

HSPICE® User Guide: Basic Simulation and Analysis

Version H-2013.03, March 2013

SYNOPTIS®

Copyright and Proprietary Information Notice

Copyright © 2013 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>. All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.
700 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Inside this Guide	xxxiii
The HSPICE Documentation Set	xxxvi
Searching Across the HSPICE Documentation Set	xxxvii
Known Limitations and Resolved STARS	xxxviii
Conventions	xxxviii
Customer Support	xxxix

Part I: Introduction to HSPICE

1. HSPICE Overview	1
HSPICE Varieties	2
Features	3
Case Sensitivity	5
Custom CMI	5
TSMC Model Interface (TMI)	5
HSPICE Features for Running Higher-Level Simulations	6
Simulation Structure	7
Experimental Methods Supported by HSPICE	7
Measurement System in HSPICE	9
Simulation Process Overview	9
Parser Syntax Requirements (Unsupported Formats)	10
Use of Example Syntax	12
Recommended HSPICE Resources	12
2. Setup	15
Setting Environment Variables	15
License Variables	16
License Queuing Variable	17
Controlling the License Queueing Interval	17

Contents

License Server Down Iterations	18
Using the FlexLM MAX Option	18
HSPICE License Schedule (G-2012.06)	18
Limiting the Number of Licenses for Your Group	18
Temporary Directory Variable	19
Windows Variables	19
Shared Libraries Environment Variable	20
Compiled Function Library Environment Variable	20
Using Environment Variables as Include Statements	21
Setting Environment for 64-bit HSPICE	21
Setting Distributed Processing Environment Variable	22
Verilog-A (pVA) Environment Variables	22
Standard Input Files	22
Design and File Naming Conventions	23
Initialization File (hspice.ini)	23
DC Operating Point Initial Conditions File	24
Input Netlist File	24
Library Input File	24
Analog Transition Data File	24
Standard Output Files	25
AC Analysis Results File	26
AC Analysis Measurement Results File	26
DC Analysis Results File	26
DC Analysis Measurement Results File	26
FFT Analysis Graph Data File	27
Operating Point Information File	27
Operating Point Node Voltages File	27
Output Listing File	27
Output Status File	29
Output Tables	29
Subcircuit Cross-Listing File	29
Transient Analysis Measurement Results File	29
Transient Analysis Results File	29
Waveform Viewing File	30
Working Directory Path Character Limit	30
Continuing HSPICE Simulations after a Power Down	31

3. Startup and Simulation	33
Running HSPICE Simulations	34
Using Isomorphic Analyses in Subckt Blocks	36
Data Inputs and Outputs	37
Limitations	38
Running HSPICE Simulations on Windows	38
Running HSPICE RF Simulations	39
Running HSPICE Interactively	39
Starting Interactive Mode	40
Running a Command File in Interactive Mode	40
Quitting Interactive Mode	40
Using HSPICE in Client-Server Mode	40
To Start Client-Server Mode	41
To Simulate a Netlist in Client-Server Mode	42
To Quit Client-Server Mode	43
Launching the Advanced Client-Server Mode (-CC)	43
Advanced Client-Server Command Syntax	44
Application Instances	45
Running HSPICE to Calculate New Measurements	48
<hr/>	
4. Multiple Simulations, DP, and HPP	49
Running Multi Simulations	49
Running Distributed Processing (DP) on a Network Grid	50
HSPICE DP Feature Support	53
Output Files	54
HSPICE DP Limitations	57
Running Multithread/Multiprocess HSPICE Simulations	57
Running Multithreading and Multiprocessing Concurrently	58
Performance Improvement Estimations	60
Multithread (-mt) on Windows	61
HSPICE Precision Parallel (-hpp)	61
HPP Supported Features	61
HPP Status Reporting in *.lis File	64
HPP Limitations	64

5. Input Netlist and Data Entry	65
Input Netlist File Guidelines	65
Input Line Format	67
Case Sensitivity	69
Special Characters	70
First Character	74
Delimiters	74
Instance Names	75
Hierarchy Paths	76
Numbers	76
Parameters and Expressions	78
Reserved Keywords	79
Input Netlist File Structure	81
Schematic Netlists	82
Compressed Netlist Guidelines	83
Input Netlist File Composition	84
HSPICE Topology Rules	86
Title of Simulation	87
Comments and Line Continuation	88
Element and Source Statements	89
Defining Subcircuits	91
Node Name (or Node Identifier) Conventions	91
Using Wildcards on Node Names	93
Wildcard Applications and Examples	93
Element, Instance, and Subcircuit Naming Conventions	95
Subcircuit Node Names	95
Path Names of Subcircuit Nodes	96
Abbreviated Subcircuit Node Names	97
Automatic Node Name Generation	97
Global Node Names	98
Circuit Temperature	98
Data-Driven Analysis	98
Library Calls and Definitions	99
Library Building Rules	99
Automatic Library Selection (HSPICE)	100
Defining Parameters	100
Predefined Analysis	100
Measurement Parameters	101

Altering Design Variables and Subcircuits	101
Using Multiple .ALTER Blocks	102
Adding or Changing Analyses in .ALTER Blocks	103
Connecting Nodes	105
Deleting a Library	105
Ending a Netlist	105
Condition-Controlled Netlists (IF-ELSE).	106
Using Subcircuits	108
Hierarchical Parameters.	109
M (Multiply) Parameter	110
S (Scale) Parameter	110
Using Hierarchical Parameters to Simplify Simulation	111
Undefined Subcircuit Search (HSPICE).	113
Troubleshooting Subcircuit Node Issues	113
Subcircuit Call Statement Discrete Device Libraries	116
DDL Library Access	116
Vendor Libraries	117
Subcircuit Library Structure	118

6. Using Interactive Mode	121
Invoking Interactive Mode	121
Quitting Interactive Mode	122
Executing an Interactive Script.	122
Using Interactive Mode on the Windows Platform	122
Examples	122
Getting Help	123
Creating a Netlist	124
Specifying an Analysis	124
Running an Analysis	124
Viewing a Netlist.	124
Loading and Running an Existing Netlist	125
Using Environment Commands	126
Recording and Saving Interactive Commands to a File	126
Printing a Voltage Value During Simulation	127
Using a Command File to Run in Interactive Mode	128
Running Multiple Testcases	128

Contents

7. HSPICE GUI for Windows	129
Working with Designs	130
Configuring the HSPICE GUI for Windows	131
Launching Waveview in HSPUI	133
Running Multiple Simulations.	135
Building the Batch Job List.	136
Simulating a Batch Job	136
Sample Batch Work-Flow	137
Running Multithreading	139
SPutil, Metaencrypt and Converter Utilities, Client/Server Operation	139
CMI Directory Structure	140
Troubleshooting Guide	140
Setting the hspui.cfg File Values	140
Text Editor Issues	142
Simulating a UNIX Netlist File	142
Running AvanWaves Using the Cscope Button	142

8. Library and Data Encryption	145
Organization	145
Library Encryption	146
Encrypting a Model Library Using the metaencrypt Utility	146
Three Encryption Methods	147
Installing and Running metaencrypt	147
Installing metaencrypt	148
Running metaencrypt	148
Encryption Guidelines	149
General Example	150
Using the -pipe Option	152
Traditional Library Encryption	152
Creating Files Using Traditional Encryption	153
Non-Library Encrypted Portions	153
*.lib File Encryption	154
Example: Traditional (freelib) Encryption in an HSPICE Netlist	155
8-Byte Key Encryption	156
Creating 8-byte key Encryption	157

Placing an 8-byte key Encrypted File into a HSPICE Netlist	158
Triple DES Public and Random Keys	158
Creating 3DES Encrypted Files	159
Placing 3DES Encryption Files into a HSPICE Netlist	160
Troubleshooting	161
warning parameters... as an expression containing output signals . .	161
Encrypting S-parameter files	162
Freelib Issue with Equations	163

Part II: Elements and Devices

9. Elements	167
Passive Elements	167
Values for Elements	168
Resistor Elements in a HSPICE or HSPICE RF Netlist	168
Linear Resistors	170
Behavioral Resistors	172
Frequency-Dependent Resistors	172
Skin Effect Resistors	173
Capacitors	173
Linear Capacitors	176
Frequency-Dependent Capacitors	177
Behavioral Capacitors in HSPICE or HSPICE RF	178
DC Block Capacitors	179
DC Block and Choke Elements	179
Charge-Conserved Capacitors	180
Inductors	181
Linear Inductors	183
Frequency-Dependent Inductors	184
AC Choke Inductors	185
Reluctors	186
Mutual Inductors	188
Ideal Transformer	190
Ideal Transformer Format in HSPICE RF	192
Multi-Terminal Linear Elements	193
S-element (Scattering Parameter Data)	194
Frequency-Dependent Multi-Terminal (S-element)	194
W-element (Distributed Transmission Lines)	197
W-element Statement	197

Contents

U-element (Lumped Transmission Lines)	201
Using the Scattering Parameter Element in HSPICE	203
S-element (Generic Multiport)	203
S-element Syntax	204
Node Example	209
Frequency-Dependent Multi-Terminal (S-element) in HSPICE RF	211
Port Element	213
Using the Port Element for Mixed-Mode Measurement	217
Active Elements	217
Diode Element	218
Bipolar Junction Transistor (BJT) Element	220
JFETs and MESFETs	222
MOSFETs	224
Extended MOSFET Element Support Using .OPTION MACMOD	227
Direct X-Element Mapping to a MOSFET Model Card	230
IBIS Buffers (HSPICE Only)	231
<hr/>	
10. Sources and Stimuli	233
Independent Source Elements	234
Source Element Conventions	234
Independent Source Element Syntax	234
DC Sources	239
AC Sources	239
Transient Sources	239
Mixed Sources	240
Independent Source Functions	240
Trapezoidal Pulse Source	241
Sinusoidal Source Function	245
Exponential Source Function	249
Repeat Option in Sinusoidal and Exponential Sources	252
Piecewise Linear Source	252
General PWL Form	253
MSINC and ASPEC Form	253
Data-Driven Piecewise Linear Source	255
File-Driven PWL Source	256
PWLZ High Impedance State	257
Single-Frequency FM Source	258
Single-Frequency AM Source	260

Pattern Source	262
Nested-Structure Pattern Source	265
Pattern-Command Driven Pattern Source	266
Workaround to 1024 Character Limitation for Long Pattern Sources	267
Pseudo Random-Bit Generator Source	268
Linear Feedback Shift Register	270
Conventions for Feedback Tap Specification	271
Example: Noise Generator Used for a Pulse or DC Level	272
Voltage and Current Controlled Elements	273
Polynomial Functions	275
One-Dimensional Function	275
Two-Dimensional Function	276
Three-Dimensional Function	277
N-Dimensional Function	279
Piecewise Linear Function	279
Power Sources	280
Independent Sources	280
Using the Keyword POWER	281
Calculation for Total Dissipated Power and for Voltage Source Power	283
Subcircuit Power Calculation	284
Controlled Sources	285
Voltage-Dependent Voltage Sources — E-elements	285
Voltage-Controlled Voltage Source (VCVS)	286
E-element Parameters	287
Linear	289
Polynomial (POLY)	289
Piecewise Linear (PWL)	290
Multi-Input Gates	290
Delay Element	290
Laplace Transform	290
Pole-Zero Function	291
Frequency Response Table	292
Foster Pole-Residue Form	294
Behavioral Voltage Source (Noise Model)	295
Ideal Op-Amp	296
Ideal Transformer	296
E-element Examples	297
Ideal OpAmp	298
Voltage Summer	298
Polynomial Function	299
Zero-Delay Inverter Gate	299
Delayed and Inverted Signal	299

Contents

Differential Amplifiers and Opamp Signals	300
Ideal Transformer	300
Voltage-Controlled Oscillator (VCO).	300
Switching between Two Voltage Sources Connected to the Same Node	302
Using the E-element for AC Analysis	302
Current-Dependent Current Sources — F-elements	303
Current-Controlled Current Source (CCCS) Syntax.	304
Linear	304
Polynomial (POLY)	304
Piecewise Linear (PWL)	304
Multi-Input Gates	304
Delay Element	305
F-element Parameters	305
F-element Examples	307
Voltage-Dependent Current Sources — G-elements	308
Voltage-Controlled Current Source (VCCS).	309
G-element Parameters	310
Linear	313
Polynomial (POLY)	313
Piecewise Linear (PWL)	314
Multi-Input Gate	314
Delay Element	314
Laplace Transform	315
Pole-Zero Function.	315
Frequency Response Table	315
Foster Pole-Residue Form	315
Behavioral Current Source (Noise Model)	315
Voltage-Controlled Resistor (VCR)	317
Linear	317
Polynomial (POLY)	317
Piecewise Linear (PWL)	317
Multi-Input Gates	318
Voltage-Controlled Capacitor (VCCAP)	318
NPWL Function	318
PPWL Function	319
G-element Examples	319
Modeling Switches	320
Switch-Level MOSFET	320
Runtime Current Source with Equation Containing Output Variable	321
Voltage-Controlled Capacitor	321
Zero-Delay Gate.	321
Delay Element	322
Diode Equation.	322

Diode Breakdown	322
Diode Lookup Table (vs. Model)	322
Triodes	323
Behavioral Noise Model	323
Turning off Smoothing	323
Using Dependent Sources to Convert I to V and V to I	324
Additional Full Demonstration Netlists Using G-Element	324
Current-Dependent Voltage Sources — H-elements	324
Current-Controlled Voltage Source (CCVS)—H-Element	325
Linear	325
Polynomial (POLY)	325
Piecewise Linear (PWL)	325
Multi-Input Gate	326
Delay Element	326
HSPICE RF-Specific Sources and Stimuli	329
Steady-State Voltage and Current Sources	329
I and V Element Syntax	330
Steady-State HB Sources	333
Phase Differences Between HB and SIN Sources	335
Behavioral Noise Sources	336
Using Noise Analysis Results as Input Noise Sources	338
Syntax Rules for Behavioral Noise Source Files	339
Power Supply Current and Voltage Noise Sources	341
Function Approximations for Distributed Devices	342
Foster Pole-Residue Form for Transconductance or Gain	342
Advantages of Foster Form Modeling	343
G and E-element Syntax	343
Complex Signal Sources and Stimuli	344
Vector-Modulated RF (VMRF) Source	344
Voltage and Current Source Elements	346
SWEEPBLOCK in Sweep Analyses	352
Input Syntax	353
Using SWEEPBLOCK in a DC Parameter Sweep	354
Using in Parameter Sweeps in TRAN, AC, and HB Analyses	354
Limitations	355
Clock Source with Random Jitter	355
Syntax of SIN, COS, and Pulse Sources	355
Specifying a Digital Vector File and Mixed Mode Stimuli	358
Commands in a Digital Vector File	358
Vector Patterns	359
Defining Tabular Data	359

Contents

Input Stimuli	360
Expected Output.	361
Verilog Value Format	362
Periodic Tabular Data	363
Waveform Characteristics	363
Modifying Waveform Characteristics	364
Using the Context-Based Control Option (CBC)	365
Comment Lines and Line Continuations	366
Parameter Usage	366
First Group	366
Second Group	367
Third Group	367
Digital Vector File Example	368
<hr/>	
11. Parameters and Functions	371
Using Parameters in Simulation (.PARAM)	371
Defining Parameters	371
Assigning Parameters	373
Example: Modeling an eFuse.	374
Inline Parameter Assignments	374
Parameters in Output	374
User-Defined Function Parameters	376
Using Parameter Functions to Evaluate Expressions Containing Dynamic Signals	377
Predefined Analysis Function.	378
Measurement Parameters	378
.PRINT and .PROBE Parameters	378
Multiply Parameter	379
Using Algebraic Expressions	379
Built-In Functions and Variables	380
Parameter Scoping and Passing	385
Library Integrity	385
Reusing Cells	386
Creating Parameters in a Library	386
String Parameter (HSPICE Only)	389
String Parameters in Passive and Active Component Keywords	390
Parameter Defaults and Inheritance.	391
Parameter Passing	392
Parameter Passing Solutions	394

12. Simulation Output	395
Overview of Output Statements	396
Output Commands	396
Output Variables	397
Displaying Simulation Results	398
.PRINT Statement	399
Statement Order	399
.PROBE Statement	399
Using Wildcards in PRINT and PROBE Statements	400
Supported Wildcard Templates	401
Using filter in .PRINT and .PROBE Statements	401
Using level in .PRINT and .PROBE Statements	402
Switching to .PROBE to Output Subcircuit Port Names	402
Print Control Options	403
Changing the File Descriptor Limit (HSPICE Only)	403
Printing the Subcircuit Output	404
Using .MODEL_INFO to Print Model Parameters	405
Progress Message Time Estimates	407
Selecting Simulation Output Parameters	407
DC and Transient Output Variables	407
Nodal Capacitance Output	408
Nodal Voltage	409
Current: Independent Voltage Sources	409
Terminal Voltage: MOS Instance	410
Current: Element Branches	410
Current: Subcircuit Pin	413
Independent Source Power Output	413
Wildcard Support	414
Print Power	414
Diode Power Dissipation	415
BJT Power Dissipation	415
JFET Power Dissipation	416
MOSFET Power Dissipation	417
Operating Point Variables	418
AC Analysis Output Variables	418
Nodal Capacitance Output	419
Nodal Voltage	420
Current: Independent Voltage Sources	420
Current: Element Branches	421
Current: Subcircuit Pin	421
Group Time Delay	421

Contents

Network	422
Noise and Distortion	423
Element Template Output (HSPICE Only)	423
Specifying User-Defined Analysis (.MEASURE)	424
.MEASURE Statement Order	425
.MEASURE Parameter Types	426
FIND and WHEN Functions	427
Continuous Measurement	428
Continuous Measure Output Files	428
Equation Evaluation	429
Average, RMS, MIN, MAX, INTEG, PP, and EM_AVG	429
Measuring Recovered Electromigration	430
INTEGRAL Function	431
DERIVATIVE Function	431
ERROR Function	431
Error Equations	431
Generating a Measure File (*.mt0, *.ac0, etc.) with All Values in a Single Row 433	
Outputting Pass/Fail Measure Data	433
Measurements in MOSRA Analysis	434
Expected State of Digital Output Signal (.DOUT)	435
Reusing Simulation Output as Input Stimuli (HSPICE Only)	437
Output Files	437
Element Template Listings (HSPICE Only)	438
Vdmargin Output	446
Output Listing (*.lis) File with .OPTION LIS_NEW Set	447
For MOSFET Information Use .OPTION LIST	449
HPP Status Updates in *.lis File	451
Verilog-A Simulation Output	452
Verilog-A Output Directory	452
Field Solver *.str File	453
Redirecting the Simulation Output Results Files to a Different Directory	454
Directing .PRINT Output to a Separate File	455
Getting Data Out of HSPICE Plot Files	455
Compressing Waveform Files	457
Using the HSPICE Output Converter Utility	458

PSF Converter	458
PWL/DATA/VEC Converter	459
Input Line Dependencies	461
Running the Converter Utility in Batch Mode	462
Troubleshooting Issues	463
Resolving Inductor/Voltage Source Loop Errors	463
Voltage Source Missing Rising and Falling Edges	463

Part III: Analyses and Simulation

13. Initializing DC-Operating Point Analysis	469
Simulation Flow—Initialization and Analysis	469
DC Initialization and Operating Point Calculation	473
.OP Statement — Operating Point	473
Output	474
Element Statement IC Parameter	474
Initial Conditions and UIC Parameters	475
.SAVE and .LOAD Statements (HSPICE Only)	476
.DC Statement—DC Sweeps	476
Multi-dimensional DC Sweep	477
Other DC Analysis Statements	478
DC Initialization Control Options	478
Accuracy and Convergence	479
Accuracy Tolerances	480
Autoconverge Process	482
Effects of GMINDC	482
Reducing DC Errors	486
Shorted Element Nodes	487
Inserting Conductance, Using DCSTEP	488
Floating-Point Overflow	489
Diagnosing Convergence Problems	489
Non-Convergence Diagnostic Table	489
Traceback of Non-Convergence Source	491
Solutions for Non-Convergent Circuits	492
Poor Initial Conditions	492
Inappropriate Model Parameters	493

Contents

PN Junctions (Diodes, MOSFETs, BJTs)	495
Troubleshooting DC Bias Point and DC Sweep Non-Convergence	495
Convergence Failure: Too Many Current Probes in Netlist.	497
Troubleshooting: Nodes set to initial conditions with .IC may not always begin at those voltage values	497
<hr/>	
14. AC Small-Signal and Noise Analysis	499
Using the .AC Statement	500
.AC Control Options	500
.AC Command Examples	500
AC Small Signal Analysis.	501
AC Analysis of an RC Network	503
Using .NOISE for Small-Signal Noise Analysis	506
Using .AC/.NOISE Analyses with .TRAN.	510
Measuring Total Noise	510
Other AC Analysis Statements - .LSTB, .DISTO, and .SAMPLE.	511
Using .LSTB for Loop Stability Analysis.	511
Output Formats for Loop Stability (.LSTB) Analysis.	512
Loop Stability Analysis Usage	512
Single-Ended Mode Example: Ideal Inverter	513
Differential Mode Example: Bandgap.	514
Controlling LSTB Warning Messages	516
Using .DISTO for Small-Signal Distortion Analysis	517
Using .SAMPLE for Noise Folding Analysis.	518
<hr/>	
15. Transient Analysis	519
Simulation Flow	520
Overview of Transient Analysis	520
Data-Driven vs. Outer Parameter Sweeps	521
Transient Analysis Output	524
Transient Analysis of an RC Network.	525
Transient Analysis of an Inverter	526
Transient Control Options	528
Simulation Speed and Accuracy Using the RUNLVL Option	529
RUNLVL Features	530
Interactions Between .OPTION RUNLVL and Other Options.	531

