .endl.

Note: .prot and .unpr must be placed between .lib and .endl since Hspice will first find the library name in the .lib file.

Example

```
*sample.sp*
......
.lib ‘./sample.lib’ test1
.lib ‘./sample.lib’ test2
.lib ‘./sample.lib’ test3
......
.end

*sample.lib*
.lib test1 $ .prot , .unpr should be put between
+ .lib and .endl
.prot
...... $ data to be encrypted
.unpr
...... $ data not to be encrypted
.end1 test1

.lib test2 $ .prot , .unpr should be put between
+ .lib and .end1
.prot
...... $ data to be encrypted
.unpr
.end1 test2

.lib test3
...... $ data
.end1 test3
```

Supporting .inc File Encryption

You can insert any important data into a .inc file and encrypt the file. There are no restrictions on the placement of .prot and .unpr.

Example

```
*sample.sp*
......
.inc sample.inc
......
.end
```

```
*sample.inc*
.prot $ no restriction on the placement of .prot
....$ exclude .inc, .lib .load from data to encrypt
.unpr
......
```

Supporting .load Encryption

You can encrypt a .ic file just like a .inc file.

Supporting 80+ Columns Encryption

Star-Hspice version 2001.2 and later can support 1-255 column encryption.

Statements Not Supported

- Embedded .lib encryption as this will cause confusion in decryption
- Embedded .inc encryption as this will cause confusion in decryption
- Data should not contain .inc, .lib or .load statements to include another .inc, .lib or .ic file

Additional Recommendations for Encryption

Using .option search for encrypting models and sub-circuits is not recommended. This method was supported using the old metaencryption functionality. sub-circuits and model libraries can be encrypted directly using the .inc and .lib encryption method.

Complete Encryption Structure Example

The following complete example illustrates the encryption structure.

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

```
file mm.inc:
.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%.ahT3VQ
H0[I:[
.ENDL
```

Avant!

Appendix D
Full Simulation Examples

The examples in this chapter display the basic text and post-processor output for a sample input netlist. The first example uses AvanWaves to view results and the second example uses Cosmos-Scope.

This chapter includes the following sections:

Full Simulation Example with AvanWaves

Input Netlist and Circuit

The input netlist for the linear CMOS amplifier is shown below. The individual sections of the netlist are indicated using comment lines. Refer to Simulation Input and Controls on page 3-1 for information about the individual commands.

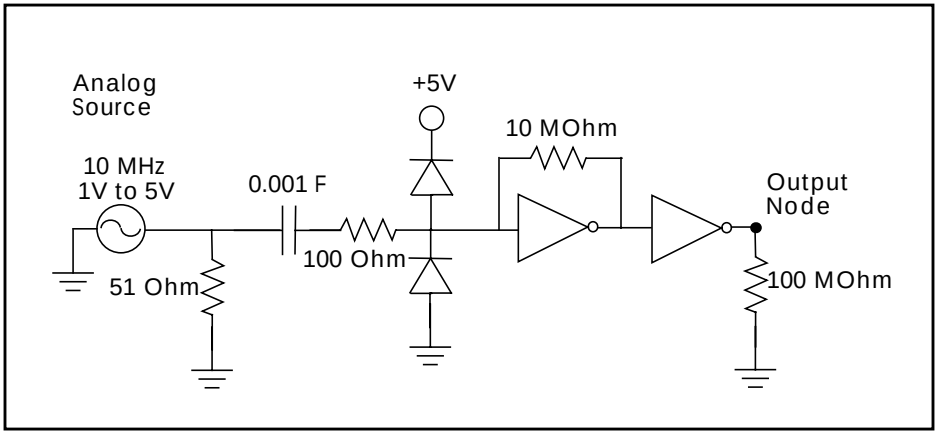
```
* Example HSPICE netlist using a linear CMOS amplifier

* netlist options
.option post probe brief nomod

* defined parameters
.param analog_voltage=1.0

* global definitions
.global vdd

* source statements
Vinput in gnd SIN ( 0.0v analog_voltage 10x )
Vsupply vdd gnd DC=5.0v

* circuit statements
Rinterm in gnd 51
Cincap in infilt 0.001
Rdamp infilt clamp 100
Dlow gnd clamp diode_mod
Dhigh clamp vdd diode_mod
Xinv1 clamp inv1out inverter
Rpull clamp inv1out 1x
Xinv2 inv1out inv2out inverter
Routterm inv2out gnd 100x

* subcircuit definitions
.subckt inverter in out
Mpmos out in vdd vdd pmos_mod l=1u w=6u
Mnmos out in gnd gnd nmos_mod l=1u w=2u
.ends
```

```
* model definitions
.model pmos_mod pmos level=3
.model nmos_mod nmos level=3
.model diode_mod d

* analysis specifications
.TRAN 10n 1u sweep analog_voltage lin 5 1.0 5.0

* output specifications
.probe TRAN v(in) v(clamp) v(inv1out) v(inv2out)
+ i(dlow)
.measure TRAN falltime TRIG v(inv2out) VAL=4.5v FALL=1
+                      TARG V(inv2out) VAL=0.5v FALL=1

.end
```

The following is the circuit diagram for the linear CMOS amplifier that is described in the circuit portion of the netlist. The two sources shown in the diagram are also in the netlist.

Note: The inverter symbols in the circuit diagram are constructed from two complementary MOSFET elements. Also, the diode and MOSFET models in the netlist do not have non-default parameter values, except to specify Level 3 MOSFET models (empirical model).