Numerical Integration Algorithm Controls	533
TRAP	534
GEAR and Backward-Euler	534
BDF	535
Dynamic Check Using the .BIASCHK Statement	536
Using BIASCHK with HPP	539
Storing and Restoring Checkpoint Files	539
Store by Using the .STORE Command	540
.....Store	
by Interrupting the Simulation Process.	541
Restore Operation	542
Example	542
Usage Requirements	542
Specifying Monte Carlo or Temperature Sweeps.	542
Troubleshooting: Internal Timestep, Measurement Errors	543
Troubleshooting 'Timestep Too Small' Errors	543
Stepsize Increases During a Simulation	545
How TSTEP Affects a Transient Simulation	545
Troubleshooting .MEASUREMENT Issues	547

16. Spectrum Analysis.	549
Spectrum Analysis (Fourier Transform)	549
Using the Fourier-Related Statements and Options	551
Fourier Accuracy	551
Fourier Equation.	551
.FFT Analysis.	554
Using Windows in FFT Analysis.	554
Examining the FFT Output.	558
Measuring FFT Output Information	561
AM Modulation.	562
Graphical Output	563
Balanced Modulator and Demodulator	564
Signal Detection Test Circuit	569
Output	570

Contents

17. Pole-Zero Analysis	575
Overview of Pole-Zero Analysis	575
Using Pole-Zero Analysis	576
Matrix Approach	576
Muller Method	577
How HSPICE Calculates Poles and Zeros	577
Pole/Zero Analysis Examples	578
Example 1 – Low-Pass Filter	579
Example 2 – Kerwin’s Circuit	582
Example 3 – High-Pass Butterworth Filter	583
Example 4 – CMOS Differential Amplifier	584
Example 5 – Simple Amplifier	586
Example 6— Active Low-Pass Filter	587
References	594

18. Performing Digital Cell Characterization	595
Performing Basic Cell Measurements	596
Rise, Fall, and Delay Calculations	596
Delay versus Fanout	599
Pin Capacitance Measurement	600
Op-amp Characterization of LM124	601
Performing Advanced Cell Characterization	602
Cell Examples	603

19. MOSFET Model Reliability Analysis (MOSRA)	607
MOSRA Overview	607
Reliability Analysis Use Model	608
Example Setup	609
MOSRA Commands and Control Options	609
.MOSRA	610
MOSRA Support for DC/AC/MC Analysis in Post-Stress Simulation	615
Getting Measurements in a MOSRA Analysis	617
.MOSRAPRINT	618
.MOSRA_SUBCKT_PIN_VOLT	619
.MODEL	619

.APPENDMODEL	620
.OPTION APPENDALL	622
.OPTION DEGF	622
.OPTION DEGFN	622
.OPTION DEGFP	623
.OPTION MOSRALIFE	623
.OPTION MOSRASORT	624
.OPTION MRAAPI	624
.OPTION MRAEXT	625
.OPTION MRAPAGED	625
.OPTION MRA00PATH, MRA01PATH, MRA02PATH, MRA03PATH .	625
.OPTION RADEGFILE	626
.OPTION RADEGOUTPUT	626
Simulation Output Files	627
RADEG Output Sorting (.OPTION MOSRASORT)	629
Specify a MOSRA degradation File Name to be Used with SIMMODE=1 (.OPTION RADEGFILE)	629
Usage Model: SimMode=3 (continual degradation integration through alters) 629	
CSV Format Degradation Information (.OPTION RADEGOUTPUT)	631
Level 1 MOSRA BTI and HCI Model Parameters	632

20. Post-Layout Simulation: RC Network Reduction and Back-Annotation .	641
Linear Acceleration	642
PACT Algorithm	643
PI Algorithm	644
LNE Algorithm	644
Linear Acceleration Control Options Summary	645
Supporting Parasitic L- and K-elements	646
Pruning Parasitics from a Post-Layout Flat Netlist	647
Post-Layout Back-Annotation	650
Full Back-Annotation	651
Flow for Full Back-Annotation	651
Selective Net Back-Annotation	653
Flow for Selective Net Back-Annotation	654
Active Net Generation	655
Warnings/Error Messages	656
Listing of Back-Annotation Commands and Options	657
Application Note: Back-Annotation	658
Full Back-Annotation	659

Contents

Selective Back-Annotation	664
Recommended Settings for STAR-RC-XT Command File	666
DSPF and SPEF File Structures	667
<hr/>	
21. Multi-Technology Simulation of 3D Integrated Circuit	671
Overview of HSPICE 3D-IC Simulation Netlist	671
HSPICE Netlist Definitions	672
IC Module Usage	672
3D-IC Netlist Construct and Usage	673
Scope Reference Rules	675
Using Verilog-A Modules Within the .MODULE Scope	677
VA Reference Usage Rules	678
Top-Level IC Module Reference	680
Intrinsic Model Card Reference from a Top-Level Instance	681
. IC	
Module Instance-Specific Properties Scope	681
Over-writing Circuit Properties	683
Parameter Direct Access from Top-Level Netlist	685
Transient Analysis and Alters Simulation Features	686
Sweeps for Temperature and Parameters	686
.ALTER Features	688
Full Circuit Example	690
<hr/>	
Part IV: Variation and Statistical Analysis	
<hr/>	
22. Monte Carlo—Traditional Flow Statistical Analysis	695
Application of Statistical Analysis	696
Analytical Model Types	697
Circuit and Model Temperature Simulation	698
Temperature Analysis	699
Worst-Case Analysis	701
Model Skew Parameters	701
Using Skew Parameters	703
Skew File Interface to Device Models	705
Traditional Monte Carlo Simulations	706

Traditional Monte Carlo Analysis Characteristics	709
Monte Carlo Setup	710
Monte Carlo Output	712
.PARAM Distribution Function	713
Monte Carlo Parameter Distribution	716
Non-Gaussian Probability Distribution Functions	716
Monte Carlo Examples	718
Gaussian, Uniform, and Limit Functions	718
Major and Minor Distribution	719
RC Time Constant	721
Switched-Capacitor Filter Design	721
Advanced Sampling Methods	723
Invoking Advanced SRS	725
Worst-Case and Monte Carlo Sweep Example	726
Transient Sigma Sweep Results	728
Operating-Point Results in Transient Analysis	728
Monte Carlo Results	729
Global and Local Variations with Monte Carlo	734
Key to Demonstration Examples for Monte Carlo	735
Variations Specified on Geometrical Instance Parameters	735
Variations Specified in the Context of Subcircuits	737
Variations on a Model Parameter Using a Local Model in Subcircuit	737
Indirect Variations on a Model Parameter	738
Variations Specified on Model Parameters	738
Local Variations for Transistor Fingers	739
Variations Specified Using DEV and LOT	740
Combinations of Variation Specifications	740
Variation on Model Parameters as a Function of Device Geometry	741
Troubleshooting Monte Carlo Issues	742
Perturbation Information Missing from Output Listing in Monte Carlo and	
Subcircuit Local Variables	742

23. Variability Analysis Using the Variation Block	745
Overview of Variation on Silicon	745
Variability in HSPICE	747
Overview of the Variation Block	749
Variation Block Structure	751
General Section	751
Subblocks for Global, Local, and Spatial Variations	752

Contents

Independent Random Variables	753
Dependent Random Variables	756
Absolute Versus Relative Variation.	758
Variations on Model Parameters	758
Variations on Subcircuit Parameters	760
Variations on Top-Level Parameters.	762
Variations on Temperature	762
Access Functions, Get_E(), Get_P(), Get_M(), Get_O()	763
Spatial Variation	769
Special Rules Regarding Variation Block Usage	769
Variation Block Examples	770
Group Operator {...} and Subexpressions	772
Syntax	772
Syntax Extension with Bins	772
Example	772
Rules for Using the Group Operator.	773
Interconnect Variation in StarRC with the HSPICE Flow	775
Variation Block and Statistical Sensitivity Coefficients	776
Usage Example and Input Syntax	778
1: Interconnect Variation Block.	778
2: Model Card in the Header Section	780
3: Parasitic Section	780
Control Options and Syntax.	781
References.	783
<hr/>	
24. Monte Carlo Analysis—Variation Block Flow	785
Overview: Monte Carlo Using the Variation Block Flow	785
Monte Carlo Analysis in HSPICE.	787
Input Syntax	790
Monte Carlo-Specific Variation Block Options	791
Output for Variation Block Monte Carlo	794
Simulation Listing	794
Measurement Output File.	795
Parameter File	795
Sampling Options.	800
Simple Random Sampling (SRS).	801
One-Factor-at-a-Time (OFAT) Sampling	801
Factorial Sampling	802

Latin Hypercube Sampling (LHS)	803
Monte Carlo with firstrun and list options:	804
Sobol and Niederreiter Sampling (LDS)	805
External Sampling	806
Usage Model for External Sampling	807
Syntax	807
Controlling the Read-in of an External File	808
Using the MC_File_Only Option	809
Allowing Zero Variations in External Sampling for Missing IRVs	809
Comparison of Sampling Methods	810
Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo 819	
Gaussian Style Random Variable Definition	821
Input/Output with New Capability	822
Example 1: Variation Duplicated in Traditional Format and Variation Block	823
Example 2: Subcircuit and Macro Models	826
Application Considerations	830
Troubleshooting Monte Carlo-VB Issues	830
Independent Random Variable Assignments	830
References	833

25. Mismatch Analyses	835
Mismatch	836
DCMatch Analysis	837
Input Syntax	838
DCMatch Table Output	839
Output Using .PROBE and .MEASURE Commands	843
Syntax for .PROBE Command for DCMatch	844
Syntax for .MEASURE Command	844
DCMatch Example Netlist	845
ACMatch Analysis	847
Parasitic Capacitor Sensitivity	848
Input Syntax	849
ACMatch Table Output	850
Output from .PROBE and .MEASURE Commands for ACMatch	853
Application Considerations	855
Mismatch Compared to Monte Carlo Analysis	856
Supported Models with DCMatch and ACMatch	856

Contents

References	857
<hr/>	
26. Monte Carlo Data Mining	859
Post-Processing of Monte Carlo Results	859
Summary Statistics	860
Variable Screening	868
External Sampling — *.corner File	873
Corner Analysis - DC Monte Carlo/Transient Analysis	874
Back-Annotation — *.annotate File	877
Standalone Data Mining in HSPICE	877
<hr/>	
27. DC Sensitivity Analysis and Variation Block	879
Sensitivity Block Using the Variation Block Construct	879
Input Syntax	880
<hr/>	
Part V: Optimization	
<hr/>	
28. Exploration Block	885
Exploration Block Functions	886
Usage Guidelines	886
Multiple Instantiations of the Same Cell or Subcircuit	886
Specifying Relationships between Devices	887
Specifying Relationships between Device Properties	887
Subcircuits and Elements Supported for Exploration	888
Flow Using an External Exploration Tool	889
Information Extraction and Export Phase	889
Definition Phase (Outside HSPICE)	890
Exploration Phase	891
Exploration Block Structure	892
Syntax	892
Control Options	893
Example: Option secondary_param=yes	895
Parameters Section	895
Device Relationships	897
Property Relationships	897

Area Measurement	899
Rules for Area Measurement using design_area Keyword	900
Specifying Constraints	901
The Processing of Netlist Parameters	903
Export File Syntax	903
Syntax Structure	904
Execution of Exploration in HSPICE	905
EXCommand	Option:
Export	Data
Block.	Action
906	
Exploration Data Block Syntax.	906
Exploration and Variation Block Interactions	907
Limitations	907

29. Optimization	909
Overview	909
Optimization Control.	910
Simulation Accuracy.	910
Curve Fit Optimization	911
Goal Optimization.	911
Timing Analysis	912
Optimization Statements	912
Optimizing Analysis (.DC, .TRAN, .AC)	913
Optimization Examples	914
MOS Level 3 Model DC Optimization.	914
Input Netlist File for Level 3 Model DC Optimization	915
MOS Level 13 Model DC Optimization.	917
DC Optimization Input Netlist File for Level 13 Model	918
RC Network Optimization.	918
Optimization Results	919
Optimized Parameters OPTRC	920
Optimizing CMOS Tristate Buffer	922
Input Netlist File to Optimize a CMOS Tristate Buffer.	923
BJT S-parameters Optimization.	925
BJT Model DC Optimization.	927
Optimizing GaAsFET Model DC	929
Optimizing MOS Op-amp.	930

Contents

30. Timing Analysis Using Bisection	935
Overview of Bisection	936
Bisection Methodology	938
Measurement	938
Optimization	938
Using Bisection	939
Examining the Command Syntax	940
Performing Transient Analyses with Bisections	942
Setup Time Analysis	942
Minimum Pulse Width Analysis	946
Pushout Bisection Methodology	947
Using RELOUT and RELIN to Affect HSPICE Bisection Optimization	950
Using Bisection with Monte Carlo Analysis	952
Setting Up Monte Carlo Analysis with Bisection	953
Performing Bisection with Monte Carlo Sweep	953
Bisection Optimization Application Example	961

Part VI: Errors-Warnings/Demonstration Files

31. Running Demonstration Files	965
Using the Demo Directory Tree	965
Two-Bit Adder Demo	967
One-Bit Subcircuit	967
MOS Two-Bit Adder Input File	968
MOS I-V and C-V Plotting Demo	968
Printing Variables	969
MOS I-V and C-V Plot Example Input File	973
CMOS Output Driver Demo	973
Strategy	974
CMOS Output Driver Example Input File	978
Temperature Coefficients Demo	978
Input File for Optimized Temperature Coefficients	979
Optimization Section	980

Modeling Wide-Channel MOS Transistors	980
Listing of Demonstration Input Files	982
HSPICE Integration to ADE Demonstration Examples	983
Applications of General Interest Examples	984
Back-Annotation Demo Cases	986
Behavioral Application Examples	988
Benchmark Examples	990
Bisection-Timing Analysis Examples	990
BJT and Diode Examples	991
Cell Characterization Examples	991
Circuit Optimization Examples	993
Device Optimization Examples	993
Encryption Examples	995
Filters Examples	995
Fourier Analysis Examples	997
IBIS Examples	998
Loop Stability Analysis	999
Magnetics Examples	999
MOSFET Device Examples	999
RF Examples	1001
Signal Integrity Examples	1002
Sources Examples	1003
S-parameter Examples	1004
Transmission Lines Examples	1004
Transmission (W-element) Line Examples	1005
Variability Examples	1005
Verilog-A Examples	1006

32. Warning/Error Messages	1009
Warning Messages	1010
Topology Warnings	1010
Topology Integrity	1010
No DC Path to Ground	1010
Duplicate Initialization	1011
Model Warnings	1011
Zero or Negative Conductance	1011
Encryption-Related Warnings	1011
Model Binning Warnings	1012
Key Model Parameter Checking	1012

Contents

Parameter Expression Warning	1012
Control Option Warnings	1012
RUNLVL	1012
ACCURATE	1013
FAST	1013
GMIN, GMINDC	1013
Device Warnings	1013
Device Geometry Check	1013
Device Parameter Check	1013
Analysis Warnings	1014
Transient	1014
Bisection	1014
Measure	1015
.DC and .OP Analysis Warnings	1015
Error Messages	1016
Topology Errors	1016
Model Errors	1017
Analysis Errors	1018
.DC and Operating Convergence Errors	1018
Convergence/Conductance	1021
Convergence/Diode Resistance	1022
Analysis Options: DIAGNOSTIC	1023
Transient Analysis Errors and Solutions	1024
Transient Analysis Error	1024
Transient Non-Convergence	1024
Transient Convergence Aids	1025
Safe Operating Area (SOA) Warnings	1026
Verilog-A (pVA) Messages	1029
Warning Message Index [10001-10076]	1029
Error Message Index [20001-20024]	1039
Exit Codes	1042
<hr/>	
A. Full Simulation Example	1045
Simulation Example Using WaveView	1045
Input Netlist and Circuit	1045
Execution and Output Files	1047
Example.ic0	1048
Example.mt0	1048

Example.lis	1049
Example.pa0	1052
Example.st0	1052
View HSPICE Results in WaveView.	1055
<hr/>	
B. Obsolete HSPICE Functionality	1063
U-element Digital and Mixed Mode Stimuli	1063
U-element Digital Input Elements and Models	1063
General Form	1064
Model Syntax	1064
Digital-to-Analog Input Model Parameters	1065
U Element Digital Outputs	1066
Model Syntax	1067
Analog-to-Digital Output Model Parameters	1067
Replacing Sources With Digital Inputs	1069
.NET Parameter Analysis	1071
Behavioral Modeling, Obsolete Functionality	1074
Digital Stimulus Files	1074
Op-Amp Subcircuit Generators (Behavioral Modeling	1074
Op-Amp Model Generator	1075
Op-Amp Element Statement Format	1076
Op-Amp .MODEL Statement Format	1076
Op-Amp Model Parameters	1077
Op-Amp Model Parameter Defaults	1083
Simulation Results	1084
Unity Gain Resistor Divider Mode	1086
<hr/>	
C. HSPICE Parser Strict Syntax Requirements	1087
Listing of Tighter Syntax Restrictions	1087
<hr/>	
Index	1093

Contents

About this User Guide

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

Inside this Guide

This user guide contains the chapters described below. For descriptions of the other manuals in the HSPICE documentation set, see [The HSPICE Documentation Set](#).

Chapter	Description
Part 1, Introduction to HSPICE	
Chapter 1, HSPICE Overview	Describes HSPICE features and the simulation process.
Chapter 2, Setup	Describes the environment variables and standard I/O files.
Chapter 3, Startup and Simulation	Describes the invocation commands and types of simulation and analysis available in HSPICE.
Chapter 4, Multiple Simulations, DP, and HPP	Describes the MultiProcess, MultiThread, Distributed Processing (DP), and HSPICE Precision Parallel (HPP) simulation modes available in HSPICE.
Chapter 5, Input Netlist and Data Entry	Describes the input netlist file and methods of entering data.
Chapter 6, Using Interactive Mode	Provides details on invoking and using interactive mode.
Chapter 7, HSPICE GUI for Windows	Describes the graphical user interface available on the Windows platform only.
Chapter 8, Library and Data Encryption	Describes the three methods available to create encrypted files.

Chapter	Description
Part 2, Elements and Devices	
Chapter 9, Elements	Describes the syntax for the basic elements of a circuit netlist in HSPICE or HSPICE RF.
Chapter 10, Sources and Stimuli	Describes element and model statements for independent sources, dependent sources, analog-to-digital elements, and digital-to-analog elements.
Chapter 11, Parameters and Functions	Describes how to use parameters within an HSPICE netlist.
Chapter 12, Simulation Output	Describes how to use output format statements and variables to display steady state, frequency, and time domain simulation results.
Part 3, Analyses and Simulation	
Chapter 13, Initializing DC-Operating Point Analysis	Describes DC initialization and operating point analysis.
Chapter 14, AC Small-Signal and Noise Analysis	Describes how to perform AC and noise small signal analyses in HSPICE.
Chapter 15, Transient Analysis	Describes how to use transient analysis to compute the circuit solution.
Chapter 16, Spectrum Analysis	Describes how to use spectrum analysis to provide the highest FFT accuracy with minimal overhead in simulation time.
Chapter 17, Pole-Zero Analysis	Describes how to use pole/zero analysis in HSPICE to study the behavior of linear, time-invariant networks.
Chapter 18, Performing Digital Cell Characterization	Describes how to characterize cells in a data-driven analysis.
Chapter 19, MOSFET Model Reliability Analysis (MOSRA)	Describes reliability analysis for MOSFET devices.
Chapter 20, Post-Layout Simulation: RC Network Reduction and Back-Annotation	Describes RC network reduction in HSPICE.

Chapter	Description
Chapter 21, Multi-Technology Simulation of 3D Integrated Circuit	Describes the HSPICE solution to simulate a modularized IC chip inside a 3D integrated circuit.
Part 4, Variation and Statistical Analysis	
Chapter 22, Monte Carlo—Traditional Flow Statistical Analysis	Describes the traditional statistical analysis features in HSPICE and HSPICE RF.
Chapter 23, Variability Analysis Using the Variation Block	Describes the use model and structure of the Variation Block in HSPICE.
Chapter 24, Monte Carlo Analysis—Variation Block Flow	Describes Monte Carlo analysis in HSPICE.
Chapter 25, Mismatch Analyses	Describes the use of DCmatch analysis in HSPICE.
Chapter 26, Monte Carlo Data Mining	Discusses data mining capabilities of Monte Carlo results.
Chapter 27, DC Sensitivity Analysis and Variation Block	Describes enhanced sensitivity analysis in HSPICE focusing on DC simulation using Variation Block.
Part 5, Optimization	
Chapter 28, Exploration Block	Describes the use of the Exploration Block in HSPICE.
Chapter 29, Optimization	Describes optimization for optimizing electrical yield.
Chapter 30, Timing Analysis Using Bisection	Describes how to use the bisection function in timing optimization.
Part 6, Errors-Warnings/Demonstration Files	
Chapter 31, Running Demonstration Files	Contains examples of basic file construction techniques, advanced features, and simulation tricks. Lists and describes several HSPICE and input files.
Chapter 32, Warning/Error Messages	Provides an overview of the type of warnings and error messages that HSPICE prints and troubleshooting measures to take when possible.
Appendices	

Chapter	Description
Appendix A, Full Simulation Example	Contains information and sample input netlist for a full simulation example in HSPICE.
Appendix B, Obsolete HSPICE Functionality	Describes out-of-date, rarely used, or de-emphasized functionality.
Appendix C, HSPICE Parser Strict Syntax Requirements	Discusses HSPICE requirements for efficient usage with regard to improved HSPICE parser.

The HSPICE Documentation Set

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

Manual	Description
HSPICE User Guide: Basic Simulation and Analysis	Describes how to use HSPICE to simulate and analyze your circuit designs, and includes simulation applications. This is the main HSPICE user guide.
HSPICE User Guide: Signal Integrity Modeling and Analysis	Describes how to use HSPICE to maintain signal integrity in your chip design.
HSPICE User Guide: Advanced Analog Simulation and Analysis	Describes how to use special set of analysis and design capabilities added to HSPICE to support RF and high-speed circuit design.
HSPICE Reference Manual: Elements and Device Models	Describes standard models you can use when simulating your circuit designs in HSPICE, including passive devices, diodes, JFET and MESFET devices, and BJT devices.
HSPICE Reference Manual: MOSFET Models	Describes available MOSFET models you can use when simulating your circuit designs in HSPICE.
HSPICE Integration to Cadence® Virtuoso® Analog Design Environment User Guide	Describes use of the HSPICE simulator integration to the Cadence tool.

Manual	Description
AMS Discovery Simulation Interface Guide for HSPICE	Describes use of the Simulation Interface with other EDA tools for HSPICE.
AvanWaves User Guide	Describes the AvanWaves tool, which you can use to display waveforms generated during HSPICE circuit design simulation.

Searching Across the HSPICE Documentation Set

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

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

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

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

Known Limitations and Resolved STARs

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

To access the *HSPICE Release Notes*:

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

Conventions

The following typographical conventions are used in Synopsys HSPICE documentation.

Table 1

Convention	Description
Courier	Indicates command syntax.
<i>Italic</i>	Indicates a user-defined value, such as <i>object_name</i> .
Bold	Indicates user input—text you type verbatim—in syntax and examples. For a graphical user interface, Bold designates any GUI element such as a button or control.
[]	Denotes optional parameters, such as: <code>write_file [-f filename]</code>
()	When shown in a command-line, the parentheses () are part of the syntax. For example: <code>+ LISTFREQ=(1k 100k 10meg)</code>
...	Indicates that parameters can be repeated as many times as necessary: <code>pin1 pin2 ... pinN</code>

Table 1

Convention	Description
	Indicates a choice among alternatives, such as low medium high
+	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.
Ctrl-C	Indicates a keyboard combination, such as holding down the Control key and pressing C.

Customer Support

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

Accessing SolvNet

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

To access SolvNet:

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

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

Contacting the Synopsys Technical Support Center

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

- Go to the Synopsys [Global Support](#) site on [synopsys.com](#). There you can find e-mail addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNet site or the Synopsys Global Support site and [open a case online](#) (Synopsys user name and password required).

Part 1: Introduction to HSPICE

This manual is organized according to the following Parts:

- [Introduction to HSPICE](#)
- [Elements and Devices](#)
- [Analyses and Simulation](#)
- [Variation and Statistical Analysis](#)
- [Errors-Warnings/Demonstration Files](#)

Part 1 presents the following chapters/topics:

- [Chapter 1, HSPICE Overview](#)
- [Chapter 2, Setup](#)
- [Chapter 3, Startup and Simulation](#)
- [Chapter 4, Multiple Simulations, DP, and HPP](#)
- [Chapter 5, Input Netlist and Data Entry](#)
- [Chapter 6, Using Interactive Mode](#)
- [Chapter 7, HSPICE GUI for Windows](#)
- [Chapter 8, Library and Data Encryption](#)

HSPICE Overview

Describes HSPICE features and the simulation process.

HSPICE ships hundreds of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

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

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

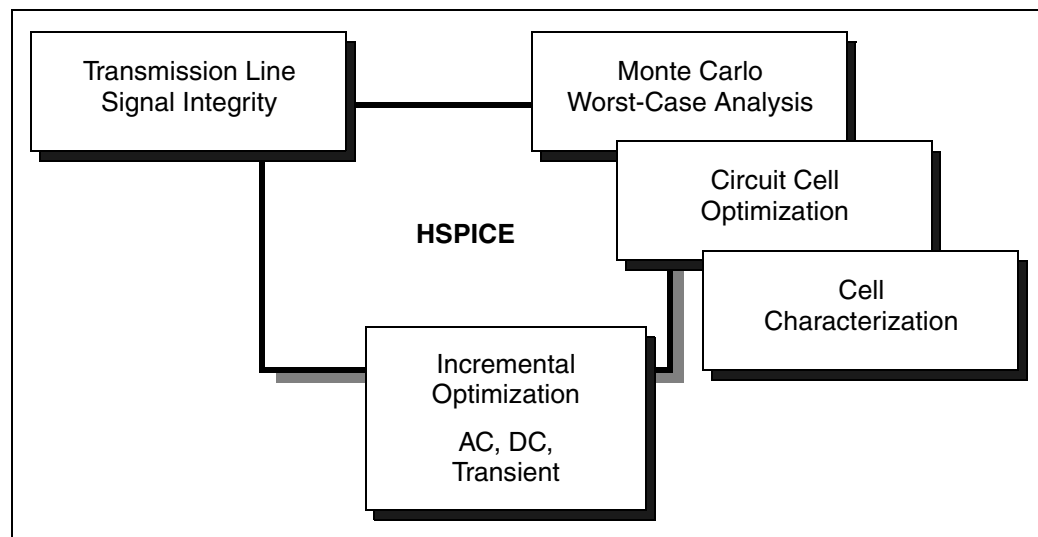


Figure 1 Synopsys HSPICE Design Features

Chapter 1: HSPICE Overview

HSPICE Varieties

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

Only memory can limit the size of the circuits that HSPICE can simulate. As a 32-bit application, HSPICE can address a maximum of 4GB memory on UNIX/Linux depending on your system. While on Windows, HSPICE normally can address a maximum of 2GB memory, or maximum of 3GB memory with Windows large memory mode enabled. For details, consult with Microsoft regarding application memory limitation on Windows.

For a description of commands and options that you can include in your HSPICE netlist, see the [HSPICE and HSPICE RF Netlist Commands](#) and [HSPICE Netlist Simulation Control Options](#) chapters in the *HSPICE Reference Manual: Commands and Control Options*.

These topics are covered in the following sections:

- [HSPICE Varieties](#)
- [Features](#)
- [HSPICE Features for Running Higher-Level Simulations](#)
- [Simulation Structure](#)
- [Parser Syntax Requirements \(Unsupported Formats\)](#)
- [Use of Example Syntax](#)
- [Recommended HSPICE Resources](#)

HSPICE Varieties

Synopsys HSPICE is available in two varieties:

- HSPICE
- HSPICE RF

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

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

HSPICE RF can be launched with either the integrated executable (`hspice`) or in standalone mode (`hspicerf`).

This guide describes all of the features that HSPICE supports. Standalone HSPICE RF supports most — but not all — of these features as well. For descriptions of HSPICE RF features (See [HSPICE User Guide: Advanced Simulation and Analysis](#) for all information relating to HSPICE RF, noise analysis, behavioral modeling, and Verilog-A.

Features

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

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

If you suspend a simulation job (Ctrl-Z), the load sharing facility (LSF) license manager signals HSPICE to release that job's license. This frees the license for another simulation job, or so the stopped job can reclaim the

license and resume. You can also prioritize simulation jobs you submit; LSF automatically suspends low-priority simulation jobs to run high-priority jobs. When the high-priority job completes, LSF releases the license back to the lower-priority job, which resumes at the point of suspension. To resume the LSF job, on the same terminal, type either **fg** or **bg**.

- A number of circuit analysis types (see [Figure 2 on page 4](#)) and device modeling technologies.
- Support for the Compiled Function Library (CFL) function: Enables dynamic linking to HSPICE during run time through a built-in mathematical function or user-defined function written in C. You can include multiple C functions in each library. A general CFL function input argument can come from a predefined parameter value, a mathematical expression of multiple predefined parameter values, a built-in mathematical function in the standard library, or an output of another evaluated CFL function. The CFL function allows users to initialize a data structure and return its address as an input argument of another CFL function on 32-bit and 64-bit machines. CFL is a static function is usable only for parameter evaluations. See [Compiled Function Library Environment Variable](#) to set the required environment variable and file path. See also, [.OPTION CFLFLAG](#) in the *HSPICE Reference Manual: Commands and Control Options*.

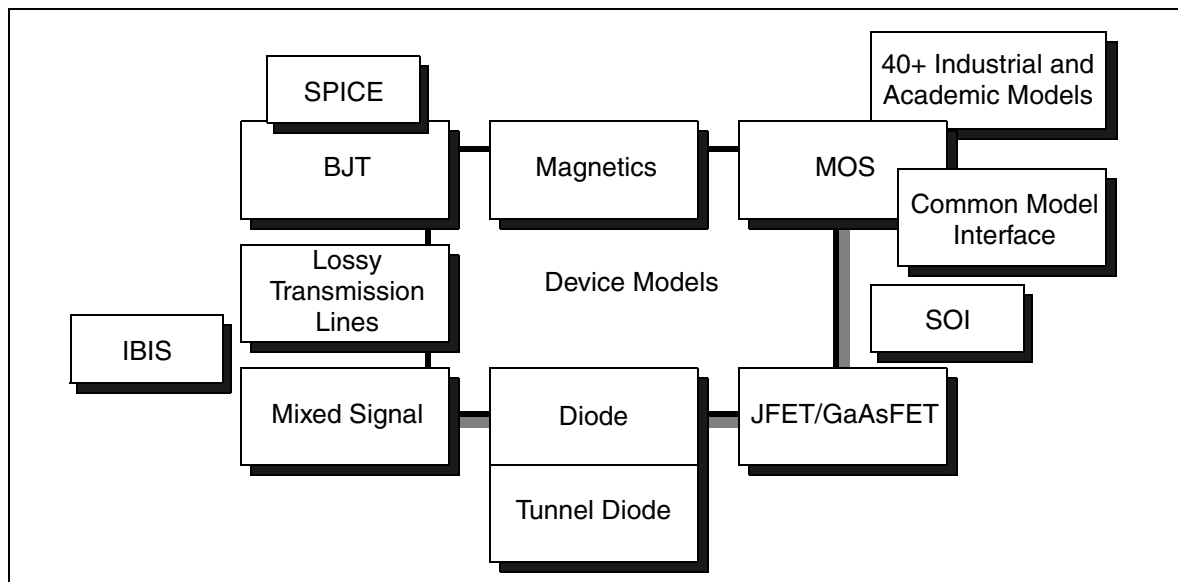


Figure 2 Synopsys HSPICE Modeling Technologies

The following sections introduce these topics:

- [Case Sensitivity](#)
- [Custom CMI](#)
- [TSMC Model Interface \(TMI\)](#)

Case Sensitivity

See [Case Sensitivity on page 69](#). When HSPICE RF runs in integrated executable mode (`hspice`), the case sensitivity feature applies. (When run in standalone mode, HSPICE RF does not support case sensitivity.)

Custom CMI

HSPICE provides the ability to integrate models with the Custom CMI for which HSPICE uses a dynamically linked shared library. Consult your HSPICE technical support team for access to the HSPICE CMI API application note and source code.

TSMC Model Interface (TMI)

HSPICE provides the ability to invoke the TMI flow using proprietary TSMC model files and compiled libraries. Jointly developed by Synopsys and TSMC the TMI technology and API provides a compact model with additional instance parameters and equations for an advanced modeling approach to approach to support TSMC's extension of the standard CMC core model (such as BSIM4) with additional instance parameters and equations for advanced process technology.

- Because Modeling API code is in C, it is available in a compiled format for HSPICE and Synopsys FastSPICE products to link to during the simulation. TMI-settings to invoke the flow and the location of a `.so` file require TSMC model libraries. Use `.OPTION TMIPATH` and `.OPTION TMIFLAG` to

Chapter 1: HSPICE Overview

HSPICE Features for Running Higher-Level Simulations

access these libraries. The simulators enable automatic platform selection on the `.so` file. Both HSPICE and the Synopsys FastSPICE products provide the tool binaries and support the same `*.so` file.

- Use the existing HSPICE and FastSPICE commands to run the simulation.
- The API also performs automatic platform selection on the `.so` file. Both HSPICE and HSIM provide the tool binaries and support the same `*.so` file. Use the existing HSPICE and HSIM commands to run the simulation.
- HSICE supports hybrid simulation with both TMI models and non-TMI models.
- A TMI model parameter, `tmimodel` (available in TSMC's SPICE model parameter libraries), enables you to switch between the built-in model only and the TMI model:
 - `tmimodel = 1` turns on the TMI model
 - `tmimodel = 0` (default) turns on the native/built-in models without calling TMI

(Contact Synopsys Technical Support for further information.) See also the *HSPICE Reference Manual: Commands and Control Options* for [.OPTION TMIFLAG](#) and [.OPTION TMIPATH](#).

HSPICE Features for Running Higher-Level Simulations

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

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

- Explicit include files – `.INCLUDE` statement.
- Implicit include files – `.OPTION SEARCH='lib_directory'`.
- Algebraics and parameters for devices and models – `.PARAM` statement.
- Parameter library files – `.LIB` statement.
- Automatic model selector – `LMIN`, `LMAX`, `WMIN`, and `WMAX` model parameters.

- Parameter sweep – sweep analysis statements.
- Statistical analysis – sweep monte analysis statements.
- Multiple alternative – `.ALTER` statement.
- Automatic measurements `.MEASURE` statement.
- Condition-controlled netlists (`IF-ELSEIF-ELSE-ENDIF` statements (HSPICE only)).

Simulation Structure

The following sections discuss these topics:

- [Experimental Methods Supported by HSPICE](#)
- [Measurement System in HSPICE](#)
- [Simulation Process Overview](#)

Experimental Methods Supported by HSPICE

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

Chapter 1: HSPICE Overview

Simulation Structure

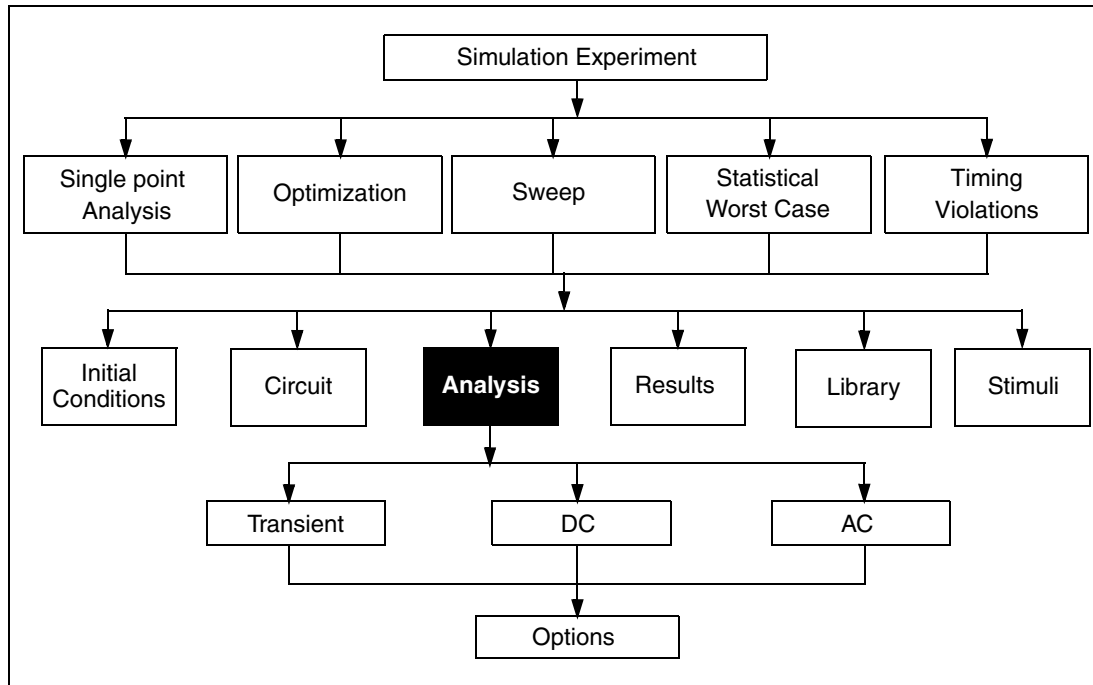


Figure 3 Simulation Program Structure

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

- process
- voltage
- temperature
- parasitics

HSPICE supports two experimental methods:

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

The following are examples of multipoint experiments:

- Process variation – Monte Carlo or worst-case model parameter variation.
- Element variation – Monte Carlo or element parameter sweeps.

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

Measurement System in HSPICE

The measurement system in this manual always refers to MKS units (meter, kilogram, second measurement), unless otherwise stated. HSPICE expects length and width units of meters. But HSPICE does directly support units of “mil” (.001inch, 25.4e-06 meters) as input.

This example defines a transmission line with a length of .4 inches:

```
T1 IN 0 OUT 0 Z0=50 f=1meg L=400mil
```

To get the results you expect, use caution when mixing units of measure.

For reference, some other “m” units can become confusing. Mega, sometimes expressed as “meg” or “x”, may look like “m” (mili):

1m = 1e-3 (mili)

1meg = 1x = 1e6 (mega)

1u = 1e-6 (micro)

Simulation Process Overview

[Figure 4](#) shows the HSPICE simulation process.

Chapter 1: HSPICE Overview

Parser Syntax Requirements (Unsupported Formats)

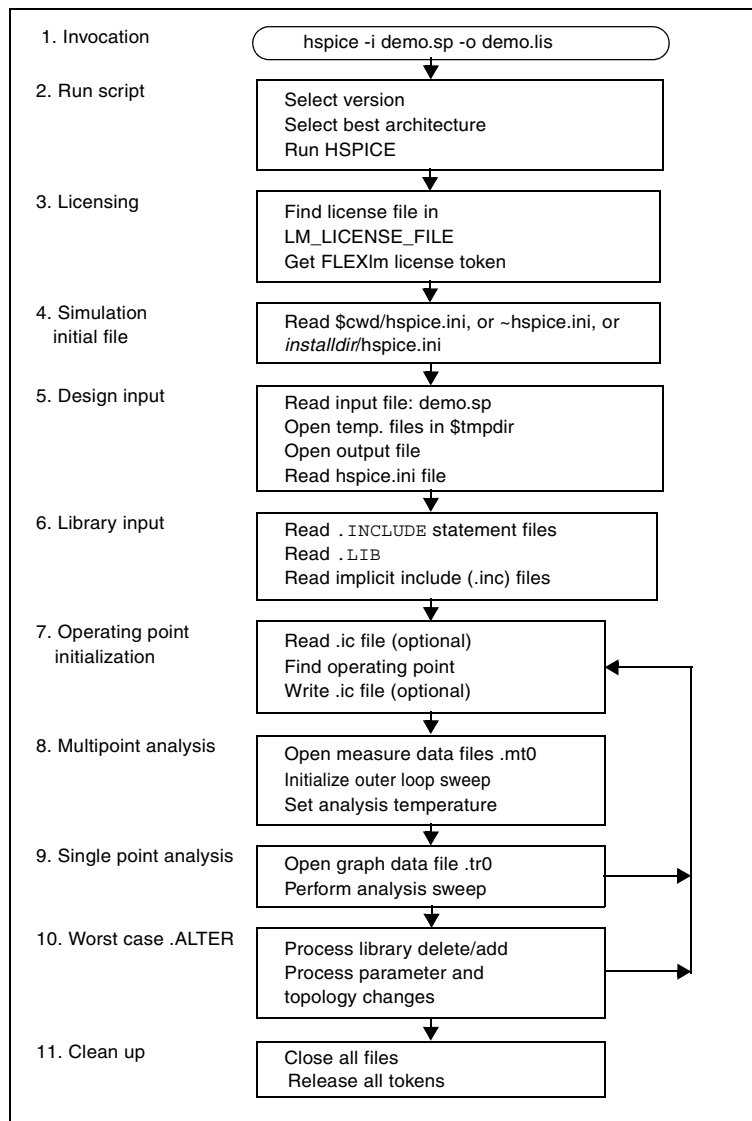


Figure 4 Simulation Process

Parser Syntax Requirements (Unsupported Formats)

Since 2009.03, strict syntax requirements are in effect to satisfy the requirements of HSPICE's more efficient parser.

The following table defines the unsupported formats:

Unsupported format/Rule	Examples	Result
1. Unsupported format/syntax in any line	<code>".temp 55"</code>	Error message issued
2. Unbalanced brackets, quotes	<code>+ P_CGDL'</code>	Error message issued
3. Unbalanced brackets, quotes in one line	<code>.probe v(1</code>	Warning message issued
4. In model definition, model parameter value written with quotes instead of equal sign ("a=20" instead of a=20)	<code>+ "XTIS = 3"</code>	Error message issued
5. Strictly following brackets (parentheses)/equals wherever necessary	<code>.measure dc VT find + par('v(2)*pnorp') + when I1(MAIN) pspvt1 VSOURCE in 0 PWL TD= + 10ps R 0ps</code>	Error message issued (should be:) I1(MAIN)=pspvt1 (should be:) VSOURCE in 0 PWL + TD = 10ps R=0ps
6. Incomplete model parameter definition in model	<code>XW= or + CGDL =</code>	Error message issued
7. Entry name must exactly match in both the .lib call and the .lib definition	<code>.lib 'typical.lib' 'slow' or .lib 'typical.lib' slow ...matches: .lib slow</code>	slow=slow 'slow'='slow' otherwise, error issued

With 2009.03-SP1, the Windows version of HSPICE uses the new parser similar to the Linux version in 2009.03.

See [Reserved Keywords](#) and [HSPICE Parser Strict Syntax Requirements](#) for additional information.

Use of Example Syntax

To copy and paste proven syntax, use the demonstration files shipped with your installation of HSPICE (see [Listing of Demonstration Input Files](#)). Do not attempt to copy and paste from the documentation as you may create unexpected results. Text used in formatting may include hidden characters, white space, etc. for visual clarity.