Figure D-1: Circuit Diagram for Linear CMOS Inverter



Execution and Output Files

The following section displays the various output files from a Star-Hspice simulation of the amplifier shown in the previous section. The simulation was executed by entering:

```
hspice example.sp > example.lis
```

where the input netlist was named *example.sp* and the output listing was named *example.lis*. The following output files were created with a brief explanation of their content.

example.ic	initial conditions for the circuit
example.lis	text simulation output listing
example.mt0	post-processor output for MEASURE statements
example.pa0	subcircuit path table
example.st0	run-time statistics
example.tr0	post-processor output for transient analysis

The following subsections show the text files in their entirety for the amplifier simulation performed using Star-Hspice 1998.4 on a Sun workstation. The two post-processor output files cannot be shown because they are in binary format.

Example.ic

```
* “simulator” “HSPICE”
* “version” “98.4 (981215) ”
* “format” “HSP”
* “rundate” “13:58:43  01/08/1999”
* “netlist” “example.sp ”
* “runtitle” “* example hspice netlist using a linear
* cmos amplifier ”
* time =    0.
* temperature =   25.0000
```

```
*** BEGIN: Saved Operating Point ***
.option
+ gmindc=   1.0000p
.nodeset
+ clamp =   2.6200
+ in =   0.
+ infilt =   2.6200
+ inv1out =   2.6200
+ inv2out =   2.6199
+ vdd =   5.0000
***   END: Saved Operating Point ***
```

Example.lis

```
Using: /net/sleepy/l0/group/hspice/98.4beta/sol4/
+ hspice

****** Star-HSPICE --    98.4 (981215)  13:58:43
******  01/08/1999 solaris
Copyright (C) 1985-1997 by Avant! Corporation.
Unpublished-rights reserved under US copyright laws.
This program is protected by law and is subject to the
terms and conditions of the license agreement found in:

      /afs/rtp.avanticorp.com/product/hspice/current/
      license.txt

Use of this program is your acceptance to be bound by this
license agreement. Star-HSPICE is the trademark of
Avant! Corporation.

Input File: example.sp

lic:
lic: FLEXlm:v5.12 USER:hspiceuser HOSTNAME:hspiceserv
+ HOSTID:8086420f PID:1459

lic: Using FLEXlm license file:
lic: /afs/rtp/product/distrib/bin/license/license.dat
lic: Checkout hspice; Encryption code: AC34CE559E01F6E05809
lic: License/Maintenance for hspice will expire on 14-apr-
+ 1999/1999.200
lic: 1(in_use)/10 FLOATING license(s) on SERVER hspiceserv
lic:
```

```
******
* example hspice netlist using a linear cmos amplifier

******

* netlist options
.option post probe brief nomod

* defined parameters
Opening plot unit= 15
file=./example.pa0

****** Star-HSPICE --    98.4 (981215) 13:58:43
******  01/08/1999 solaris ******

* example hspice netlist using a linear cmos amplifier
***** transient analysis  tnom= 25.000 temp= 25.000 *****
*** parameter analog_voltage   =    1.000E+00 ***

node    =voltage      node    =voltage      node    =voltage

+0:clamp =   2.6200 0:in     =   0.     0:infilt =   2.6200
+0:inv1out =2.6200 0:inv2out = 2.6199   0:vdd    =   5.0000

Opening plot unit= 15
file=./example.tr0

**warning** negative-mos conductance = 1:mnmos  iter= 2
vds,vgs,vbs =       2.45          2.93       0.
gm,gds,gmbs,ids=   -3.636E-05     1.744E-04  0.  1.598E-04
******

* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom= 25.000 temp= 25.000 *****

falltime= 3.9149E-08 targ= 7.1916E-08   trig= 3.2767E-08

*** Star-HSPICE -- 98.4 (981215) 13:58:43
*** 01/08/1999 solaris ***
* example hspice netlist using a linear cmos amplifier
****** transient analysis tnom= 25.000 temp= 25.000 ******

*** parameter analog_voltage   =    2.000E+00 ***

node    =voltage      node    =voltage      node    =voltage
+0:clamp = 2.6200   0:in     =   0.     0:infilt =   2.6200
+0:inv1out = 2.6200  0:inv2out = 2.6199 0:vdd    =   5.0000
```

```
******
* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom= 25.000 temp= 25.000 *****

falltime= 1.5645E-08 targ= 5.7994E-08   trig= 4.2348E-08

**** Star-HSPICE --    98.4 (981215) 13:58:43
**** 01/08/1999 solaris ****

* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom= 25.000 temp= 25.000 *****

*** parameter analog_voltage   =    3.000E+00 ***

node    =voltage      node    =voltage      node    =voltage
+0:clamp =   2.6200 0:in      =   0.     0:infilt =   2.6200
+0:inv1out = 2.6200 0:inv2out = 2.6199 0:vdd    =   5.0000

******
* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom= 25.000 temp= 25.000 *****

falltime= 1.1917E-08 targ= 5.6075E-08   trig= 4.4158E-08

****** Star-HSPICE -- 98.4 (981215) 13:58:43
******  01/08/1999 solaris ******

* example hspice netlist using a linear cmos amplifier
 ***** transient analysis tnom= 25.000 temp= 25.000 *****

*** parameter analog_voltage   =    4.000E+00 ***

node    =voltage      node    =voltage      node    =voltage
+0:clamp =   2.6200 0:in      =   0.     0:infilt =   2.6200
+0:inv1out = 2.6200 0:inv2out = 2.6199 0:vdd    =   5.0000

******

* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom= 25.000 temp= 25.000 *****

falltime= 7.5424E-09 targ= 5.3989E-08   trig= 4.6447E-08

****** Star-HSPICE -- 98.4 (981215) 13:58:43
******  01/08/1999 solaris ******

* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom= 25.000 temp= 25.000 *****
```

```
*** parameter analog_voltage   =    5.000E+00 ***

node    =voltage      node    =voltage      node    =voltage

+0:clamp =   2.6200 0:in      =   0.     0:infilt =   2.6200
+0:inv1out = 2.6200 0:inv2out = 2.6199 0:vdd     =   5.0000

******
* example hspice netlist using a linear cmos amplifier
***** transient analysis tnom= 25.000 temp= 25.000 *****

falltime= 6.1706E-09 targ= 5.3242E-08   trig= 4.7072E-08

meas_variable = falltime
mean = 16.0848n      varian = 1.802e-16
sigma = 13.4237n      avgdev =   9.2256n
max   = 39.1488n      min    =   6.1706n

***** job concluded

****** Star-HSPICE -- 98.4 (981215) 13:58:43
******  01/08/1999 solaris ******
* example hspice netlist using a linear cmos amplifier
*** job statistics summary tnom= 25.000 temp= 25.000 ***

total memory used        155 kbytes

# nodes = 8 # elements= 14
# diodes= 2 # bjts    =     0 # jfets   = 0 # mosfets = 4

analysis    time     # points   tot. iter  conv.iter
op point    0.04     1          23
transient   4.71     505        9322       2624 rev= 664
readin      0.03
errchk      0.01
setup       0.01
output      0.01

total cpu time          4.84 seconds
job started at 13:58:43 01/08/1999
job ended   at 13:58:50 01/08/1999

lic: Release hspice token(s)
HSPICE job example.sp completed.
Fri Jan 8 13:58:50 EST 1999
```