Recommended HSPICE Resources

In addition to the multiple manuals that cover HSPICE features you can access other material created by HSPICE R&D, CAEs, ACs, and training specialists. See the following resources:

- [Synopsys SolvNet](#)
- [HSPICE Product Website](#)
- [Analog Insights Blog](#)
- [SNUG Presentations and Papers](#)
- [HSPICE Demonstration Netlists](#)

Synopsys SolvNet

A wealth of useful information relating to HSPICE in the form of application notes, articles, troubleshooting, workarounds, recorded trainings, white papers, and time-saving scripts is available through Synopsys SolvNet (<https://solvnet.synopsys.com>).

Sign up for an account if you do not already have one and enter "HSPICE" to search for supplementary information, from getting started to in-depth articles of Synopsys User Group (SNUG) presentations.

To get tool and methodology training:

1. Go to <https://solvnet.synopsys.com/trainingcenter/view.faces3>.
2. Click on **Getting Started with HSPICE**
3. Choose from these modules:
 - Starting HSPICE
 - Netlist file structure

- HSPICE output files
- Using components, sources and subcircuits to construct a netlist
- HSPICE analysis types
- Simulation controls and options

From that same web page you can access training modules that highlight and demonstrate changes to HSPICE and use models for new features and enhancements in recent years.

HSPICE Product Website

Enter www.hspice.com on your Internet browser to view presentations and mini-demos of recent HSPICE features.

This website provides links to:

- Access HSPICE white papers
- Download a library of HSPICE compatible Verilog-A modules
- Sign up for HSPICE feature webinars
- Learn about product feature including integration with Custom Designer

Analog Insights Blog

Synopsys presents valuable technical discussions on its interactive blog site at: <http://blogs.synopsys.com/analoginsights/>

HSPICE users can read and participate in discussions, use model tips and go to links of other useful sites. This blog is a in a lively collection of recent posts, Q&A's, and reports from the field moderated moderated by the Staff Technical Marketing Manager for circuit simulation products.

SNUG Presentations and Papers

Synopsys User Group events are held in multiple cities and countries each year. The proceedings include tutorials, presentations, and papers written by HSPICE product CAEs, ACs, and customers. You may be able to use some of this material in projects that you are working on with similar content. Go to: <http://www.synopsys.com/Community/SNUG/Pages/snugresults.aspx?qry=HSPICE>

Articles and presentations require a SolvNet password.

HSPICE Demonstration Netlists

HSPICE delivers nearly 350 demonstration files with each release. These demo netlists, many with "readme" files, illustrate use of the HSPICE commands and control options for multiple applications. Find the directories for

Chapter 1: HSPICE Overview

Recommended HSPICE Resources

HSPICE and HSPICE RF examples at:

`$installdir/demo/hspice/` or

`$installdir/demo/hspicerf/`

For a full listing of the demo file categories and brief descriptions of the netlists, go to [Listing of Demonstration Input Files](#) in this manual (for HSPICE examples) or, for RF examples, [RF Demonstration Input Files](#) in the *HSPICE User Guide: Advanced Analog Simulation and Analysis*.

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

HSPICE ships hundreds of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

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

These sections discuss the following topics:

- [Setting Environment Variables](#)
- [Standard Input Files](#)
- [Standard Output Files](#)
- [Working Directory Path Character Limit](#)
- [Continuing HSPICE Simulations after a Power Down](#)

Setting Environment Variables

The following sections discuss these topics:

- [License Variables](#)
- [Temporary Directory Variable](#)
- [Windows Variables](#)
- [Shared Libraries Environment Variable](#)
- [Compiled Function Library Environment Variable](#)
- [Using Environment Variables as Include Statements](#)

- [Setting Environment for 64-bit HSPICE](#)
- [Setting Distributed Processing Environment Variable](#)
- [Verilog-A \(pVA\) Environment Variables](#)

License Variables

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

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

```
setenv LM_LICENSE_FILE port@hostname
```
- If you are using the Bash or Bourne shell, add these lines to the `.bashrc` or `.profile` file:

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

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

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

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

The following sections discuss:

- [License Queuing Variable](#)
- [Controlling the License Queueing Interval](#)
- [License Server Down Iterations](#)
- [Using the FlexLM MAX Option](#)
- [HSPICE License Schedule \(G-2012.06\)](#)
- [Limiting the Number of Licenses for Your Group](#)

License Queuing Variable

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

Setting the `META_QUEUE` environment variable to 1 enables queuing of HSPICE/HSPICE RF licenses:

```
setenv META_QUEUE 1
```

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

You can create additional pools by having multiple `INCREMENT` lines in the same license file, such as exist in the case of separate tool purchases or off-maintenance keys.

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

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

Controlling the License Queuing Interval

Setting the environment variable `META_QUEUE` causes HSPICE to wait for an available license from the license server. If no license becomes available, HSPICE waits indefinitely. There may be times where it is desirable to abandon the queuing attempt, such as during scripted operations.

Since the 2008.03-SP1 release, you can set the environment variable: `META_QUEUE_TIME`. The argument to `META_QUEUE_TIME` is in seconds.
Example:

Chapter 2: Setup

Setting Environment Variables

```
#!/bin/csh -f
setenv META_QUEUE 1
setenv META_QUEUE_TIME 3600
setenv LM_LICENSE_FILE 27000@corp_lic_server
hspice -i myjob.sp -o outfile
```

Here, `META_QUEUE` enables HSPICE to wait for an available license. Next, it sets `META_QUEUE_TIME` to 3600 seconds. The license request de-queues if no license is available in 3600 seconds (60 minutes), and the script can continue. This feature is available on all platforms.

License Server Down Iterations

The default behavior when the license server is down is to try to reconnecting to the license server indefinitely, which is in conformance with other Synopsys tools.

For backward compatibility, you can use the environment variable `HSPICE_LIC_EXIT`. When set, for example, in cshell, using the command `setenv HSPICE_LIC_EXIT 1`, HSPICE exits after 60 tries to reconnect.

A limited number of retries is available in advanced client-server mode (`-CC`).

Using the FlexLM MAX Option

With the F-2011.09 release, HSPICE supports the MAX option if your group uses `META_QUEUE`. If `META_QUEUE` is on, when a user exceeds the MAX option value then the job is queued instead of terminated.

HSPICE License Schedule (G-2012.06)

Number of Cores	1-2	3-4	5-6	7-8	9-10	11-12	13-14	15-16
Basic Analyses incl. HPP	1	2	3	4	5	6	7	8
Advanced Analyses ¹	2	3	4	5	6	7	8	9

1. *Harmonic balance, Shooting Newton, Transient Noise, StatEye*

Limiting the Number of Licenses for Your Group

You can create a license option file named `snpslmd.opt` in the directory where the license file is located. In the example file below:


```
HOST_GROUP HSPICEGR 1.160.10.240 238.17.159.4  
MAX 2 hspice HOST_GROUP HSPICEGR
```

.... the first line creates a group named HSPICEGR followed by a list of the IP addresses of the machines in that group. The second line defines (for the license server) that the group is allowed a maximum of two HSPICE licenses.

You can change the name of the file and its directory. The option file also allows you to reserve licenses for a group and create groups by user name instead of IP address. For more information about license option files, see the *Administration_Guide.pdf* (installed with the Synopsys Common Licensing product).

Temporary Directory Variable

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

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

Windows Variables

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

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

You may encounter the licensing prompt either when using non-administrator priority installed HSPICE or even after correctly setting the `LM_LICENSE_FILE` or `SNPS_LICENSE_FILE`. If this occurs, set the `FLEXLM_BATCH` environment variable to 1 as the user environment to disable the pop-up. For details on licensing management and control, scroll down to **Documentation > FLEXnet User Manual** on the Synopsys license supporting

website: <http://www.synopsys.com/Support/Licensing/Licensing/Pages/default.aspx>

Environment Variables in Windows HSPUI

There are a number of environment variables which are useful in HSPICE but if you add or change one in Windows, you must close and reopen the application for the variable to take effect. HSPUI retrieves the current environment variables on startup and does not have a mechanism to dynamically update environment variables.

Shared Libraries Environment Variable

If you get an “error while loading shared libraries” message, it is due to an OS issue, relating to a missing library on your machine. For example, if you are using an hspice 64-bit machine, verify that the library exists in the `/usr/lib` directory. Depending on the HSPICE platform you are using, hspice 32-bit (`usr/lib`) or hspice-64-bit (`usr/lib64`) shared libraries may require the setting of the following environment variable:

```
setenv LD_LIBRARY_PATH /usr/lib:/usr/lib64
```

Verify that your OS is compatible with the Synopsys Platforms recommendations given in URL:

<http://www.synopsys.com/Support/Licensing/SupportPlatform/ReleaseSupport/Pages/default.aspx>

Compiled Function Library Environment Variable

Set the CFL environment variable and the specified path and `*.so` file as follows:

```
setenv CFL_COMPILED_LIB CFL library_file_name
```

For example:

```
% setenv CFL_COMPILED_LIB /path/libcfl.so
```

See [Features](#) in Chapter 1 for a discussion of the CFL capability.

Using Environment Variables as Include Statements

To use environment variables as include statements:

1. Set an environment variable to the name of the file that you want to include.

```
% setenv MY_OWN_INCLUDE_FILE inv.inc
```

2. Use the environment variable in the netlist:

```
*  
.inc '$MY_OWN_INCLUDE_FILE' $ include the file inv.inc  
.option list  
.op  
X1 in out inv  
c1 out 0 0.1p  
.global vdd gnd  
.model n nmos level= 49  
.model p pmos level= 49  
vdd vdd 0 5  
vin in 0 pulse 0 5 0 1n 1n 5n 10n  
.tran 10p 100n  
.option post=2  
.end
```

Result: HSPICE includes the file defined by the environment variable in the netlist.

Setting Environment for 64-bit HSPICE

You can control selection of the 64 bit HSPICE binary by setting an environment variable, `HSPICE_64`. When you run the wrapper script for “hspice”, if it detects the existence of this variable it automatically selects the 64 bit version of the program.

Note: Always invoke HSPICE from the bin directory which uses the wrapper script and performs important functions for setting the HSPICE environment.

For example, enter:

```
% setenv HSPICE_64 1  
% hspice -v HSPICE Version D-2010 64-BIT  
% unsetenv HSPICE_64  
% hspice -v HSPICE Version C-2009.09 32-BIT
```

Setting Distributed Processing Environment Variable

Only if you run the HSPICE binary directly, set the following environment variable before invoking DP:

```
setenv CDPL_HOME $INSTALLDIR/hspice/cdpl
```

For Synopsys distributed processing, the HSPICE wrapper sets up `$CDPL_HOME` automatically. The Synopsys Common Distributed Processing Library (CDPL) provides the engine for the `-dp` command-line invocation. The `-mp` (multiprocessing) calls are interchangeable with DP.

Verilog-A (pVA) Environment Variables

The pVA library allows both absolute path and relative paths in `.hdl` statements; alternatively, you can use environment variables. For example:

```
.hdl "/tmp/design01/lib/INV_linux4020110.pvalib"  
.hdl "${PVA_PATH}/${LIBNAME}_${PLATFORM}4020110.pvalib"  
.hdl "../../lib/${PVA_LIBRARY}.pvalib"
```

Where: You can define `${PVA_PATH}`, `${LIBNAME}`, `${PLATFORM}`, and `${PVA_LIBRARY}` using the `setenv` UNIX command.

In addition, to prevent re-compiling, you can set the `PVA_DIR` environment variable to the name of the directory that you want to share the compiled Verilog-A modules. In the following example, the environment variable redirects `*.pvadir` to directory `${PVA_DIR}.pvadir`.

```
setenv PVA_DIR $abs|relative_path/xyz
```

Standard Input Files

This section describes the standard input files to HSPICE.

The following sections discuss these topics:

- [Design and File Naming Conventions](#)
- [Initialization File \(hspice.ini\)](#)
- [DC Operating Point Initial Conditions File](#)
- [Input Netlist File](#)

- [Library Input File](#)
- [Analog Transition Data File](#)

Design and File Naming Conventions

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

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

HSPICE extracts the design name from their input files, and performs actions based on that name. HSPICE reads and writes files related to the current circuit design. Files related to a design usually reside in one directory. The output file is stdout on UNIX platforms, which you can redirect. [Table 1](#) lists input file types, and their standard names. The sections that follow describe these files.

Table 1 Input Files

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

Initialization File (hspice.ini)

The initialization file enables you to specify user defaults. If HSPICE reads one `hspice.ini` file, HSPICE includes its contents at the top of the input file. (This

does not apply to HSPICE RF). All HSPICE simulations look for ONE implicit `hspice.ini` file. The HSPICE default search order for the `hspice.ini` file is:

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

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

DC Operating Point Initial Conditions File

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

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

Input Netlist File

The input netlist file, `design.sp`, contains the design netlist. Optionally, it can also contain statements that specify the type of analysis to run, type of output desired, and what library to use.

Library Input File

You use `library_name` files to identify libraries and macros that need inclusion for simulating `design.sp`.

Analog Transition Data File

When you run HSPICE in standalone mode, a `design.d2a` file contains state information for a U-element mixed-mode simulation.

Standard Output Files

This section describes the standard output files from HSPICE. [Table 2](#) lists the various types of output files produced. For information about the standard output file from HSPICE RF, see [HSPICE RF Output File Types](#) in the *HSPICE User Guide: Advanced Analog Simulation and Analysis*.

Table 2 HSPICE Output Files and Extensions

Output File Type	Extension
AC analysis measurement results	.ma# ¹
AC analysis results (from .POST statement)	.ac#
Monte Carlo results	.mc#
Data mining results	.mpp0
DC analysis measurement results	.ms#
DC analysis results (from .POST statement)	.sw#
Digital output	.a2d
FFT analysis graph data (from FFT statement)	.ft#
Hardcopy graph data (from meta.cfg PRTDEFAULT)	.gr# ²
Operating point information (from .OPTION OPFILE statement)	.dp#
Operating point node voltages (initial conditions)	.ic#
Output listing	.lis, or user-specified
Output status	.st#
Output tables (from .DCMATCH OUTVAR statement)	.dm#
Subcircuit cross-listing	.pa#
Transient analysis measurement results	.mt#
Transient analysis results (from .POST statement)	.tr#

Table 2 HSPICE Output Files and Extensions

Output File Type	Extension
Waveform viewing files from .OPTION WDF argument for use with Synopsys WaveView/SX tools	*_wdf.tr#, *_wdf.sw#, or *_wdf.ac#

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

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

AC Analysis Results File

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

AC Analysis Measurement Results File

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

DC Analysis Results File

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

DC Analysis Measurement Results File

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

FFT Analysis Graph Data File

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

Operating Point Information File

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

Operating Point Node Voltages File

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

Output Listing File

The output listing is a text file. You can name it *output_file* (no file extension), *output_file .lis*, or a file extension that you specify, depending on which format you use to start the simulation.

The output file includes the following information:

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

Chapter 2: Setup

Standard Output Files

- Actual control option values that HSPICE uses for the present simulation (useful when options such as RUNLVL override user-set values.)
- Details of the volt drop, current, and power for each source and subcircuit.
- Results of a `.PRINT` statement.
- Results of the `.OPTION` statements.
- Total CPU time (the sum of op point, transient, readin, errchk, setup, and output times).
- In the following snippet of a `*.lis` file, you can see that the total cpu time is the sum of op point, transient, readin, errchk, setup and output analysis times.

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

The different analyses stand for time required to:

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

Output Status File

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

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

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

Output Tables

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

Subcircuit Cross-Listing File

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

Transient Analysis Measurement Results File

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

Transient Analysis Results File

Both HSPICE and HSPICE RF place the results of transient analysis in file *output_file.tr#*, where # is 0-9999, which you set in the `-n` command-

Chapter 2: Setup

Working Directory Path Character Limit

line argument. This file lists the numerical results of transient analysis. A `.TRAN` statement in the input file, together with an `.OPTION POST` statement, creates this post-analysis file. If the input file includes an `.OPTION POST` statement, then the output file contains simulation output suitable for a waveform display tool.

Waveform Viewing File

After you use `.OPTION WDF` for transient, DC, or AC analyses, for the WDF waveform file, HSPICE automatically appends `_wdf` into the output file root name to specify that it is in WDF format. The file names appear as:

`*_wdf.tr#`, `*_wdf.sw#`, or `*_wdf.ac#`.

For example, HSPICE names the WDF waveform output file `design_wdf.tr0`.

Working Directory Path Character Limit

HSPICE has a limitation on the number of characters in a path name plus a file name of 1024 characters. For example:

```
hspice -i path_name/input_file -o out_file
```

When specifying a path and file name using `-i` or `-o`, the length must be 1024 characters or fewer. If the working directory path is greater than 1024 characters, HSPICE aborts with a signal 11 error.

To check the length of the working directory path, use the UNIX command:

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

Workaround

Because this can be an issue for automated programs that create pathnames based on appending long design and cell names, there is a workaround. Users on Linux/UNIX platforms can create a soft link in their local design directory to the file at the end of the long path.

```
% ln -s /long/directory/path/target.inc target.inc
```

Then, include the link instead of the actual file:

```
.inc target.inc
```

This way, the file created by the automated program does not require relocation.

Continuing HSPICE Simulations after a Power Down

When a power down occurs, one of the following situations can occur:

1. The license server shuts down.
2. The machine running the HSPICE job shuts down.
3. Both the license server and the machine running the HSPICE job shuts down.

For case 1, if the power down happens only to the license server, then HSPICE gives you a 30-minute grace period to reconnect to the license server. If power is restored and the license server is up within 30 minutes, then any interrupted HSPICE jobs start up from where they stopped.

For cases 2 and 3, if the machine running the HSPICE jobs power down, then you can make use of `.STORE` and `-restore` features in HSPICE.

The `.STORE` command can save the simulation state to a file at a certain time or set of times.

For more information of using the `.STORE` command, see [Storing and Restoring Checkpoint Files](#).

Chapter 2: Setup

Continuing HSPICE Simulations after a Power Down

Startup and Simulation

Describes the invocation commands, and simulation modes.

HSPICE ships numerous examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

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

For simulation involving multiple cores, distributed processing, multithreading and HSPICE Precision Parallel (HPP), see [Chapter 4, Multiple Simulations, DP, and HPP](#).

The following sections discuss these topics:

- [Running HSPICE Simulations](#)
- [Using Isomorphic Analyses in Subckt Blocks](#)
- [Running HSPICE Simulations on Windows](#)
- [Running HSPICE RF Simulations](#)
- [Running HSPICE Interactively](#)
- [Using HSPICE in Client-Server Mode](#)
- [Running HSPICE to Calculate New Measurements](#)

Running HSPICE Simulations

Use the following syntax to start HSPICE:

```
hspice
[-i path/input_file]
[-o path/output_file] [-n number]
[-html path/html_file] [-gz] [-d]
[-C path/input_file] [-CC path/input_file] [-I] [-K]
[-L command_file] [-S] [-case 0|1]
[-datamining -i datamining.cfg [-o outname]
[-dp [#num] [-dpconfig dp_configuration_file] [-merge]]
[-mp process_count] [-mt thread_count] [-hpp]
[-meas measure_file] [-mrasim [0|1|2|3]] [-top subcktname]
[-restore checkpoint_file]
[-hdl file_name] [-hdlpath pathname]
[-vamodel name] [-vamodel name2...]
[-help] [-doc] [-h] [-v]
```

For a description of the `hspice` command syntax and arguments, see [hspice](#) in the *HSPICE Reference Manual: Commands and Control Options*. For multiple processing, multithreading, distributed processing, and HSPICE Parallel Process features, see [Chapter 4, Multiple Simulations, DP, and HPP](#).

HSPICE provides a quick demo file for a simple LRC circuit to test your installation (see `demo.sp` under [Benchmark Examples](#) in this user guide).

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

1. Invocation.

For example, at the shell prompt, enter:

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

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

2. Script execution.

The `hspice` shell command starts the HSPICE executable from the appropriate architecture (machine type) directory. The UNIX `run` script launches a HSPICE simulation. This procedure establishes the environment

for the HSPICE executable. The script prompts for information, such as the platform that you are using, and the version of HSPICE to run. (When you install HSPICE, available versions are pre-determined.)

3. Licensing.

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

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

4. Simulation configuration.

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

5. Design input.

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

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

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

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

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

Chapter 3: Startup and Simulation

Using Isomorphic Analyses in Subckt Blocks

6. Library input.

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

7. Operating point initialization.

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

8. Analysis.

HSPICE can perform a single or multipoint sweep of the design and produce one set of output files. In the Step 5 example above, the `.TRAN` statement causes HSPICE to perform a multipoint transient sweep analysis for 20ns for temperatures that range from -55° C to 75° C, in steps of 10° C.

9. Worst-case `.ALTER`.

You can vary simulation conditions, and repeat the specified single or multipoint analysis. The above example changes `CLOAD` from 0.75 pF to 1.5 pF, and repeats the multipoint transient analysis. You can activate multi-processing while running `.ALTER` cases by entering `hspice -dp` or `-mp` on the command line.

10. Suspending a simulation

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

11. Normal termination.

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

Using Isomorphic Analyses in Subckt Blocks

The isomorphic analyses feature enables you to run unrelated analyses (`.DC`, `.AC`, and `.TRAN`) many times during a simulation by grouping the set of analyses into a subcircuit, which performs multiple analyses in one simulation with calls to the subcircuit. The usage model is: Specify the analyses commands within the subckt definition block and then instantiate the subckt to

perform the analyses. Each call of the subcircuit is an individual analysis with its own set of parameters.