Example.pa0

```
1 xinv1.
2 xinv2.
```

Example.st0

```
***** Star-HSPICE --    98.4 (981215) 13:58:43
*****  01/08/1999 solaris

Input File: example.sp
lic:
lic: FLEXlm:v5.12 USER:hspiceuser HOSTNAME:hspiceserv
+ HOSTID:8086420f PID:1459
lic: Using FLEXlm license file:
lic: /afs/rtp/product/distrib/bin/license/license.dat
lic: Checkout hspice; Encryption code: AC34CE559E01F6E05809
lic: License/Maintenance for hspice will expire on
+ 14-apr-1999/1999.200
lic: 1(in_use)/10 FLOATING license(s) on SERVER hspiceserv
lic:

init: begin read circuit files, cpu clock= 2.21E+00
      option probe
      option nomod

init: end read circuit files, cpu clock= 2.23E+00
+ memory= 145 kb
init: begin check errors, cpu clock= 2.23E+00
init: end check errors, cpu clock= 2.24E+00 memory= 144 kb
init: begin setup matrix, pivot=    10 cpu clock= 2.24E+00

establish matrix -- done, cpu clock= 2.24E+00 memory= 146 kb
re-order matrix -- done, cpu clock= 2.24E+00 memory= 146 kb
init: end setup matrix, cpu clock= 2.25E+00 memory= 154 kb

sweep: parameter parameter1       begin, #sweeps=   5
parameter: analog_voltage =    1.00E+00
dcop: begin dcop, cpu clock= 2.25E+00
dcop: end dcop, cpu clock= 2.27E+00 memory= 154 kb
tot_iter= 11
output: ./example.mt0

sweep: tran tran1 begin, stop_t= 1.00E-06 #sweeps= 101
cpu clock=  2.28E+00
tran: time= 1.03750E-07 tot_iter= 78   conv_iter= 24
tran: time= 2.03750E-07 tot_iter= 179   conv_iter= 53
tran: time= 3.03750E-07 tot_iter= 280   conv_iter= 82
```

```
tran: time= 4.03750E-07 tot_iter= 381   conv_iter= 111
tran: time= 5.03750E-07 tot_iter= 482   conv_iter= 140
tran: time= 6.03750E-07 tot_iter= 583   conv_iter= 169
tran: time= 7.03750E-07 tot_iter= 684   conv_iter= 198
tran: time= 8.03750E-07 tot_iter= 785   conv_iter= 227
tran: time= 9.03750E-07 tot_iter= 886   conv_iter= 256
tran: time= 1.00000E-06 tot_iter= 987   conv_iter= 285

sweep: tran tran1 end, cpu clock= 2.82E+00 memory= 155 kb
parameter: analog_voltage =    2.00E+00
dcop: begin dcop, cpu clock= 2.83E+00

dcop: end dcop, cpu clock= 2.83E+00 memory= 155 kb
+ tot_iter= 14

output: ./example.mt0

sweep: tran tran2 begin, stop_t= 1.00E-06 #sweeps= 101
+ cpu clock= 2.83E+00
tran: time= 1.01016E-07 tot_iter= 186   conv_iter= 54
tran: time= 2.02642E-07 tot_iter= 338   conv_iter= 98
tran: time= 3.01763E-07 tot_iter= 495   conv_iter= 145
tran: time= 4.04254E-07 tot_iter= 668   conv_iter= 198
tran: time= 5.02594E-07 tot_iter= 841   conv_iter= 248
tran: time= 6.10102E-07 tot_iter= 983   conv_iter= 289
tran: time= 7.01850E-07 tot_iter= 1161   conv_iter= 340
tran: time= 8.01776E-07 tot_iter= 1306   conv_iter= 383
tran: time= 9.04268E-07 tot_iter= 1481   conv_iter= 436
tran: time= 1.00000E-06 tot_iter= 1654   conv_iter= 486

sweep: tran tran2 end, cpu clock= 3.71E+00 memory= 155 kb
parameter: analog_voltage =    3.00E+00
dcop: begin dcop, cpu clock= 3.71E+00

dcop: end dcop, cpu clock= 3.72E+00 memory= 155 kb
+ tot_iter= 17

output: ./example.mt0

sweep: tran tran3 begin, stop_t= 1.00E-06 #sweeps= 101
+ cpu clock= 3.72E+00
tran: time= 1.00313E-07 tot_iter= 143   conv_iter= 42
tran: time= 2.01211E-07 tot_iter= 340   conv_iter= 100
tran: time= 3.01801E-07 tot_iter= 539   conv_iter= 156
tran: time= 4.02192E-07 tot_iter= 729   conv_iter= 211
tran: time= 5.01997E-07 tot_iter= 917   conv_iter= 265
tran: time= 6.01801E-07 tot_iter= 1088   conv_iter= 314
tran: time= 7.01801E-07 tot_iter= 1221   conv_iter= 351
tran: time= 8.01801E-07 tot_iter= 1362   conv_iter= 392
```

```
tran: time= 9.02387E-07 tot_iter= 1515   conv_iter= 435
tran: time= 1.00000E-06 tot_iter= 1674   conv_iter= 479

sweep: tran tran3 end, cpu clock= 4.57E+00 memory= 155 kb
parameter: analog_voltage =    4.00E+00
dcop: begin dcop, cpu clock= 4.57E+00
output: ./example.mt0

sweep: tran tran4 begin, stop_t= 1.00E-06 #sweeps= 101
+ cpu clock= 4.58E+00

tran: time= 1.00110E-07 tot_iter= 236   conv_iter= 70
tran: time= 2.04376E-07 tot_iter= 475   conv_iter= 139
tran: time= 3.07892E-07 tot_iter= 767   conv_iter= 221
tran: time= 4.01056E-07 tot_iter= 951   conv_iter= 273
tran: time= 5.01086E-07 tot_iter= 1250   conv_iter= 353
tran: time= 6.00965E-07 tot_iter= 1541   conv_iter= 432
tran: time= 7.03668E-07 tot_iter= 1805   conv_iter= 506
tran: time= 8.01114E-07 tot_iter= 2046   conv_iter= 571
tran: time= 9.01005E-07 tot_iter= 2308   conv_iter= 640
tran: time= 1.00000E-06 tot_iter= 2528   conv_iter= 703

sweep: tran tran4 end, cpu clock= 5.83E+00 memory= 155 kb
parameter: analog_voltage =    5.00E+00
dcop: begin dcop, cpu clock= 5.83E+00

dcop: end dcop, cpu clock= 5.84E+00 memory= 155 kb
+ tot_iter= 23
output: ./example.mt0

sweep: tran tran5 begin, stop_t= 1.00E-06 #sweeps= 101
+ cpu clock= 5.84E+00

tran: time= 1.00195E-07 tot_iter= 176   conv_iter= 47
tran: time= 2.00617E-07 tot_iter= 431   conv_iter= 115
tran: time= 3.00475E-07 tot_iter= 661   conv_iter= 176
tran: time= 4.00719E-07 tot_iter= 914   conv_iter= 246
tran: time= 5.04084E-07 tot_iter= 1157   conv_iter= 311
tran: time= 6.00666E-07 tot_iter= 1347   conv_iter= 363
tran: time= 7.01830E-07 tot_iter= 1623   conv_iter= 435
tran: time= 8.02418E-07 tot_iter= 1900   conv_iter= 514
tran: time= 9.01178E-07 tot_iter= 2161   conv_iter= 585
tran: time= 1.00000E-06 tot_iter= 2410   conv_iter= 650

sweep: tran tran5 end, cpu clock= 7.03E+00 memory= 155 kb
sweep: parameter     parameter       1 end
>info:         ***** hspice job concluded
lic: Release hspice token(s)
```

Simulation Graphical Output in AvanWaves

The following plots show the six different post-processor outputs from the simulation of the example netlist, as seen in AvanWaves, the graphical waveform viewer available from Avant! These plots are postscript output from the actual data.