The syntax is as follows:

```
.subckt analyses_sb [start=p1 stop=p2 steps=p3]
.DC ...
.AC ...
.TRAN ...
.ends analyses_sb
```

...followed by the analysis call:

```
x1 analyses_sb [start=a1] [stop=a2] [steps=a3]
x2 analyses_sb [start=b1] [stop=b2] [steps=b3]
```

For information on the available analyses,

See also:

- [Chapter 13, Initializing DC-Operating Point Analysis](#)
- *.DC* in the *HSPICE Reference Manual: Commands and Control Options*
- [AC Small-Signal and Noise Analysis](#)
- *.AC* in the *HSPICE Reference Manual: Commands and Control Options*
- [Chapter 15, Transient Analysis](#)
- *.TRAN* in the *HSPICE Reference Manual: Commands and Control Options*

Data Inputs and Outputs

Input is in the form of a subckt block call. You can specify the different analyses inside the subckt and then call the subckt. Each call performs all the specified analyses in the subckt.

You generate output files based on the analyses specified. The suffix number increases for every “same type” of analysis.

For example:

```
.subckt analyses_sb start_dc=-25 stop_dc=25 steps_dc=5
+ steps_tran=1n stop_tran=10n
.DC TEMP start_dc stop_dc steps_dc
.TRAN steps_tran stop_tran
ends analyses_sb
...
x1 analyses_sb start_dc=25 stop_dc=75 steps_dc=10
x2 analyses_sb steps_tran=2n
```

This example creates the output files: *.sw0,*.sw1 and *.tr0,*.tr1.

Isomorphic Analyses Example

The following example shows how to specify different analyses within the subckt block.

```
.subckt analyses_sb start_dc=-25 stop_dc=25 steps_dc=5  
+ steps_tran=1n stop_tran=10n  
.DC TEMP start_dc stop_dc steps_dc  
.TRAN steps_tran stop_tran  
.ends analyses_sb
```

This example specifies both .DC and .TRAN analyses within the subckt. To invoke these analyses you can call the subckts.

```
x1 analyses_sb start_dc=25 stop_dc=75 steps_dc=10  
x2 analyses_sb steps_tran=2n  
x3 analyses_sb
```

- Each subckt call performs DC and Transient analysis.
- Parameters defined in the subcircuit calls override the default values in the subcircuit definition.
- If the subckt calls do not define the parameters they take the default values given in the subcircuit.

Limitations

The subcircuit block only supports DC, AC, and transient analyses inside it.

Running HSPICE Simulations on Windows

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

For example:

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

```
c:\synopsys\Hspice_release_version\bin\hspice
```

4. Or type the following command to run a simulation:

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

Running HSPICE RF Simulations

Use the following syntax to invoke HSPICE RF:

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

For a description of the `hspicerf` command syntax and arguments, see [hspicerf](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Running HSPICE Interactively

For a full discussion, refer to [Chapter 6, Using Interactive Mode](#). Interactive mode enables you to use these HSPICE commands at the HSPICE prompt to help you simulate circuits interactively:

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

The following sections discuss these topics:

- [Starting Interactive Mode](#)
- [Running a Command File in Interactive Mode](#)
- [Quitting Interactive Mode](#)

Starting Interactive Mode

To invoke the interactive mode, enter:

```
hspice -I
```

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

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

```
hspice> save command filename
```

Running a Command File in Interactive Mode

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

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

Quitting Interactive Mode

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

```
hspice> quit
```

Using HSPICE in Client-Server Mode

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

Launching the Advanced Client-Server Mode (-CC).)

Table 3 Supported Platforms

Linux RHEL	Linux SUSE	Sun/Solaris	Windows
Yes	Yes	Yes	Yes

The following sections discuss these topics:

- [To Start Client-Server Mode](#)
- [To Simulate a Netlist in Client-Server Mode](#)
- [To Quit Client-Server Mode](#)
- [Launching the Advanced Client-Server Mode \(-CC\)](#)

To Start Client-Server Mode

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

```
hspice -C
```

Server

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

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

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

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

After you create the server, it automatically runs in the background. If the server does not receive any request from a client for one hour, the server releases the license and exits automatically.

Client

The client can send a request to the server to determine the HSPICE license status, or to kill the server.

- If the request is to check the license status, the server checks, and replies to the client. The syntax of this request is:

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

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

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

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

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

```
% hspice -C casename.sp output_file
```

Beginning in HSPICE version E-2010.12 the following message is printed in the `*.lis` file when you are running a job in client-server mode:

```
Connect to server hostname:port
```

To Simulate a Netlist in Client-Server Mode

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

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


Note: This mode also supports other HSPICE command line options. For a description of the options shown, see [hspice](#) in the *HSPICE Reference Manual: Commands and Control Options*.

To Quit Client-Server Mode

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

```
hspice -C -K
```

Launching the Advanced Client-Server Mode (-CC)

The Advanced Client/Server Mode provides an efficient interface for cell characterization applications and allows for multiprocessing if the given case contains .Alter, Tran Sweep, or Monte Carlo analyses. The advanced C/S mode facilitates the Client/Server mode as follows:

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

Note: To set the environment to enable netlist echoing in `-CC` mode, enter:

```
setenv HSP_LIS_201012
```

Note: The Client user ID should be same as the user ID which started the server.

Table 4 Supported Platforms

Linux RHEL	Linux SUSE	Sun-Solaris	Windows
Yes	Yes	Yes	No

The following sections present these topics:

- [Advanced Client-Server Command Syntax](#)
- [Application Instances](#)

Advanced Client-Server Command Syntax

These commands start the HSPICE server, do simulations including multiprocessing, and stop the server. The tables that follow describe the arguments.

1. To start the server, enter:

```
hspice -CC [-share inc_file] [-port hostname:port_num]
+ [-mp [process_count]] [-o output]
+ [-stop idle_time]
```

2. To begin a simulation, enter:

```
hspice -CC input_file [-port hostname:port_num]
+ [-o output_file]
```

Explanation: The port information must come *after* the input file information on the command line.

3. To stop the HSPICE server, enter:

```
hspice -CC -K [-port hostname:port_num]
```

Argument	Description
-CC	Launches the advanced HSPICE C/S mode. After the server starts, it runs in background.
-port hostname:port_num	Starts server on the designated port hostname:port_num. If you do not specify this argument, it uses the default port number (25001). If the default port is not available, HSPICE chooses any free port. When you start the server or run a case with -port hostname:port_num, it ignores the hostname. But you can stop the server with -port hostname:port_num to implement remote control. HSPICE prints out port information to the screen.
-share inc_file	Specifies a common file name shared by different circuits.

Argument	Description
-mp [<i>process_count</i>]	On UNIX platforms, option -mp [<i>process_count</i>] triggers a multi process simulation when the server receives a netlist with .alter and sweep analyses. The optional <i>process_count</i> is a nonzero integer, (the initial value of available child process number to fork). If you do not set <i>process_count</i> , then HSPICE uses the CPU number of the server as the default. On the client side, you do not need to specify -mp. See Running Multithread/Multiprocess HSPICE Simulations on page 57 for details and limitations.
-stop idle_time	The idle time unit=hour(s). "idle_time" means that the server has not received any client request. If you do not specify an "idle_time" the server quits automatically after the default idle_time of 1 hour.
-o output_file	Specifies the output file name. If you do not specify an "output_file" i, HSPICE uses the shared file root filename as the output file root filename. If there is no shared file, no output file is generated.
-K	Shuts down the client server.

Application Instances

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

t1.sp

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

t2.sp

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

...

Chapter 3: Startup and Simulation

Using HSPICE in Client-Server Mode

t5.sp

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

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

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

Explanation: Since the .inc "/home1/user1/test/model/model_file" statement appears in each netlist, HSPICE does not read it in again because the server has already processed the information.

3. Repeat Step 2 until you simulate all cases.
4. Exit the HSPICE Client/Server mode.
hspice -CC -K

The sequence of commands is:

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

Notes

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

```
#!/usr/bin/perl
##Start server without designated port, redirect output
##information
$cmd = "hspice -CC >& log ";
system($cmd);

##Get the port_num on which server is started
open (IN,"log");
while (<IN>) {
if ($_ =~ "started on") {
$portnum=$_; }
}
close (IN);

##Stop server
$cmdn ="hspice -CC -K -port $portnum";
system($cmdn);
```

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

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

Chapter 3: Startup and Simulation

Running HSPICE to Calculate New Measurements

```
system($cmdn) ;
```

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

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

Each of the netlists includes .inc "/home1/tom/test/model/model_file" In every case, the absolute path name of the common file in .inc "/home1/user1/test/model/model_file"... is the same as the absolute path name of the common file specified by -share /home1/user1/test/model/model_file.

The common file /home1/user1/test/model/model_file will only be read in once and .inc "/home1/user1/test/model/model_file" will be ignored in every case.

Running HSPICE to Calculate New Measurements

To calculate new measurements from previous simulation results produced by HSPICE, you can rerun HSPICE.

To get new measurements from a previous simulation, enter:

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

For a description of the options shown, see [hspice](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Multiple Simulations, DP, and HPP

Describes the MultiProcess, MultiThread, Distributed Processing (DP), and HSPICE Precision Parallel (HPP) simulation modes available in HSPICE.

HSPICE ships numerous examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

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

The following sections discuss these topics:

- [Running Multi Simulations](#)
- [Running Distributed Processing \(DP\) on a Network Grid](#)
- [Running Multithread/Multiprocess HSPICE Simulations](#)
- [HSPICE Precision Parallel \(-hpp\)](#)

Running Multi Simulations

Use the following syntax to start up HSPICE depending on the type of multiple or parallel processes available to your site:

```
hspice
[-dp [#num] [-dpconfig dp_configuration_file]
+ [-dplocation NFS|TMP] [-merge] [-dpgui]]
+ [-mp process_count] [-mt thread_count] [-hpp]
```

For a description of the `hspice` command syntax and arguments, see [hspice](#) in the *HSPICE Reference Manual: Commands and Control Options*.

You can run multiple simulation jobs on the Linux and Windows platforms, but neither HSPICE Precision Parallel (-hpp) nor distributed processing (-dp) is currently available on the Windows platform.

Multi-Jobs on the Windows HSPICE GUI

For information on running batch file jobs from the HSPICE GUI, go to [Running Multiple Simulations](#) in [Chapter 7, HSPICE GUI for Windows](#).

Running Distributed Processing (DP) on a Network Grid

You can submit an HSPICE simulation job to a network of machines for parallel processing on the following network grids: SGE, LSF, RSH, and SSH. These machine cluster/compute farm manager modes are controlled by the `dp_configuration_file`.

Distributed processing uses the Synopsys Common Distributed Processing Library (CDPL). For Synopsys distributed processing, the HSPICE wrapper sets up `$CDPL_HOME` automatically. If you run the HSPICE binary directly, set the following environment variable before invoking DP:

```
setenv CDPL_HOME $INSTALLDIR/hspice/cdpl
```

For more information on using CDPL, See `$installdir/hspice/cdpl/doc/CDPLUsersManual.pdf`

The supported platforms (for both 32-bit and 64-bit) are:

Linux RHEL	Linux SUSE	Solaris	Windows
Yes	Yes	Yes	No

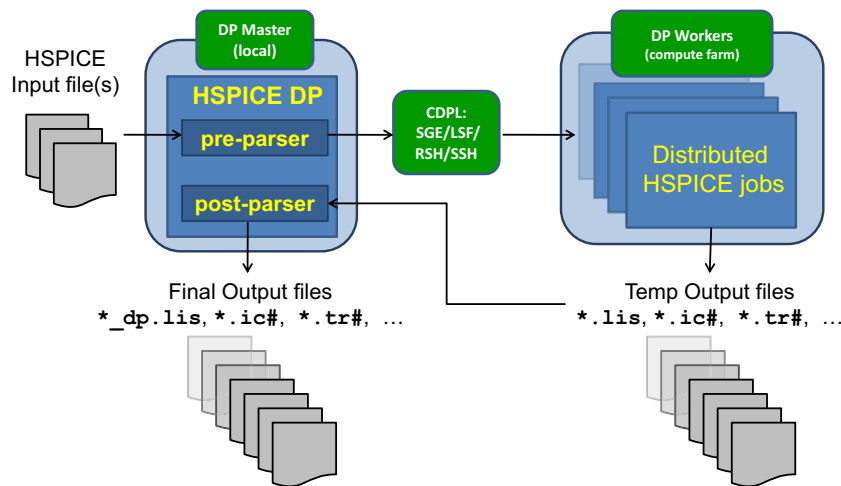
Typical configuration files have the following forms, depending on the type of network grid:

	#flag	hostname	slots	tmpdir	protocol	command
SGE	1	arch=glinus	-1	/tmp	SGE	qsub -P bnormal -V -cwd -l
LSF	1		-1	/tmp	LSF	bsub -q bnormal -R arch=glinux
RSH	1	rhas74	1	/tmp	RSH	rsh
	1	rhas75	3	/tmp	RSH	rsh

	#flag	hostname	slots	tmpdir	protocol	command
SSH	1	rhas78	2	/tmp	SSH	ssh
	1	rhas79	2	/tmp	SSH	ssh

where: the **flag** column posts the value “1” if the host can be used and “0” if it cannot. The **slots** column notes the number of workers that can be started on the particular host and has a value of “-1” for grid types **LSF** and **SGE**.

The following is the illustration of DP flow:



CDPL flow terminology:

- An HSPICE job starts on the *Master* (local machine).
- *Workers* may be on the same or different hosts than the Master.
- Each worker is assigned *tasks*.
- Simulation is performed in parallel but the initial parse and final file merge/summary steps are executed in scalar mode.

The target simulation job is a netlist which uses either:

- `.ALTER` blocks
- Monte Carlo analysis

Chapter 4: Multiple Simulations, DP, and HPP

Running Distributed Processing (DP) on a Network Grid

- Data block
- Parametric sweep

Using distributed processing reduces run times for sequential simulations such as Monte Carlo, Alter, Sweep with no loss in accuracy. For example, solving 1 million Monte Carlo data points takes approximately the same time as 10 thousand Monte Carlo points (plus the time for results integration) when 100 systems are available.

For more information on Monte Carlo analysis, see [Chapter 22, Monte Carlo—Traditional Flow Statistical Analysis](#).

Starting DP on the HSPICE Command-Line

Invoke DP using the `-dp` command as follows:

```
hspice -i input_file -o [output_file] -dp [#num ]/  
[-dpconfig dp_configuration_file] [-dplocation NFS/TMP] [-merge]  
[-dpgui]
```

where:

- `-dp [#num]` Invokes DP and specifies the number of workers. The workers can be distributed on one multiple core machine or multiple machines across the network. If you are running `-dp` on one multiple core machine, `#num` cannot be greater than the core count of the machine. If you do not specify `#num`, then DP defaults to the core count of the machine. When running `-dp` with `-dpconfig` on multiple machines across the network, you must specify `#num`.
- `-dpconfig dp_configuration_file` Specifies the configuration file for DP. Refer to the *CDPL Users Manual* for details about the configuration file.

Note: If `-dp` is triggered without specifying the `-dpconfig dp_configuration_file` option, all the distributions are run on the same machine.

- `-dplocation` Specifies if results files are to be written directly to NFS while intermediate files are written in local `/tmp` folder.
- `-merge` Merges the output files from HSPICE only if you specify this option.
- `-dpgui` Launches the DP manager to monitor the status of the DP run.

For more information on using DP Manager, See `$installdir/hspice/cdpl/doc/DPMangerUserGuide.pdf`

Licensing

HSPICE DP uses the same license keys as HSPICE. Each process running on the machine cluster checks out one copy of the license key.

The following sections discuss these topics:

- [HSPICE DP Feature Support](#)
- [Output Files](#)
- [HSPICE DP Limitations](#)

HSPICE DP Feature Support

HSPICE Distributed Processing supports the following features for .AC, .DC, and .TRAN analyses:

- .ALTER blocks
- Monte Carlo analysis
- Data block
- Parametric sweep
- The /tmp directory stores output files, then moves them back to the current run directory when the simulation finishes.
- Distributed processing works with the following command-line options:
 - -hpp
 - -mt *mt#*
 - -hdl
 - -hdlpath
 - -vamodel
 - -gz
- HSPICE automatically turns on .OPTION WARN_SEP when running DP to separate out warnings to a file, while suppressing them in the *.lis file.
- HSPICE automatically turns on .OPTION MCBRIEF=4 when running DP to generate minimal measure files.
- HSPICE provides load balance for Monte Carlo analysis, data block, and parametric sweeps:
 - DP may create many more tasks than workers.

- Workers located on fast machines process more tasks.
- Workers located on slow machines process fewer tasks.
- Ability to specify the location where the results files are to be written (NFS/TMP).

Output Files

The following table lists the names of the output files generated for `.ALTER`, Monte Carlo, data block, and parametric sweep simulations.

Table 5 Output files

Output File Name	Comments
<code>test_dp/test_dp_parseonly_report</code>	Contains the information that the DP pre-parser collects; HSPICE controls the DP behavior according to its content.
<code>test_dp/task#/test.lis</code>	Without <code>-merge</code> , the <code>*.lis</code> for each task remains in the worker subdirectory.
<code>test.lis</code>	Merged from all the <code>*.lis</code> files generated by all workers.
<code>test_dp.lis</code>	New file created by HSPICE DP. It contains parts of the errors and warnings generated by the simulation, Monte Carlo results, and DP runtime statistics.
<code>test_dp.progress</code>	New file created by HSPICE DP. It contains the DP progress status. This file is updated every 2 minutes.
<code>test.st0</code>	Merged from all the <code>*.st0</code> files generated by all workers.
<code>test.mt# (ALTER)</code>	The <code>*.mt#</code> files generated by all the ALTER blocks.
<code>test.mt0</code>	Merged from all the <code>*.mt0</code> files generated by all workers.
<code>test.mc# (ALTER)</code>	The <code>*.mc#</code> files generated by all the ALTER blocks.
<code>test.mc0</code>	Merged from all the <code>*.mc0</code> files generated by all workers.
<code>test.tr#</code>	The waveform files generated by all the ALTER blocks.

Chapter 4: Multiple Simulations, DP, and HPP

Running Distributed Processing (DP) on a Network Grid

Table 5 Output files

Output File Name	Comments
test_dp/worker#/test.tr0	The waveform file generated by each worker for Monte Carlo, data block or parametric sweep.
test.tr0@mc.grp	A waveform group file for Monte Carlo. When using Custom WaveView to open this file, the tool can load all the test_dp/worker#/test.tr0# files automatically. Use 2012.06 or later versions of Custom WaveView for maximum compatibility.
test.tr0@ds.grp	A waveform group file for data block or parametric sweep. When using Custom WaveView to open this file, the tool can automatically load all the test_dp/worker#/test.tr0 files. Use 2012.06 or later versions of Custom WaveView for maximum compatibility.
\$HOME/.synopsys/cdpl/master.hostname.pid.timestamp/bcast	
test_dp/master.hostname.pid.err	
test_dp/master.hostname.pid.log	
test_dp/worker.W#.hostname.pid.err	Only the master and worker produces the *.err file suffix.
test_dp/worker.W#.hostname.pid.log	

Table 6 Without “-merge” option

Alter Block	Monte Carlo, Data Block, Parametric Sweep
test_dp/test.dp_parseonly_report	test_dp/test.dp_parseonly_report
test_dp/worker#/test.lis	test_dp/worker#/test.lis
test_dp.lis	test_dp.lis
test_dp.progress	test_dp.progress
test.mc#	test.mc0
test.st#	test_dp/worker#/test.st0
test.mt#	test_dp/worker#/test.mt0
test.tr#	test_dp/worker#/test.tr0

Chapter 4: Multiple Simulations, DP, and HPP
 Running Distributed Processing (DP) on a Network Grid

Table 6 Without “-merge” option

Alter Block	Monte Carlo, Data Block, Parametric Sweep
	test.tr0@mc.grp (Monte Carlo only) test.tr0@ds.grp (Data block or parametric sweep only)
\$HOME/.synopsys/cdpl/master.hostname.pid.timestamp.bcast	
test_dp/master.hostname.pid.err	test_dp/master.hostname.pid.err
test_dp/master.hostname.pid.log	test_dp/master.hostname.pid.log
test_dp/worker.W#.hostname.pid.err	test_dp/worker.W#.hostname.pid.err
test_dp/worker.W#.hostname.pid.log	test_dp/worker.W#.hostname.pid.log

Table 7 With “-merge” option

Alter Block	Monte Carlo, Data Block, Parametric Sweep
test_dp/test.dp_parseonly_report	test_dp/test.dp_parseonly_report
test_dp.lis	test_dp.lis
test.lis	test.lis
test_dp.progress	test_dp.progress
test.st0	test.st0
test.mt#	test.mt0
test.mc#	test.mc0
test.tr#	test.tr0 test.tr0@mc.grp or test.tr0@ds.grp for HPP
\$HOME/.synopsys/cdpl/master.hostname.pid.timestamp/bcast	
test_dp/master.hostname.pid.err	test_dp/master.hostname.pid.err
test_dp/master.hostname.pid.log	test_dp/master.hostname.pid.log
test_dp/worker.W#.hostname.pid.err	test_dp/worker.W#.hostname.pid.err
test_dp/worker.W#.hostname.pid.log	test_dp/worker.W#.hostname.pid.log

HSPICE DP Limitations

The following limitations apply to the distributed processing (`-dp`) features:

- When a case has both Alter blocks and Monte Carlo (Data block or parametric sweep), DP is based on the Alter blocks only.
- HSPICE ignores DP when a case has the following:
 - Multiple `.tran`, `.dc`, or `.ac` statements and no `.alter` command
 - Temperature sweeps and no `.alter`
 - A `.load` command with `run=previous`
 - A TMI case (and sets the `.option tmiage` to 1)
 - A case with `.option radegfile` (MOSRA analysis)
 - RF features
- HSPICE ignores `.OPTION ALTCC` during a DP run.

Running Multithread/Multiprocess HSPICE Simulations

HSPICE can make use of multiple cores or processors within a single node via multithreading (`-mt`) for model evaluations and matrix solving, and multiprocessing (`-mp`) for multiple ALTER, SWEEP and Monte Carlo iterations. Since the 2009.09 release, you can run multi thread and multi process simulations simultaneously. You can run model evaluations concurrently on multiple CPUs by using multithreading to significantly improve simulation performance. In circuits composed primarily of active devices, model evaluation takes the most time. For designs with large numbers of linear resistors and capacitors (such as post-layout circuits) matrix solving can be multi threaded to speed up simulation with no loss of accuracy. You can run `-mt` with standard HSPICE, `-mt` with the backward differentiation formulae (BDF).

Note: Multithreading supports multiple CPUs and cores on a single node. Do not confuse multithreading with distributed processing, i.e., running on multiple nodes across an Ethernet LAN.

For multithreading to be effective in model evaluation, the number of active devices or elements should meet certain requirements.

The condition for model evaluation to be multithreaded is ONE of the following:

- MOSFET ≥ 64
- BJT ≥ 128
- Diode ≥ 128
- VCCS ≥ 128
- VCVS ≥ 128
- CCCS ≥ 128
- CCVS ≥ 128
- or parameter expressions ≥ 64 (parameter expression evaluation is parallelized)

If the circuit lacks the required number of active devices, HSPICE automatically uses a single thread. You can manually enforce multithreading on model evaluation by using [.OPTION MTTHRESH](#). The default MTTHRESH value is 64. You can set it to any positive integer number equal to or greater than 2.

For example, if MTTHRESH=50, HSPICE threads model evaluation of MOSFETs if the number of MOSFETs is greater than 50. Similarly, a diode model evaluation would receive benefit from multithreading if the circuit contains more than 100 (50 x 2) diodes. This option has no effect on matrix solving.

Note: In the case of too few active devices or elements, a multithreaded simulation may run slower than single-threaded. Whether or not performance degrades depends on many facets of the circuit topology. You can lower the default threshold of 64 in case the circuit under simulation has less than 64 active elements and still accelerates well with multithreading.

The following sections discuss these topics:

- [Running Multithreading and Multiprocessing Concurrently](#)

Running Multithreading and Multiprocessing Concurrently

You can run multi threading singly or concurrently with multiprocessing. On UNIX platforms, option `-mp [process_count] -mt [thread_count]` from the `hspice` command triggers a multi process simulation plus multithreading.

Note: You can run both MP and MT simultaneously only on LINUX.

The following is the command usage for simultaneous multithreading and multiprocessing:

```
% hspice -mp [process_count] -mt [thread_count] -i input.sp -o  
output_file
```

where,

- *process_count* is a nonzero integer, which defines the initial value of available child process numbers to fork. If you do not set *process_count*, then HSPICE uses the number of CPUs of the current machine as the default.
- *thread_count* is the maximum number of threads used in model evaluation and matrix solving. If you do not set *thread_count*, then HSPICE issues an error. If you specify a number of threads larger than the number of available CPUs, HSPICE sets the number equal to the number of available CPUs.

MP takes priority in CPU resource allocation. HSPICE triggers MT only if a remainder of CPU resources after MP's *process_count* is satisfied. A limitation exists on the instance number to trigger MT.

Available Cores

If the number of cores available for multithreading is not sufficient (after -mp) for the specified thread count, HSPICE lowers the value of the thread count to that of available cores. For example:

Example 1:

```
hspice -mp 6 -mt 4 -I input.sp -o input  
Core=8  
Monte=1000  
Total multiprocessed jobs=6  
Thread count is reset to 1
```

Example 2:

```
hspice -mp 2 -mt 4 -I input.sp -o input  
Core=4  
Monte=1000  
Total multiprocessed jobs=2  
Thread count is reset to 2
```

To determine the number of available processors and cores:

- Linux: Examine the processor, physical ID and core ID flags in the `/proc/cpuinfo` file.
- Solaris: Use the `/usr/platform/platform_name/sbin/prtdiag` command, where *platform_name* is determined by `uname -i`.
- Windows: Invoke the Task Manager, select the Performance tab, and count the number of CPU Usage History graphs.

One license is required per two CPUs.

For additional information about command-line options, see [hspice](#) in the *HSPICE Reference Manual: Commands and Control Options*.

In the Synopsys HSPICE User Interface (HSPUI):

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

Performance Improvement Estimations

$$T_{mt} = T_{serial} + T_{parallel}/N_{cpu} + T_{overhead}$$

Where:

`Tserial` represents HSPICE calculations that do not thread.

`Tparallel` represents threaded HSPICE calculations.

`Ncpu` is the number of CPUs used.

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

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

$$T_{1cpu} / (0.2T_{1cpu} + 0.8T_{1cpu}/2cpus) = 1.67$$

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

Multithread (-mt) on Windows

HSPICE multithreading (-mt) on Windows does not work together with Windows hyperthreading. You will have to turn-off the Windows Hyperthreading on the Windows server before running HSPICE multi-threading (-mt).

HSPICE Precision Parallel (-hpp)

HSPICE takes advantage of the latest multi core technology to speed up pre- and post-layout circuit simulation especially for medium to large size blocks including PLLs, ADCs, DACs, SERDES, and other full mixed-signal circuits of over 10 million elements. HSPICE Precision Parallel (HPP) scales performance from four to eight cores and beyond with no loss in accuracy. For a mini-video demonstration of HPP go to <http://www.hspice.com> and click **HSPICE MINI DEMOS** to load the **HSPICE Precision Parallel Technology** video.

The multi core algorithm can combine with multithreading to speed up transient analysis simulation by inserting -hpp and -mt *N* on the command line, where *N* is the number of threads. The multi core algorithm can apply without multithreading, but you obtain the best performance when you specify four or more threads.

When running the -hpp multi core algorithm with multi threading, every two threads requires one HSPICE license.

To enable HSPICE Precision Parallel, on the command-line, enter:

```
hspice -mt N -hpp -i input.sp -o output.lis
```

Table 8 Supported Platforms

Linux RHEL	Linux SUSE	Sun/Solaris	Windows
Yes	Yes	Yes	Yes

HPP Supported Features

The following list highlights major supported features. See also [HPP Limitations on page 64](#).

- .BIASCHK
- Distributed processing (-dp)

Chapter 4: Multiple Simulations, DP, and HPP

HSPICE Precision Parallel (-hpp)

- Multiprocessing (-mp)
- Monte Carlo analysis
- [Transient Noise Analysis](#) — (Monte Carlo method only)
- .IBIS, .PKG, .EBD (see [Using Supported IBIS with HPP](#))
- Verilog-A
- WDF and PSF formats
- .PRINT/.PROBE command keywords: level and filter
- Integration Method Support:
 - .OPTION METHOD=TRAP (default).
 - .OPTION METHOD=GEAR. Only second-order GEAR is supported.
 - .OPTION METHOD=BDF (defaults to METHOD=GEAR)
 - .OPTION MAXORD is limited up to the value 2.
- For data converter circuits: When running a transient simulation with HPP, you can tighten tolerances to resolve the least significant bit by using .OPTION RES_BITS. See [.OPTION RES_BITS](#) in the *HSPICE Reference Manual: Commands and Control Options*.
- You can enable analytical derivative computation of expression-based element evaluations in HPP for extensive accuracy by using .OPTION EQN_ANALYTICAL_DERIV. See [.OPTION EQN_ANALYTICAL_DERIV](#) in the *HSPICE Reference Manual: Commands and Control Options*.
- BSIM4: .OPTION IVTH, options ivth, ivthn and ivthp (see [HPP Limitations on page 64](#) for ivth measurements)

- Supported element output templates:

Element	HSPICE Output Templates
Resistor	lv1, lv2, lv3, lv4
Capacitor	lv1, lx0, lx1, lx2
Diode	lv1, lv23, lx0-lx5, lx7
MOS (level=54)	lv1- lv38, lx0-lx79

- Models supported by HPP include:

MOSFET Models	BJT Models	Diode, JFET Models	Other
BSIM3v3.3	HICUM L2.23	Diode Lvl 1	CMC R2 Lvl 2
BSIM4 4.6.5	Mextram 504.6.1	Diode Lvl 2	
BSIM3SOI 3.2.0	UCSD HBT	Diode Lvl 3	Model APIs
BSIMSOI 4.4	VBIC 95	JUNCAP NXP diode	CMI BSIM4-like topology
SSIMSOI 10500.2	VBIC 99	JUNCAP 200.3.3	TMI-1 (TSMC model interface)
PSP 103.1.2	BJT Lvl 1	JFET/MESFET Lvl 1	MOSRA
EKV (MOSFET)	BJT Lvl 2	JFET/MESFET Lvl 2	
MOS1	BJT Lvl 3	JFET/MESFET Lvl 3	
MOS2			
MOS3			
MOS9 (NXP) 903			
MOS11 (NXP) 1102.3			
BSIM1			
BSIM2			
BSIMCMG 105.03			
HiSIM251			
HiSIM_HV 2.0.0			
TFT Lvl 62			
Level66 (HVMOS)			

HPP Status Reporting in *.lis File

When you run HPP, the *.lis file shows running updates for the simulation status and the CPU utilization.

For example:

```
98.8% time = 34.597003 us ( etc = 56.4 sec, ett = 2:16:41 )  
( wall = 2:15:44 cpu = 1:02:14:45 s=11.6008 )
```

The terminology translates as follows:

% - *Percentage of simulation completion*
time = *Simulation time (seconds)*
etc = *Estimated time to completion sec*
ett = *Estimated total time*
wall= *Wall clock time*
cpu = *Total cpu time*
s = *Number of CPU's - utilization*

HPP Limitations

The following features and commands are *not* supported:

- For Monte Carlo:
 - .Variation Block
 - .Traditional AGAUSS style sampling method.

Note: HPP uses the option `sampling_method=SRS` as default.
- Transient Noise analysis-HPP, does not support the SDE method.
- CAP () measurement
- IBIS: .ICM
- .SAVE and .LOAD
- The following MOS element output templates are excluded from HPP support: lx13,lx14,lx15,lx17, lx25, lx30, lx31,lv5,lv6,lv7,lv14,lv16,lv17,lv20, lv23-26, and lv31-33.
- HPP does not support IVTH() or LX142() in .MEASURE statements.

Input Netlist and Data Entry

Describes the input netlist file and methods of entering data for HSPICE and HSPICE RF.

HSPICE ships hundreds of examples for your use; for HSPICE demo cases, see [Listing of Demonstration Input Files](#) for paths to demo files. For HSPICE RF demo cases, see [RF Demonstration Input Files](#) in the *HSPICE User Guide: RF Analysis*.

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

The following sections discuss these topics:

- [Input Netlist File Guidelines](#)
- [Input Netlist File Composition](#)
- [Using Subcircuits](#)
- [Subcircuit Call Statement Discrete Device Libraries](#)

Input Netlist File Guidelines

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

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

Chapter 5: Input Netlist and Data Entry

Input Netlist File Guidelines

- Specifies the type of output desired (optional).
- Each netlist line (logical record) cannot exceed 1024 characters. Use the “+” line continuation character to break up lines > 1024 characters in length to avoid generating an error.
- An input filename can be up to 1024 characters long for all platforms except PC Windows, which has a limitation of 256 characters.
- HSPICE has a limitation on the number of characters in a path name plus a file name of 1024 characters (except PC Windows, 256 characters or fewer). For example:

```
hspice -i path_name/input_file -o out_file
```

When specifying a path and file name using `-i` or `-o`, the length must be 1024 characters or fewer on all platforms. If the working directory path is greater than 1024 characters, HSPICE aborts with an error message.

- Model names in a netlist must begin with a letter. If you enter a model name with a leading integer HSPICE issues a parsing error.
- To generate input netlist and library input files, HSPICE uses either a schematic netlister or a text editor.
- Statements in the input netlist file can be in any order, except that the first line is a title line. In HSPICE, the last `.ALTER` submodule must appear at the end of the file and before the `.END` statement.

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

- For information on compressed netlist usage see [Compressed Netlist Guidelines on page 83](#).
- Netlist input processing is case insensitive, except for file names and their paths (see [Case Sensitivity on page 69](#)).

The following sections discuss these topics:

- [Input Line Format](#)
- [Case Sensitivity](#)
- [Special Characters](#)
- [First Character](#)
- [Delimiters](#)
- [Instance Names](#)

- [Hierarchy Paths](#)
- [Numbers](#)
- [Parameters and Expressions](#)
- [Reserved Keywords](#)
- [Input Netlist File Structure](#)
- [Schematic Netlists](#)
- [Compressed Netlist Guidelines](#)

Input Line Format

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

Any valid string of characters between two token delimiters is a token. You can use a character string as a parameter value in HSPICE, but not in HSPICE RF. See [Delimiters on page 74](#).
- A limitation of 1024 characters per line exists for an input statement or equation.
- Continuation lines:
 - Prepend a "+" character at the beginning of the continued line when the continuation is between tokens. For example:

```
R1 1 0
+ R='res1-res2'
```
 - Use a double backslash "\\\" at the end of the line for continuation when the continuation is inside a token or string. Whitespaces are optional to precede the string continuation. *A whitespace must not precede the token continuation, nor can a whitespace precede the double backslash.* For example:

Chapter 5: Input Netlist and Data Entry

Input Netlist File Guidelines

```
*** string continuation ***
R6 4 0 R='res1-\\
res2'
R5 4 0 R='res1- \\
res2'
*** token continuation ***
R4 node1 no\\
de2 R= 'res1-res2'
```

Here, the double slash equals `R4 node1 node2 R='res1-res2'`.

- If you do not invoke the `-case 1` command-line switch, HSPICE ignores differences between upper and lower case in input lines, except in quoted filenames and or after the `.INC` or `.LIB` commands.
- HSPICE automatically completes parentheses and quotation marks if they carry over to (+) continued lines.
- To indicate “to the power of” in your netlist, use two asterisks (**). For example, `2**5` represents two to the fifth power (2^5).
- HSPICE ignores all characters after the following listed statement lines:
 - `.include 'filename'`
 - `.lib 'filename' corner`
 - `.enddata`, `.end`, `.endl`, `.ends` and `.eom`

For example:

```
.include 'biasckt.inc';      $ semicolon ignored
.lib 'mos251.l' tt,         $ comma ignored
```

- Parameter names must begin with an alphabetic character, but thereafter can contain numbers and some special characters. See [Special Characters](#).
 - When you use an asterisk (*) or a question mark (?) with a `.PRINT`, `.PROBE`, `.LPRINT` (HSPICE RF), or `.CHECK` (HSPICE RF) statement, HSPICE uses the character as a wildcard. For additional information, see [Using Wildcards on Node Names on page 93](#).
 - When you use curly braces ({ }), HSPICE converts them to square brackets ([]) automatically.
 - Names become input tokens. Token delimiters must precede and follow names. See [Delimiters](#).
 - Names can be up to 1024 characters long and are not case-sensitive.

- Do not use any of the time keywords as a parameter name or node name in your netlist.
 - Reserved operator keywords include these symbols: () = ' `
- Do not use these symbols as part of any parameter or node name that you define. Use of these reserved operator keywords as names causes a syntax error, and HSPICE stops immediately.

Case Sensitivity

Where net, instance, parameter, and measure names, and any other name labels for downstream tools, require case sensitivity, HSPICE provides a mechanism to enable case-sensitive simulation. (Case insensitivity is unaffected for HSPICE keywords, commands, and options.) To invoke case sensitivity, on the HSPICE command-line enter:

```
-case 1
```

For example, HSPICE reads a netlist with case sensitivity when you enter:

```
%hspice -case 1 -i design.sp -o design.lis
```

Case-sensitivity applies for the following instances:

- Parameter Names
- Node Names
- Instance Names
- Model Names
- Subcircuit Names
- Data Names
- Measure Names
- File Names and Paths (enabled by default)
- Library Entry Names

Mixed-case data is writable to wavefiles. To assure backward compatibility in Custom WaveView, in the **Waveview** tab of the Preference Settings form, switch on the **Case Sensitive HSPICE Waveform Files** check box.

Special Characters

Table 9 on page 70 lists the special characters usable as part of node names, element parameter names, and element instance names. For details, see the appropriate sections in this chapter.

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

Table 9 HSPICE/ HSPICE RF Netlist Special Characters

Special Character		Node Name	Instance Name (cannot be the first character; element key letter only)	Parameter Name (cannot be the first character, element key letter only)	Delimiters
~	tilde	HSPICE: Legal anywhere HSPICE RF: Included only	Included only	Included only	n/a
!	exclamation point	Legal anywhere	Included only	Included only	n/a
@	at sign	Legal anywhere	included only	Included only	n/a
#	pound sign	Legal anywhere	Included only	Included only	n/a
\$	dollar sign	Avoid using the \$ character in a node name	Included only	Included only	In-line comment character
%	percent	HSPICE: Legal anywhere HSPICE RF: Included only	Included only	HSPICE: included only Illegal in HSPICE RF	n/a
^	caret	HSPICE: Legal anywhere HSPICE RF: included only	Included only	HSPICE: included only (avoid usage), Illegal in HSPICE RF	“To the power of”, i.e., 2 ⁵ , two raised to the fifth power

Table 9 HSPICE/HSPICE RF Netlist Special Characters

Special Character		Node Name	Instance Name (cannot be the first character; element key letter only)	Parameter Name (cannot be the first character, element key letter only)	Delimiters
&	ampersand	HSPICE: Legal anywhere HSPICE RF Included only	Included only	Included only	n/a
*	asterisk	HSPICE: included only (avoid using * in node names), Illegal for HSPICE RF	Included only	HSPICE: included only (avoid using in parameter names), Illegal in HSPICE RF	Comment in both HSPICE/HSPICE RF. Wildcard character. Double asterisk (**) is "To the power of".
()	parentheses	Illegal	Illegal	Illegal	Token delimiter
-	minus	HSPICE: included only HSPICE RF: Legal anywhere	Included only	Included only (avoid usage)	n/a
_	underscore	Legal anywhere	Included only	Included only	n/a
+	plus sign	HSPICE: included only HSPICE RF: Legal anywhere	Included only	HSPICE: included only (avoid usage); Illegal in HSPICE RF	Continues previous line including expressions and algebraics except for quoted strings
\	backslash	Illegal	Illegal	Illegal	Continuation character (preferred usage is a "+")

Chapter 5: Input Netlist and Data Entry

Input Netlist File Guidelines

Table 9 HSPICE/ HSPICE RF Netlist Special Characters

Special Character		Node Name	Instance Name (cannot be the first character; element key letter only)	Parameter Name (cannot be the first character, element key letter only)	Delimiters
“Legal anywhere”=first character or any position in name “Included only”=any position except first character					
\\	double backslash (requires a whitespace before to use as a continuation)	HSPICE: included only HSPICE RF: Legal anywhere	Illegal	Illegal	Continuation character for quoted strings (preserves whitespace)
=	equals	Illegal	Illegal	optional in .PARAM statements	Token delimiter
< >	less/more than; '<' and '>' are treated as a relation operators in an expression	HSPICE: Illegal to use <...> as node name or first character; Legal to follow a first character (i.e., n<0>)	Illegal	Legal as less than/ more than relational operators in an expression	n/a
?	question mark	HSPICE: Legal anywhere HSPICE RF: Illegal	Included only	Included only	Wildcard in character in both HSPICE and HSPICE RF
/	forward slash	Legal	Included only	Illegal	n/a
{ }	curly braces	HSPICE: included only, converts { } to [] No conversion for HSPICE RF	Included only	Included only	Auto-converts to square brackets ([]) Single ({ }) or ()) can be used in Variation Blocks
[]	square brackets	Included only	Included only	Included only	n/a

Table 9 HSPICE/ HSPICE RF Netlist Special Characters

Special Character	Node Name	Instance Name	Parameter Name	Delimiters
“Legal anywhere”=first character or any position in name “Included only”=any position except first character		(cannot be the first character; element key letter only)	(cannot be the first character, element key letter only)	
pipe	HSPICE: Legal anywhere HSPICE RF: Included only	Included only	Included only	n/a
, comma	Illegal	Illegal	Illegal	Token delimiter
. period	Illegal	Included only	Included only	Netlist keyword, (i.e., .TRAN, .DC, etc.). Hierarchy delimiter when used in node names
: colon	Included only	Included only	Illegal	Delimiter for element attributes
; semi-colon	Included only	Included only	Included only	n/a
" " double-quotes	Illegal	Illegal	Illegal	Expression and filename delimiter
' ' single quotes	Illegal	Illegal	Illegal	Expression and filename delimiter
Blank (whitespace)	Use before \ or \\ line continuations			Token delimiter
Tab Tab				Token delimiter

First Character

The first character in every line specifies how HSPICE and HSPICE RF interprets the remaining line. [Table 10](#) lists and describes the valid characters.

Table 10 First Character Descriptions

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

Delimiters

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

Note: The equal sign (=) is a token delimiter in the sense only that when you define a parameter, both the parameter name and parameter value are tokens, so the '=' is a token delimiter. However, you cannot enter '=' anywhere in a line like you would a comma, space, tab, or in the case of HSPICE, parentheses.

For example: the following is incorrect and returns an error message:

```
xtest2==== (a) , , (b) = , = , == (mysub====r1=1000=r2=1000 ( ( ( , ,
```

Instance Names

The names of element instances begin with the element key letter (see [Table 11](#)), except in subcircuits where instance names begin with X. (Subcircuits are interchangeably known as macros or modules.) Instance names can be up to 1024 characters long. In HSPICE, the `.OPTION LENNAM` defines the length of names in printouts (default=16).