Figure D-2: Plot of Voltage on Node *in*

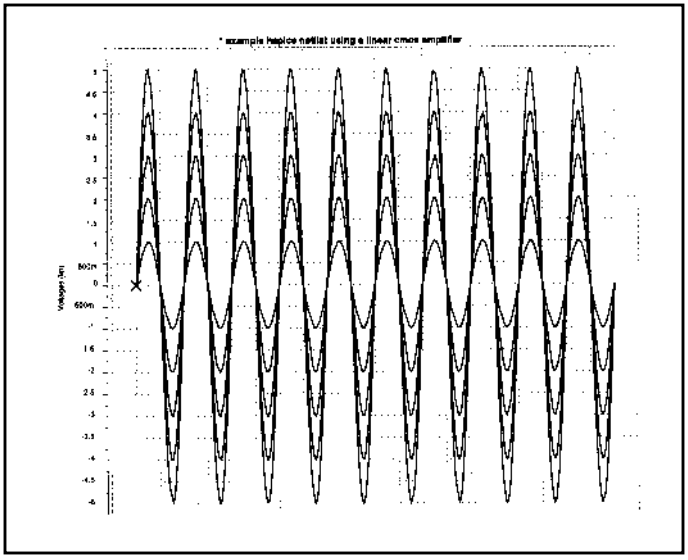


Figure D-3: Plot of Voltage on Node *clamp* vs. Time

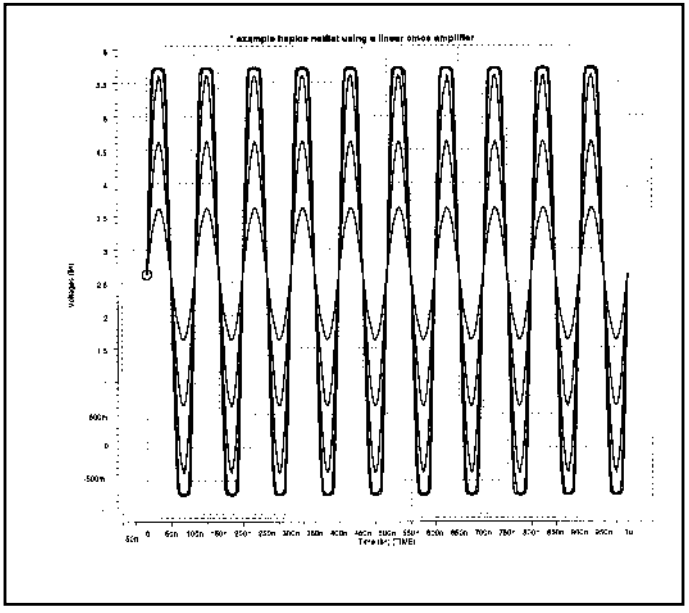


Figure D-4: Plot of Voltage on Node *inv1out* vs.Time

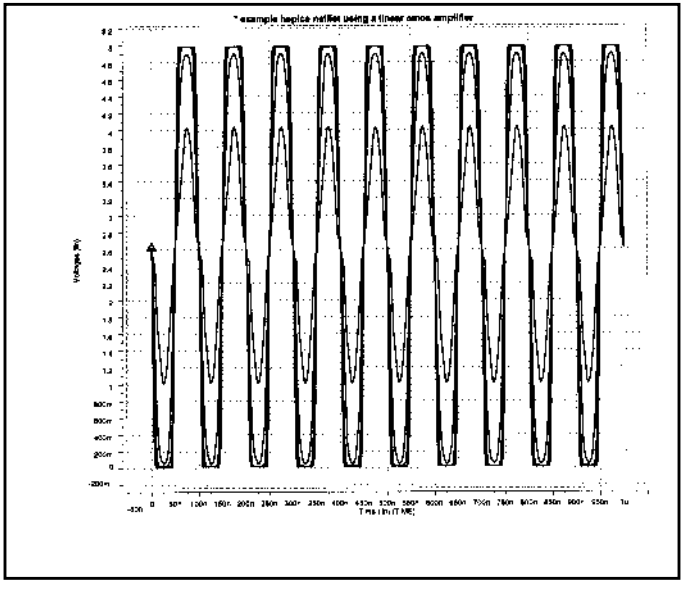


Figure D-5: Plot of Voltage on Node *inv2out* vs. Time

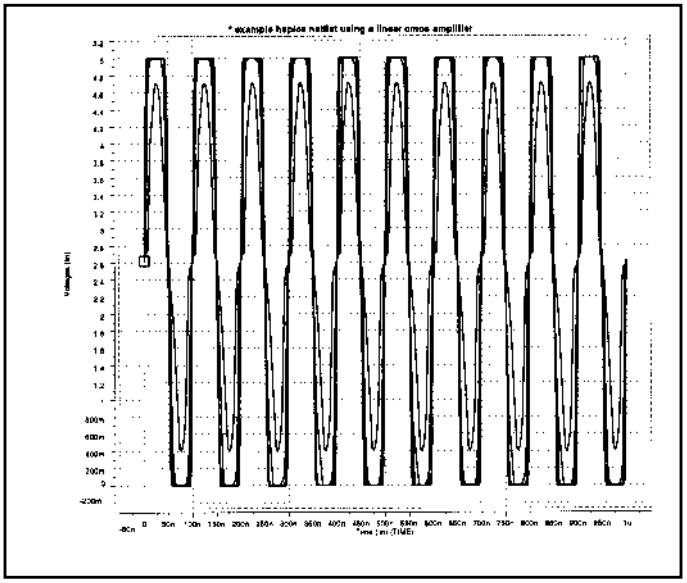


Figure D-6: Plot of Current through Diode *dlow* vs. Time

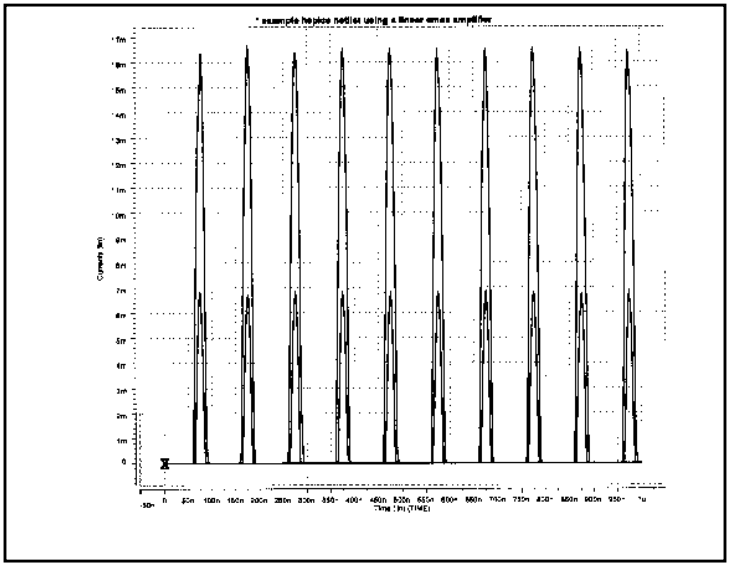
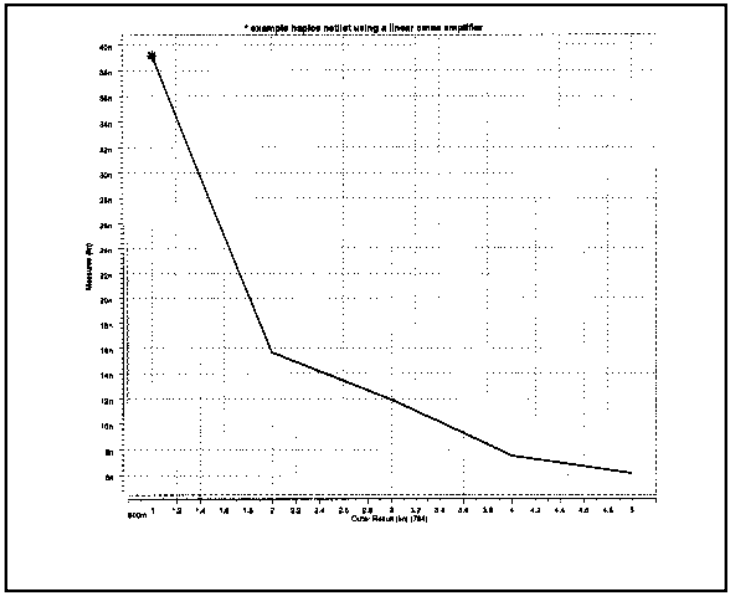


Figure D-7: Plot of User-defined Measured Variable *falltime* vs. Amplifier Input Voltage



Full Simulation Example with Cosmos-Scope

This example demonstrates the basic steps to perform simulation output and view the waveform results using Avant!'s Cosmos-Scope Waveform Viewer.

Input Netlist and Circuit

The input netlist for a BJT diff amplifier is given under Syntax. The individual sections of the netlists are indicated using comment lines. Refer to Simulation Input and Controls on page 3-1 for information about the individual commands.

Syntax

```
*file: bjtdiff.sp bjt diff amp with every analysis type
**
*# ANALYSIS: ac dc tran tf noise new four sens pz disto temp
*# OPTIONS: list node ingold=2 measdgt=5 numdgt=8
+ probe post
*# TEMPERATURE: 25
*
* netlist options
.OPTIONS list node ingold=2 measdgt=6 numdgt=8 probe post
* defined parameters
.PARAM rb1x=aunif(20k,1k,30k) rb2x=aunif(20k,1k,30k)

* analysis specifications
.TF v(5) vin
.DC vin -0.20 0.20 0.01 sweep monte=3