Table 11 Element Identifiers

Letter (First Char)	Element	Example Line
B	IBIS buffer	b_io_0 nd_pu0 nd_pd0 nd_out nd_in0 nd_en0 nd_outofin0 nd_pc0 nd_gc0
C	Capacitor	Cbypass 1 0 10pf
D	Diode	D7 3 9 D1
E	Voltage-controlled voltage source	Ea 1 2 3 4 K
F	Current-controlled current source	Fsub n1 n2 vin 2.0
G	Voltage-controlled current source	G12 4 0 3 0 10
H	Current-controlled voltage source	H3 4 5 Vout 2.0
I	Current source	I A 2 6 1e-6
J	JFET or MESFET	J1 7 2 3 GAASFET
K	Linear mutual inductor (general form)	K1 L1 L2 1
L	Linear inductor	LX a b 1e-9
M	MOS transistor	M834 1 2 3 4 N1

Table 11 Element Identifiers (Continued)

Letter (First Char)	Element	Example Line
P	Port	P1 in gnd port=1 z0=50
Q	Bipolar transistor	Q5 3 6 7 8 pnp1
R	Resistor	R10 21 10 1000
S	S parameter element	S1 nd1 nd2 s_model2
V	Voltage source	V1 8 0 5
T,U,W	Transmission Line	W1 in1 0 out1 0 N=1 L=1
X	Subcircuit call	X1 2 4 17 31 MULTI WN=100 LN=5

Hierarchy Paths

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

Numbers

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

Table 12.

Table 12 Scale Factors

Scale Factor	Prefix	Symbol	Multiplying Factor
T	tera	T	1e+12
G	giga	G	1e+9
MEG or X	mega	M	1e+6
K	kilo	k	1e+3
MIL	n/a	none	25.4e-6
M	milli	m	1e-3
U	micro	μ	1e-6
N	nano	n	1e-9
P	pico	p	1e-12
F	femto	f	1e-15
A	atto	a	1e-18
DB	DB	db	10 ^(value/20)

Note: Scale factor A is not a scale factor in a character string that contains amps. For example, HSPICE interprets a “20amps” string as 20 amperes of current, not as 20e-18mps.

- Numbers can use exponential format or engineering key letter format (1e-12 or 1p), but to use both (1e-6u), you must specify `.OPTION MIXED_NUM_FORMAT = 1`.
- To designate exponents, use D or E.
- The `.OPTION EXPMAX` limits the exponent size.
- HSPICE interprets trailing alphabetic characters as units comments.
- Units comments do not go through a checking process.
- The `.OPTION INGOLD` controls the format of numbers in printouts.

Chapter 5: Input Netlist and Data Entry

Input Netlist File Guidelines

- The `.OPTION NUMDGT=x` controls the listing printout accuracy.
- The `.OPTION MEASDGT=x` controls the measure file printout accuracy.
- In HSPICE, `.OPTION VFLOOR=x` specifies the smallest voltage for which HSPICE prints the value. Smaller voltages print as 0.
- The DB scale factor `s` is interpreted in the same manner by HSPICE, HPP, and RF.

Parameters and Expressions

- Parameter names in HSPICE use HSPICE name syntax rules, except that names must begin with an alphabetic character. The other characters must be either a number, or a special character. See [Table 9 on page 70](#) in the [Special Characters](#) section for a listing of legal parameter names. For example, a “%” is legal if included in HSPICE, but illegal in HSPICE RF.
- To define parameter hierarchy overrides and defaults, use the `.OPTION PARHIER=global | local` statement.
- If you create multiple definitions for the same parameter or option, HSPICE uses the last parameter definition or `.OPTION` statement, even if that definition occurs later in the input than a reference to the parameter or option. HSPICE does not warn you when you redefine a parameter.
- You must define a parameter before you use that parameter to define another parameter.
- When you select design parameter names, be careful to avoid conflicts with parameterized libraries.
- To delimit expressions, *always* use single or double quotes. Otherwise, if a mathematical character (+ - * /) is within a string, HSPICE regards the string also as an expression.
- Expressions cannot exceed 4096 characters.
- For readability, use a double backslash preceded by a whitespace (\\) at end of a line, to continue the line.
- Use the `PAR` (expression or parameter) function to evaluate expressions in output statements.

- Limitation 1: If you define a parameter as an expression containing output signals such as `v(node)` or `i(element)`, this parameter applies only in an element value expression directly. HSPICE does not evaluate it to another parameter.

For example, the following is correct:

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

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

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

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

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

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

Correct

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

Incorrect

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

Reserved Keywords

Reserved keywords are those which are illegal for a parameter name and/or node name. The following keywords are not “global” and are analysis/syntax-specific; that is, you can use these keywords in some specific statements, but not in other statements. For example, you cannot use `pwl` as a parameter name in the V source, but you can use it in an R-element. You can use these keywords as parameter names with single quotes anywhere. [Table 13 on](#)

Chapter 5: Input Netlist and Data Entry

Input Netlist File Guidelines

[page 80](#) lists the illegal keywords for the specified *element* type. [Table 14 on page 81](#) lists the illegal keywords for the specified *command*.

For more information on the need for stringent adherence to parser requirements see [Appendix C, HSPICE Parser Strict Syntax Requirements](#).

Table 13 Reserved Keywords and Elements

Elements	Illegal Keywords
B-element	FILE, MODEL, NOWARN, COMPL_TR
Bjt/Jfet element	OFF, IC, TNODEOUT
Capacitor	POLY, TC, SENS
Diode	OFF, IC
E/G-element	POLY, PWL, AND, NAND, PPWL, NPWL, OR, NOR, LAPLACE, POLE, VCCS, OPAMP, DELAY, TRANSFORMER, VCR, VCCAP, VCVS, SPUR, FREQ, ZTRANS, VMRF, FOSTER, NOISE, NOISEFILE, MNAME, PHASE, SCALE, MAX, PAR
F/H-elements	POLY, PWL, AND, NAND, OR, NOR, VMRF, CCCS, C CVS, DELAY
Independent Sources and Port element	SIN, PU, PWL, EXP, PULSE, PE, SFFM, AM, PAT, PL, HB, HBAC, DATA, AC, DC, LSAC, SNAC, PHOTO, NEUT, COS, VMRF, LFSR, PUL, HBLIN, R, BITSTREAM, PWLFILE, MOD, FILTER
Inductor	POLY, TC, SENS, RELUCTANCE, TRANSFORMER_NT, FILE
Resistor	POLY, TC, SENS
S-element	ZO, Z0, MNAME
T/U- element	IC
W-element	RLGCFILE, PRINTZO, RLGCMODEL, TABLEMODEL, FSMODEL, UMODEL, SMODEL

Table 14 Reserved Keywords and Commands

Commands	Illegal Keywords
.AC/.DC	UIC, MONTE, DATA, SWEEP, POI, DEC, LIN, OCT, RESULTS, LIST, OPTIMIZE, SWEEPBLOCK, EXPLORE
.DATA	MER, LAM, FILE, MERGE
.FQMODEL	DATA
.FSOPTION	COMPUTETABLE
.LAYERSTACK	LAYER
.LIN	FORMAT, LISTFREQ, FILENAME
.PAT	DATA
.PKG/.EBD/.IBIS/.ICM	FILE, MODEL, COMPONENT, NOWARN, MOD_SEL
.SAVE/.LOAD	FILE
.SHAPE	DATA, VERTEX, N
.STIM	PWL, DATA, VEC, TRAN, AC, DC, FILENAME
.TRAN	UIC, MONTE, DATA, SWEEP, POI, DEC, LIN, OCT, RESULTS, LIST, OPTIMIZE, SWEEPBLOCK, RESULT

Input Netlist File Structure

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

You can use several high-level call statements (such as `.INCLUDE` and `.LIB`) to structure the input netlist file modules. These statements can call netlists, model parameters, test vectors, analysis, and option macros into a file from library files or other files. The input netlist file also can call an external data file, which contains parameterized data for element sources and models. You must

enclose the names of included or internally specified files in single or double quotation marks when they begin with a number (0-9).

Schematic Netlists

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

The process of creating a schematic involves:

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

Table 15 Input Netlist File Sections

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

Table 15 Input Netlist File Sections (Continued)

Sections	Examples	Definition
Output	.PRINT, .PROBE, .MEASURE	Statements to output variables. Statement to evaluate and report user-defined functions of a circuit.
Library, Model and File Inclusion	.INCLUDE	General include files.
	.MALIAS	Assigns an alias to a diode, BJT, JFET, or MOSFET.
	.MODEL	Element model descriptions.
	.LIB	Library.
	.OPTION SEARCH	Search path for libraries and included files. (HSPICE)
	.OPTION BRIEF= 1 and .OPTION BRIEF= 0	Control printback to output listing. (HSPICE)
Alter blocks (HSPICE Only)	.ALIAS, .ALTER, .DEL LIB	Renames a previous model. Sequence for in-line case analysis. Removes previous library selection.
End of netlist	.END	Required statement; end of netlist.

Compressed Netlist Guidelines

To save disk space and to improve performance on large netlists, HSPICE supports the `.gzip` file format for both input files (`design.sp`) and these statements:

- `.inc`
- `.lib`
- `.load`

However, HSPICE does not support the S-parameter and RLGC file input types in the .gz format.

When both *design.sp* and *design.sp.gz* exist, HSPICE always selects the exact match file first, if it exists. For example:

1. If `.include design.sp` is in the netlist, then HSPICE picks *design.sp*.
2. If `.include design.sp.gz` is in the netlist, then HSPICE picks *design.sp.gz*
3. If `.include design.sp` is in the netlist and only *design.sp.gz* exists, then HSPICE chooses *design.sp.gz*.
4. If `.include design.sp.gz` is in the netlist and only *design.sp* exists, then HSPICE uses *design.sp*.

Input Netlist File Composition

The HSPICE and HSPICE RF circuit description syntax is compatible with the SPICE input netlist format. [Figure 6](#) and [Figure 6 on page 85](#) show the basic structure of an input netlist and note comment and continuation characters.

Netlist Structure: Overview

Title	First line is always the title
Comment character	* - comment for a line \$ - comment after a command
Options	.option post
Print/Probe/Analysis	.print v(d) i(rl) .probe v(g) .tran .1n 5n
Initial Conditions	.ic v(b) = 0 \$ input state
Sources	Vg g 0 pulse 0 1 0 0.15 0.15 0.42 * example of a voltage source
Circuit Description	MN d g gnd n nmos RL vdd d 1K
Models	.model n nmos level = 49 + vto = 1 tox = 7n * '+' continuation character
End	.end \$ terminates the simulation

Figure 5 Basic netlist structure

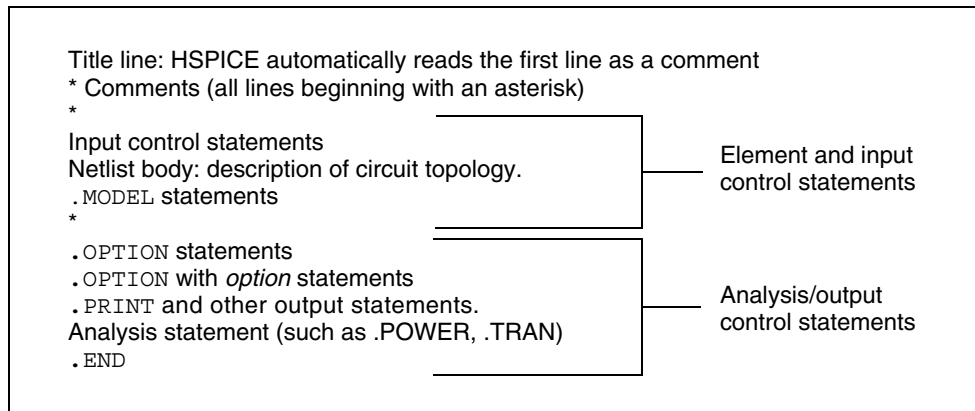


Figure 6 Sections of a netlist

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

To create the circuit:

1. Define MOSFET models for the PMOS/NMOS transistors of the inverter.
2. Insert the power supplies for both VDD and GND power rails.
3. Insert the pulse source to the inverter input.

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

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

Chapter 5: Input Netlist and Data Entry

Input Netlist File Composition

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

The following sections discuss these topics:

- [HSPICE Topology Rules](#)
- [Title of Simulation](#)
- [Comments and Line Continuation](#)
- [Element and Source Statements](#)
- [Defining Subcircuits](#)
- [Node Name \(or Node Identifier\) Conventions](#)
- [Element, Instance, and Subcircuit Naming Conventions](#)
- [Subcircuit Node Names](#)
- [Path Names of Subcircuit Nodes](#)
- [Abbreviated Subcircuit Node Names](#)
- [Automatic Node Name Generation](#)
- [Global Node Names](#)
- [Circuit Temperature](#)
- [Data-Driven Analysis](#)
- [Library Calls and Definitions](#)
- [Defining Parameters](#)
- [Altering Design Variables and Subcircuits](#)
- [Connecting Nodes](#)
- [Deleting a Library](#)
- [Ending a Netlist](#)
- [Condition-Controlled Netlists \(IF-ELSE\)](#)

HSPICE Topology Rules

When constructing the circuit description HSPICE does not allow certain topologies. See [Figure 7 on page 87](#). The following rules apply:

1. No voltage loops (no voltage sources in parallel with no other elements).

2. No ideal voltage source in closed inductor loop.
3. No stacked current sources (no current sources in series).
4. No ideal current source in closed capacitor loop.

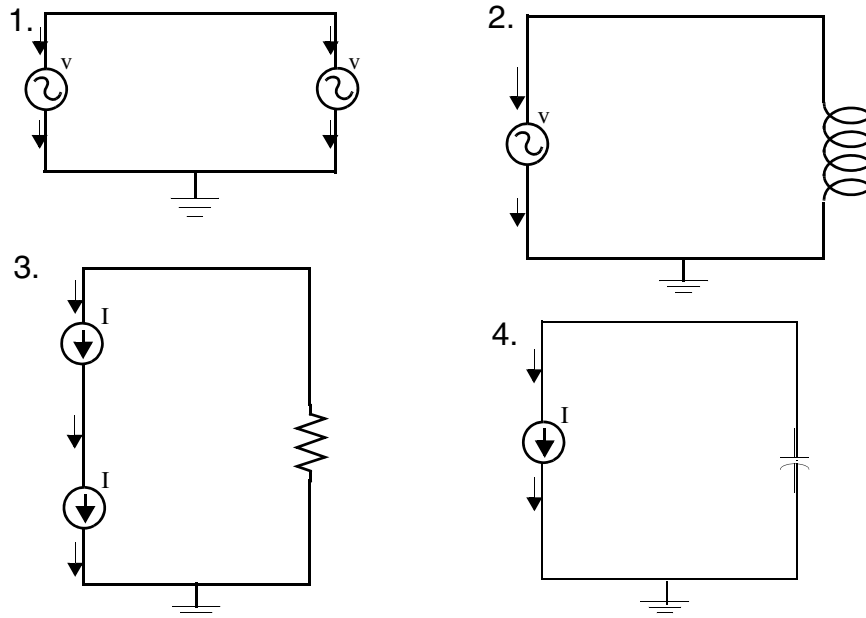


Figure 7 Illegal topologies in HSPICE

Title of Simulation

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

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

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

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

Comments and Line Continuation

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

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

You can place comment statements anywhere in the circuit description.

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

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

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

For example,

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

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

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

```
.ABC Title Line (HSPICE or HSPICE RF ignores the netlist keyword
* on this line because the first line is always a comment)
* This is a comment line
.MODEL n1 NMOS $ this is an example of an inline comment
* This is a comment line and the following line is a continuation
+ LEVEL=3
```

Element and Source Statements

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

Element statements specify:

- Type of device.
- The nodes to the connected device.
- Operating electrical characteristics of the device.

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

[Table 16](#) lists the parameters of an element statements.

Table 16 Element Parameters

Parameter	Description
elname	Element name that cannot exceed 1023 characters, and must begin with a specific letter for each element type: B IBIS buffer (HSPICE Only) C Capacitor D Diode E,F,G,H Dependent current and voltage sources I Current (inductance) source J JFET or MESFET K Mutual inductor L Inductor model or magnetic core mutual inductor model M MOSFET Q BJT P Port R Resistor S, S-parameter mode T, U, W Transmission line V Voltage source X Subcircuit call
node1 ...	Node names identify the nodes that connect to the element. The node name begins with a letter and can contain a maximum of 1023 characters. For a listing of legal and illegal special characters usable in node names, see the Special Characters section, Table 9 on page 70 .

Table 16 Element Parameters (Continued)

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

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

Example 1

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

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

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

The preceding example specifies a MOSFET named M1, where:

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

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

Example 2

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

The preceding example specifies a MOSFET named M1, where:

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

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

Defining Subcircuits

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

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

Node Name (or Node Identifier) Conventions

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

Chapter 5: Input Netlist and Data Entry

Input Netlist File Composition

The following are conventions regarding node names:

- In addition to letters and digits, node names can include, but NOT always begin with some special characters. See the [Special Characters](#) section, [Table 9 on page 70](#). Note that use of special characters in node names varies between HSPICE and HSPICE RF.
- If you use braces { } in node names, HSPICE changes them to brackets [].
- You cannot use the following characters in node names:
() , = , ' , <blank>
- You should avoid using the dollar sign (\$) after a numerical digit in a node name because HSPICE assumes whatever follows the "\$" symbol is an in-line comment (see [Comments and Line Continuation on page 88](#) for additional information). It can cause error and warning messages depending on the location of the node with the "\$". For example, HSPICE generates an error to indicate that a resistor node is missing:

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

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

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

- The period (.) is a reserved character for use as a separator between a subcircuit name and a node name: *subcircuitName.nodeName*. If a node name contains a period, the node HSPICE considers it a top level node unless there is a valid match to a subcircuit name and node name in the hierarchy.
- The sorting order for operating point nodes is:
a-z, !, #, \$, %, *, +, -, /
- To make node names global across all subcircuits, use a .GLOBAL statement (see [Global Node Names on page 98](#)).
- The 0, GND, GND!, and GROUND node names all refer to the global HSPICE ground. Simulation treats nodes with any of these names as a ground node, and produces v(0) into the output files. Besides these ground nodes, HSPICE regards all node names as separate nodes. For example, 0 and 0.3 are different nodes, 1A and 1 are different nodes, 2~ and 2 are different nodes.

The following sections discuss these topics:

- [Using Wildcards on Node Names](#)
- [Wildcard Applications and Examples](#)

Using Wildcards on Node Names

You can use wildcards to match node names.

- The `?` wildcard matches any single character. For example, `9?` matches `92`, `9a`, `9A`, and `9%`.
- The `*` wildcard matches any string of zero or more characters. For example:
 - If your netlist includes a resistor named `r1` and a voltage source named `vin`, then `.PRINT i(*)` prints the current for both of these elements: `i(r1)` and `i(vin)`.
 - And `.PRINT v(o*)` prints the voltages for all nodes whose names start with `o`; if your netlist contains nodes named `in` and `out`, this example prints only the `v(out)` voltage. For example, the following prints the results of a transient analysis for the voltage at the matched node name.

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

- Wildcards must begin with a letter or a number; wildcards must begin with output type `V`, `I` or `P`; for example,

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

Wildcard Applications and Examples

Here are some practical applications for these wildcards:

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

Chapter 5: Input Netlist and Data Entry

Input Netlist File Composition

- If your netlist contains nodes named 0, 1, 2, and 3, then `.PRINT v(0,*)` or `.PRINT v(0 *)` prints the voltage between node 0 and each of the other nodes: `v(0,1)`, `v(0,2)`, and `v(0,3)`.
- Wildcards can set initial conditions in `.IC` and `.NODESET` statements. Node names including wildcards in `.IC` and `.NODESET` must start with “v()”. For example, `.NODESET v(a*)=5`.

Examples

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

- Probe node voltages for nodes at all levels.
`.PROBE v(*)`
- Probe all nodes whose names start with “a”. For example: a1, a2, a3, a00, ayz.
`.PROBE v(a*)`
- Print node voltages for nodes at the first level and all levels below the first level, where zero-level are top-level nodes. For example: x1.A, x4.554, xab.abc123.
`.PRINT v(*.*)`
- Probe node voltages for all nodes whose name start with “x” at the first level and all levels below the first level, where zero-level are top-level nodes. For example: x1.A, x4.554, xab.abc123.
`.PROBE v(x*.*)`
- Print node voltages for nodes whose names start with “x” at the second-level and all levels below the second level. For example: x1.x2.a, xab.xdff.in.
`.PRINT v(x*.*?)`
- Match all first-level nodes with names that are exactly two characters long. For example: x1.in, x4.12.
`.PRINT v(x*.*.*)`
- In HSPICE RF, print the logic state of all top-level nodes, whose names start with b. For example: b1, b2, b3, b56, bac.

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

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

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

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

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

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

Element, Instance, and Subcircuit Naming Conventions

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

Element names in HSPICE begin with a letter designating the element type, followed by up to 1023 alphanumeric and special characters as defined in [Table 9 on page 70](#). 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 89](#)).

Subcircuit Node Names

HSPICE assigns two subcircuit node names.

- To assign the first name, HSPICE uses the (.) extension to concatenate the circuit path name with the node name — for example, X1.XBIAS.M5. Node designations that start with the same number, followed by any letter, are the same node. For example, 1c and 1d are the same node.
- The second subcircuit node name is a unique number that HSPICE automatically assigns to an input netlist subcircuit. The (:) extension concatenates this number with the internal node name, to form the entire subcircuit's node name (for example, 10:M5). The output listing file cross-references the node name.

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

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

Path Names of Subcircuit Nodes

A path name consists of a sequence of subcircuit names, starting at the highest-level subcircuit call, and ending at an element or bottom-level node. Periods separate the subcircuit names in the path name. The maximum length of the path name, including the node name, is 1024 characters. You can use path names in .PRINT, .NODESET, and .IC statements, as another way to reference internal nodes (nodes not appearing on the parameter list). You can use the path name to reference any node, including any internal node. Subcircuit node and element names follow the rules shown in [Figure 8 on page 96](#).