.AC dec 10 100k 10meghz
.NOISE v(4) vin 20
.NET v(5) vin rout=10k
.PZ v(5) vin
.DISTRO rc1 20 .9 1m 1.0
.SENS v(4)
.TRAN 5ns 200ns
.FOUR 5meg v(5) v(15)
.TEMP -55 150
```

```
* output specifications
.MEAS qa_propdly trig v(1) val=0.09 rise=1
+                targ v(5) val=6.8 rise=1
.MEAS qa_magnitude max v(5)
.MEAS qa_rmspower rms power
.MEAS qa_avgv5     avg v(5)
.MEAS ac qa_bandwidth trig at=100k targ vdb(5) val=36
+ fall=1
.MEAS ac qa_phase find vp(5) when vm(5)=52.12
.MEAS ac qa_freq when vm(5)=52.12
.PROBE dc v(4) v(5) v(14) v(15)
.PROBE ac vm(5) vp(5) vm(15) vp(15)
.PROBE ac vt(5) vt(15)
.PROBE noise onoise(m) inoise(m)
.PROBE ac z11(m) z12(m) z22(m) zin(m)
.PROBE disto hd2 hd3 sim2 dim2 dim3
.PROBE tran v(4) v(5) v(14) v(15)
.PROBE tran p(vcc) p(vee) p(vin) power

* source statements
VIN 1 0 sin(0 0.1 5meg) ac 1
VCC 8 0 12
VEE 9 0 -12

* circuit statements
q1   4 2 6 qnl
q11 14 12 16 qpl
q2   5 3 6 qnl
q21 15 13 16 qpl
rs1   1 2 1k
rs11 1 12 1k
rs2   3 0 1k
rs12 13 0 1k
rc1   4 8 10k
rc11 14 9 10k
rc2   5 8 10k
rc12 15 9 10k
q3   6 7 9 qnl
q13 16 17 8 qpl
q4   7 7 9 qnl
q14 17 17 8 qpl
rb1   7 8 rb1x
rb2 17 9 rb2x
```

```
* model definitions
.MODEL qnl npn(bf=80 rb=100 ccs=2pf tf=0.3ns tr=6ns cje=3pf
+ cjc=2pf va=50 rc=10 trb=.005 trc=.005)
.MODEL qpl pnp(bf=80 rb=100 ccs=2pf tf=0.3ns tr=6ns cje=3pf
+ cjc=2pf va=50 bulk=0 rc=10)
*
.END
```

Try using the previous example (linear CMOS amp) to draw a circuit diagram for this BJT diff amplifier. Also, specify parameter values.

Execution and Output Files

This section displays the various output files from a Star-Hspice simulation of the BJT diff amplifier example. The simulation was executed by entering:

```
hspice bjtdiff.sp > bjtdiff.lis
```

where the input file is bjtdiff.sp and the output file is bjtdiff.lis. This is a list of the output files that were created, with a brief explanation of their content.

`bjtdiff.ic`	Initial conditions for the circuit
`bjtdiff.lis`	Text simulation output listing
`bjtdiff.mt0`	Post-processor output for MEASURE statements
`bjtdiff.st0`	Run-time statistics
`bjtdiff.tr0`	Post-processor output for transient analysis
`bjtdiff.sw0`	Post-processor output for DC analysis
`bjtdiff.ac0`	Post-processor output for AC analysis
`bjtdiff.ma0`	Post-processor output for AC analysis measurements

Using Cosmos-Scope to View Star-Hspice Results

The following steps show how to use the Avant! Cosmos-Scope Waveform Viewer to view the AC, DC, and transient analysis results from the BJT diff amplifier simulation. Refer to previous examples to see a sample of .lis, .ic, and .st0 files.

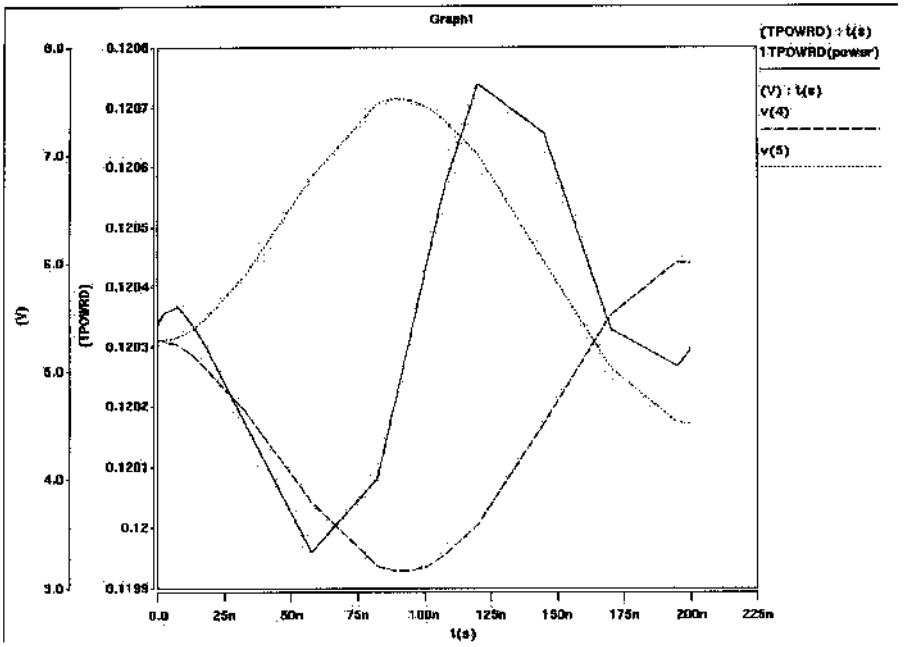
Viewing Star-Hspice Transient Analysis Waveforms

1. Invoke Cosmos-Scope.

   Scope can be invoked from a UNIX command: `% cscope`. On a Windows-NT system, Scope can be opened by choosing the **Programs** > (user_install_location)> **Cosmos-Scope**.

2. Open the Open Plotfiles dialog box: **File** > **Open** > **Plotfiles**.
3. In the Open Plotfiles dialog box, in the Files of Type fields, select the Hspice Transient (*.tr*) item.
4. You will see bjtdiff.tr0 in the menu. Click on it and click **Open**. The Signal Manager and bjtdiff Plot File windows are displayed.
5. While holding down the Ctrl key, select the v(4), v(5), and ITPOWERD(power) signals.
6. Click on **Plot** from the bjtdiff Plot File window.You will see three cascaded plots. To see three signals in one plot, right-click on the name of top-most signal to get a **Signal Menu**.
7. From the **Signal Menu**, select **Stack Region** > **Analog 0**.
8. Repeat Step 6 for next top-most signal.
9. You will see a plot similar to one below, Figure D-8.

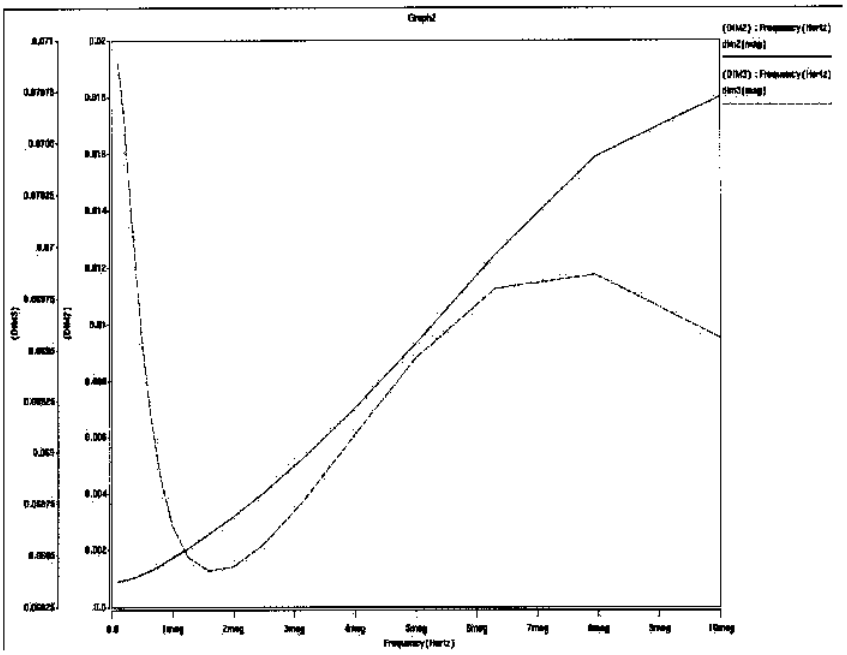
Figure D-8: Transient Analysis Result: Plot of v(4), v(5), and ITPOWERD (power)



Viewing Star-Hspice AC Analysis Waveforms

1. Close the transient waveforms. From the Signal Manager dialog box, select bjtdiff(1) and click on **Close Plotfiles**. This closes all transient plots.
2. In the Signal Manager, click on **Open Plotfiles**.
3. In the Open Plotfiles dialog box, in the Files of Type fields, select the Hspice AC (*.ac*) item.
4. You will see bjtdiff.ac0 in the menu. Click on it and click **Open**. The bjtdiff Plot File windows are displayed.
5. Hold down the Ctrl key, and select the dim2(mag) and dim3(mag) signals.
6. Click on **Plot** from the bjtdiff Plot File window. Two cascaded plots open. For two signals in a plot, right-click on dim2(mag) to get a **Signal Menu**.
7. From the **Signal Menu**, select **Stack Region > Analog 0.** You will see a plot similar to the one below, Figure D-9.

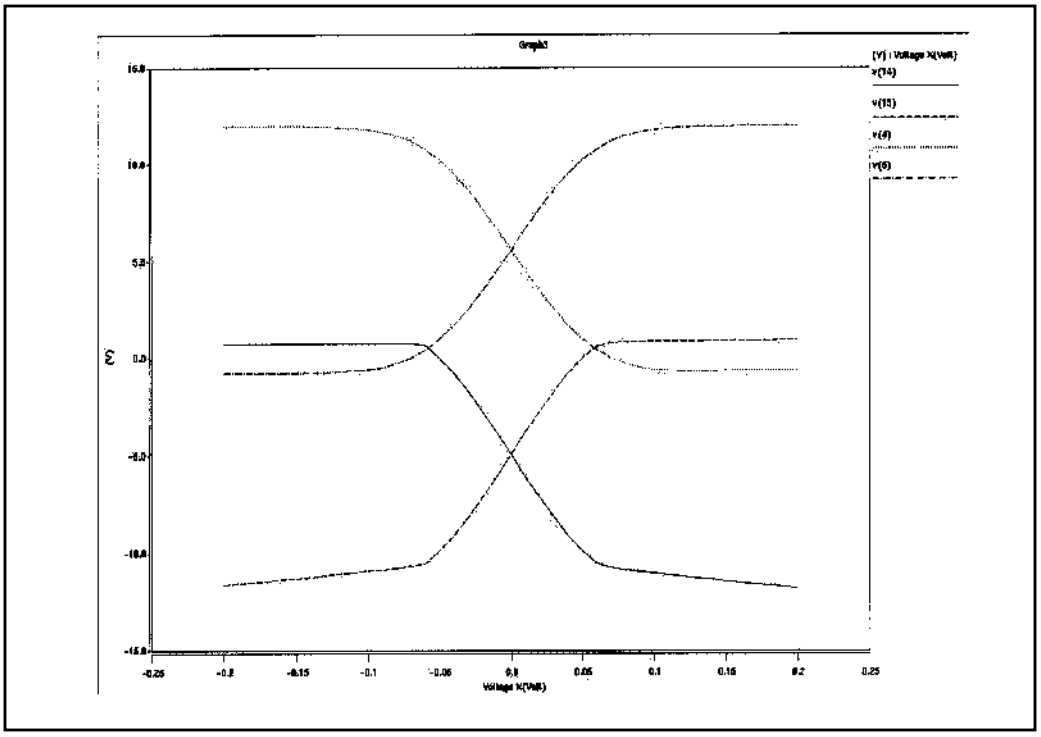
Figure D-9: AC Analysis Result: Plot of dim2(mag) and dim3(mag) from bjtdiff.ac0



Viewing Star-Hspice DC Analysis Waveforms

1. Close the AC waveforms. From the Signal Manager dialog box, select bjtdiff(1) and click on **Close Plotfiles**. This closes all AC plots.
2. In the Signal Manager, click on **Open Plotfiles**.
3. In the Open Plotfiles dialog, Files of Type field, select Hspice DC (*.sw*).
4. Click on bjtdiff.sw0 and **Open** in the menu to display Plot File windows.
5. While holding down the Ctrl key, select all signals.
6. Click on **Plot** from the bjtdiff Plot File window.You will see four cascaded plots. To see four signals in one plot, right-click on the name of the top-most signal to get a **Signal Menu**.
7. From the **Signal Menu**, select **Stack Region** > **Analog 0**.
8. Repeat Steps 6 and 7 for next two top-most signals. You will see a plot similar to one below, Figure D-10.

Figure D-10: DC Analysis Result: Plot of v(14), v(15), v(4), and v(5) from bjtdiff.sw0



The *Cosmos-Scope User's and Reference Manual* includes a full tutorial, information on the various Scope tools, and reference information on the Measure tool. For more information, see the Scope Manual or visit the Avant! website at:

```
http:// www.avanticorp.com
```

Avant!

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z