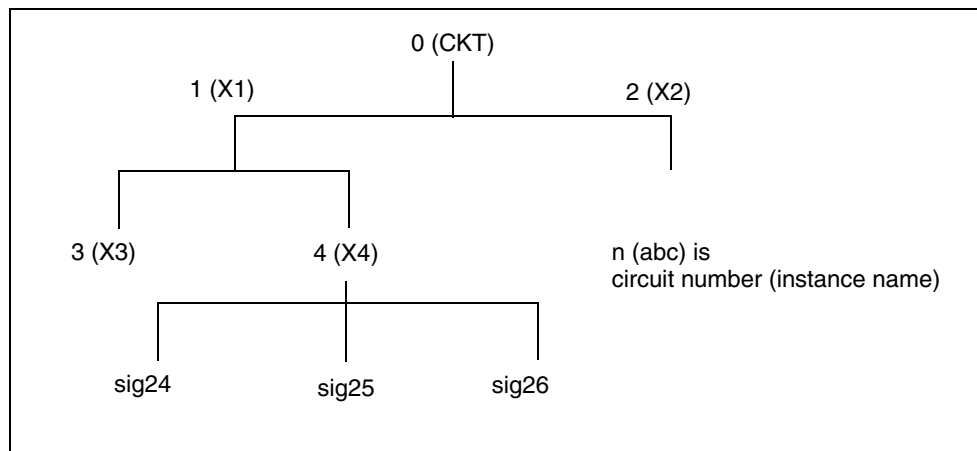


Figure 8 Subcircuit Calling Tree, with Circuit Numbers and Instance Names

In [Figure 8](#), the path name of the sig25 node in the X4 subcircuit is X1.X4.sig25. You can use this path in HSPICE statements, such as:

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

Abbreviated Subcircuit Node Names

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

subckt - num: name

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

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

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

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

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

Automatic Node Name Generation

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

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

Global Node Names

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

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

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

Circuit Temperature

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

HSPICE RF supports only the last `.TEMP` command in a netlist, if you use multiple `.TEMP` commands.

Data-Driven Analysis

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

HSPICE supports the entire functionality of the `.DATA` statement. (See [.DATA](#) in the *HSPICE Reference Manual: Commands and Control Options*.) However, HSPICE RF supports `.DATA` only for:

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

For more details about using the `.DATA` statement in different types of analysis, see [Chapter 13, Initializing DC-Operating Point Analysis](#) and [Chapter 15, Transient Analysis](#).

Library Calls and Definitions

To create and read from libraries of commonly used commands, device models, subcircuit analysis, and statements in library files, use the `.LIB` call statement. As HSPICE encounters each `.LIB` call name in the main data file, it reads the corresponding entry from the designated library file, until it finds an `.ENDL` statement. In HSPICE, you can also place a `.LIB` call statement in an `.ALTER` block.

The following sections discuss these topics:

- [Library Building Rules](#)
- [Automatic Library Selection \(HSPICE\)](#)

Library Building Rules

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

Automatic Library Selection (HSPICE)

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

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

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

1. Read the input file for a `.SUBCKT` or `.MACRO` with the specified call name.
2. Read any `.INC` files or `.LIB` files for a `.SUBCKT` or `.MACRO` with the specified call name.
3. Search the directory with the input file for the `call_name.inc` file.
4. Search the directories in the `.OPTION SEARCH` list.

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

Defining Parameters

The `.PARAM` statement defines parameters. Parameters in HSPICE are names that have associated numeric values. You can define parameters through predefined analysis or measurement

The following sections discuss these topics:

- [Predefined Analysis](#)
- [Measurement Parameters](#)

Predefined Analysis

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

HSPICE supports the following predefined analysis parameters:

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

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

- frequency
- time

Measurement Parameters

A `.MEASURE` statement produces a measurement parameter. In general, the rules for measurement parameters are the same as those for standard parameters. While a `.PARAM` statement disallows definition of measurement parameters, the `.MEASURE` statement directly defines such parameters. For more information, see [.MEASURE Parameter Types on page 426](#).

Altering Design Variables and Subcircuits

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

- If the name of a new element, `.MODEL` statement, or subcircuit definition is identical to the name of an original statement of the same type, then the new statement replaces the old. Add new statements in the input netlist file.
- You can alter element and `.MODEL` statements within a subcircuit definition. You can also add a new element or `.MODEL` statement to a subcircuit definition. To modify the topology in subcircuit definitions, put the element into libraries. To add a library, use `.LIB`; to delete, use `.DEL LIB`.
- If you want to change temperature points in multiple `.ALTER` statements then, parameterizing the `.TEMP` statement and updating the `.PARAM` statement in each `.ALTER` statement is recommended, instead of specifying the `.TEMP` value in the `.ALTER` blocks.
- If a parameter name in a new `.PARAM` statement in the `.ALTER` module is identical to a previous parameter name, then the new assigned value replaces the old value.

Chapter 5: Input Netlist and Data Entry

Input Netlist File Composition

- If you used parameter (variable) values for elements (or model parameter values) when you used `.ALTER`, use the `.PARAM` statement to change these parameter values. Do not use numerical values to redescribe elements or model parameters.
- If you used an `.OPTION` statement (in an original input file or a `.ALTER` block) to turn on an option, you can turn that option off.
- Each `.ALTER` simulation run prints only the actual altered input. A special `.ALTER` title identifies the run.
- `.ALTER` processing cannot revise `.LIB` statements within a file that an `.INCLUDE` statement calls. However, `.ALTER` processing can accept `.INCLUDE` statements, within a file that a `.LIB` statement calls.

This section covers the following topics:

- [Using Multiple `.ALTER` Blocks](#)
- [Adding or Changing Analyses in `.ALTER` Blocks](#)

Using Multiple `.ALTER` Blocks

This section does not apply to HSPICE RF.

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

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

1. Put the statements that precede the first `.ALTER` statement, into a library.
2. Use the `.LIB` statement in the main input file.

3. Put a `.DEL LIB` statement in the `.ALTER` section, to delete that library for the `.ALTER` simulation run.

Adding or Changing Analyses in `.ALTER` Blocks

When you add a new analysis statement (`.TRAN`, `.AC`, `.DC`, `.TEMP`, etc.) in an `.ALTER` block, the added analysis statement does not replace the analysis statement previously defined in the top level. Instead, the analysis command in each `.ALTER` runs in addition to the analysis statement in the top level. This will cause HSPICE to output more analysis results files than expected.

Example 1:

In the following example, a new temperature statement is added in each `.ALTER` block.

```
* top level netlist
V1 1 0 pulse 0 1 0 1n 1n 2n 4n
R1 1 2 1k tc1=0.01
R2 2 0 1k tc1=0.005
.temp 60
.tran 1n 20n
.alter 1
.temp -40
.alter 2
.temp 110
.end
```

The `.TEMP` commands from the top level and each alter are cumulative and HSPICE will generate 6 `*.tr#` files as follows:

```
Top level: *.tr0 @ temp=60
Alter 1:   *.tr1 @ temp=60
           *.tr2 @ temp=-40
Alter 2:   *.tr3 @ temp=60
           *.tr4 @ temp=-40
           *.tr5 @ temp=110
```

Chapter 5: Input Netlist and Data Entry

Input Netlist File Composition

To output only one *.tr# file for each temperature value, it is recommended that the .TEMP value be parameterized and only one .TEMP statement at the top level be used.

```
* top level netlist
.param tc=60    $ temperature control
V1 1 0 pulse 0 1 0 1n 1n 2n 4n
R1 1 2 1k tc1=0.01
R2 2 0 1k tc1=0.005
.temp 'tc'
.tran 1n 20n
.alter 1
.param tc=-40
.alter 2
.param tc=110
.end
```

Example 2:

In the following example, a new transient analysis with a different simulation time is added in each .ALTER block.

```
* top level netlist
.param pw=10n per=20n
V1 1 0 pulse 0 1 0 1n 1n 'pw' 'per'
R1 1 2 1k
R2 2 0 1k
.tran 1n 20n
.alter 1
.param pw=7.5n per=15n
.tran 1n 15n
.alter 2
.param pw=5n per=10n
.tran 1n 10n
.end
```

This netlists will generate 6 *.tr# files instead of the expected 3 *.tr# files. To generate the desired 3 *.tr# files, you can parameterize the tstop value

in the `.TRAN` analysis statement. Then use only one `.TRAN` statement and change the parameterized `tstop` value in each `.ALTER` block:

```
* top level netlist
.param pw=10n per=20n
.param ts=20n $ tstop value
V1 1 0 pulse 0 1 0 1n 1n 'pw' 'per'
R1 1 2 1k
R2 2 0 1k
.tran 1n 'ts'
.alter 1
.param pw=7.5n per=15n
.param ts=15n
.alter 2
.param ts=10n
.param pw=5n per=10n
.end
```

Connecting Nodes

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

Deleting a Library

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

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

Ending a Netlist

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

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

Condition-Controlled Netlists (IF-ELSE)

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

```
.if (condition1)
    statement_block1

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

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

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


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

Guidelines for Using IF-ELSE Blocks (HSPICE Only)

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

- In an .IF, .ELSEIF, or .ELSE condition statement, complex Boolean expressions must not be ambiguous. For example, change (a==b && c>=d) to ((a==b) && (c>=d)).
- In an IF, ELSEIF, or ELSE statement block, you can include most valid HSPICE (*not* HSPICE RF) analysis and output statements. The exceptions are:
 - .END, .ALTER, .GLOBAL, .DEL LIB, .MALIAS, .ALIAS, .LIST, .NOLIST, and .CONNECT statements.
 - .OPTIONS search, d_ibis, d_imic, d_lv56, biasfi, modsrh, cmiflag, nxx, and brief.
- You can include IF-ELSEIF-ELSE statements in subcircuits and subcircuits in IF-ELSEIF-ELSE statements.
- You can use IF-ELSEIF-ELSE blocks to select different submodules to structure the netlist (using .INC, .LIB, and .VEC statements).

Chapter 5: Input Netlist and Data Entry

Using Subcircuits

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

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

Using Subcircuits

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

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

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

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

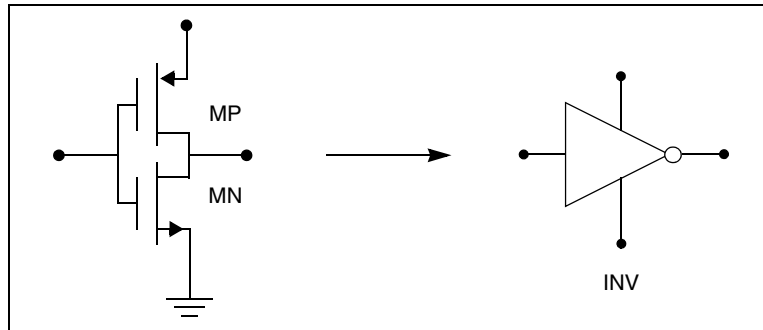


Figure 9 Subcircuit Representation

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

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

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

The following sections cover these topics.

- [Hierarchical Parameters](#)
- [Undefined Subcircuit Search \(HSPICE\)](#)
- [Troubleshooting Subcircuit Node Issues](#)

Hierarchical Parameters

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

The following section discuss these topics:

- [M \(Multiply\) Parameter](#)
- [S \(Scale\) Parameter](#)
- [Using Hierarchical Parameters to Simplify Simulation](#)

M (Multiply) Parameter

The most basic HSPICE subcircuit parameter is the M (multiply) parameter. This keyword is common to all elements, including subcircuits, except for voltage sources. The M parameter multiplies the internal component values. This, in effect, creates parallel copies of the element.

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

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

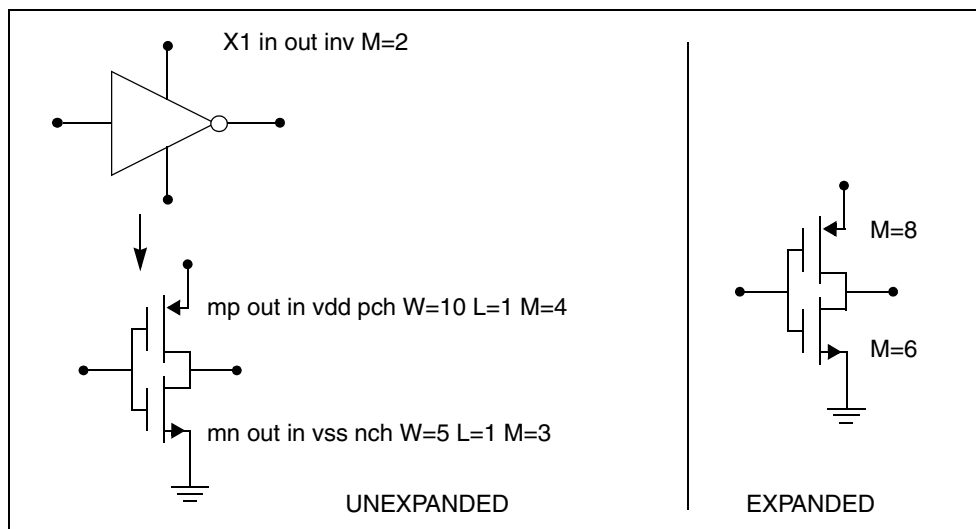


Figure 10 How Hierarchical Multiply Works

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

S (Scale) Parameter

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

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

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

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

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

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

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

In this example:

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

Using Hierarchical Parameters to Simplify Simulation

You can use the hierarchical parameter to simplify simulations. For example, see the following listing and [Figure 11 on page 112](#).

```
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
```

Chapter 5: Input Netlist and Data Entry

Using Subcircuits

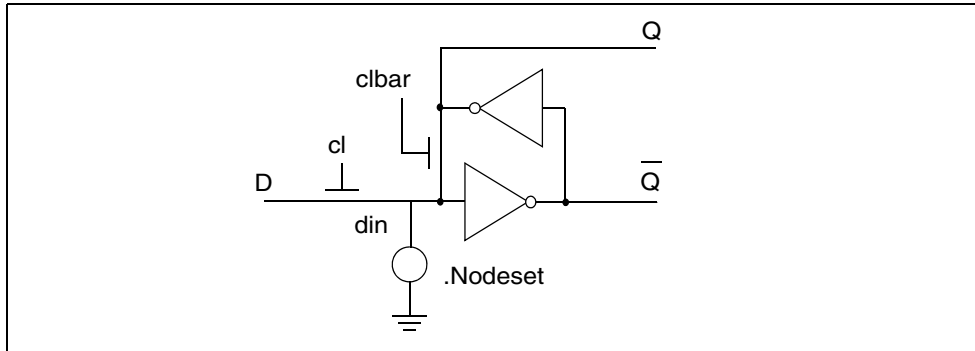


Figure 11 D Latch with Nodeset

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

To specify subcircuit nodes in `.PRINT` statements, specify the full subcircuit path and node name. Use a `.print/.probe/.measure` statement to print the hierarchical parameter. For example:

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

You add function `valp()` to refer to a parameter defined in subckt for output (including a forward slash). For example:

```
.meas tran asdf param='valp(x1/p1)'
```

Undefined Subcircuit Search (HSPICE)

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

1. Current directory for the file.
2. Directories specified in `.OPTION SEARCH = "directory_path_name"` statements.
3. Directory containing the Discrete Device Library.

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

Troubleshooting Subcircuit Node Issues

Use the following workarounds or solutions to subcircuit-related errors:

- [Subcircuit Name Missing Error](#)
- [Duplicated Subckt Definition Error](#)
- [Duplicate Node Message](#)
- [Duplicating Ports](#)
- [Using Assigned Circuit Numbers instead of Full Path Node Names](#)
- [Simple Reporting of Floating Nodes](#)
- [Rules for Defining Subcircuit Global and Local Nets](#)

Subcircuit Name Missing Error

Simulation of a subcircuit named `16_inv`, for example, results in an error issued:

```
**error** (/remote/user/hspice/inv.sp:18) subcircuit name missing
```

HSPICE does not support subcircuit names that begin with a number. Subcircuit names must begin with an alpha character and can contain up to 1023 additional alphanumeric and special characters.

Duplicated Subckt Definition Error

Use `.OPTION REDEFSUB` to define how multiple subcircuit definitions in the netlist are treated.

The syntax for the REFDEFSUB option is as follows:

```
.OPTION REDEFSUB = [0 | 1 | 2]
```

The default value is 0 which causes HSPICE to issue an error message if there are multiple subcircuit definitions.

Set REFDEFSUB=1 for HSPICE to use the last subcircuit definition.

Set REFDEFSUB=2 for HSPICE to use the first subcircuit definition.

Duplicate Node Message

In the .SUBCKT definition, if HSPICE finds two or more node names that are the same, it issues the following error:

```
**error** subcircuit definition duplicates node node_name  
  
**error** .ends card missing at readin
```

In the following example, for the .SUBCKT definition, the node “in” gets two definitions. The second definition of the node “in” is a duplicate of the first node, which is illegal in HSPICE.:

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

Duplicating Ports

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

For example:

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

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

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

Using Assigned Circuit Numbers instead of Full Path Node Names

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

HSPICE assigns a circuit number to all subcircuits in the netlist. The circuit number abbreviates the path name, as follows: *subckt-num:name*

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

Simple Reporting of Floating Nodes

HSPICE can report simple instances of floating nodes when there is no path to ground. The *.lis file prints: ****warning**** only 1 connection at node. You can increase the number of warning messages printed by setting `.option warnlimit=n` in the netlist.

The list file reports nodes that have no DC path to ground (such as a cap with one node to gnd and no connect on other end), but there is no way to check for tri-state nodes.

Rules for Defining Subcircuit Global and Local Nets

The following rules apply (from lowest precedence to highest):

1. An internal subckt net with the same name as a global net is global:

```
.subckt res net1
R1 net1 gnd 100 $ local gnd is connected to global net gnd
.ends
```

2. Interface subckt pins on a subckt are always local:

```
.subckt res2 net1 gnd
R1 net1 gnd 100 $ local gnd is connected to local net net2,
                    not to global gnd
.end
X1 net1 net2 res2
```

3. An instance connected to a global net will make the subckt pin and its internal connections global:

```
.subckt res2 net1
R1 net1 gnd 100 $ local gnd is connected to global net gnd
                    through interface pin gnd
.end
X1 net1 gnd res2
```

Subcircuit Call Statement Discrete Device Libraries

The Synopsys Discrete Device Library (DDL) is a collection of HSPICE device models of discrete components, which you can use with HSPICE. The `$installdir/parts` directory contains the various subdirectories that form the DDL. Synopsys used its own ATEM discrete device characterization system to derive the BJT, MESFET, JFET, MOSFET, and diode models from laboratory measurements. The behavior of op-amp, timer, comparator, SCR, and converter models closely resembles that described in manufacturers' data sheets. HSPICE has a built-in op-amp model generator.

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

The following sections discuss these topics:

- [DDL Library Access](#)
- [Vendor Libraries](#)
- [Subcircuit Library Structure](#)

DDL Library Access

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

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

```
X1 2 1 D1N4004
```

Where `D1N4004` is the model name.

See [Element and Source Statements on page 89](#) and the *HSPICE Elements and Device Models Manual* for descriptions of element statements.

Optional parameter fields in the element statement can override the internal specification of the model. For example, for op-amp devices, you can override the offset voltage, and the gain and offset current. Because the DDL library

devices are based on HSPICE circuit-level models, simulation automatically compensates for the effects of supply voltage, loading, and temperature.

HSPICE accesses DDL models in several ways:

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

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

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

```
.INCLUDE path/T2N2211.inc
```

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

Vendor Libraries

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

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

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

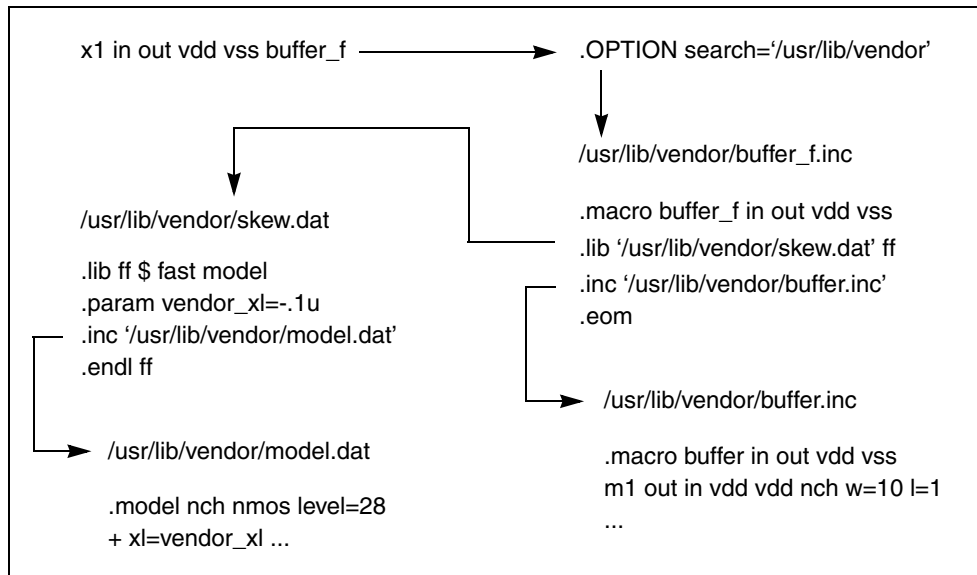


Figure 12 Vendor Library Usage

Subcircuit Library Structure

The `.OPTION SEARCH` command is an implicit include command. The include file can contain `.lib` calls in addition to the subcircuit definition. (See [Figure 12](#) for a graphic view.) Your library structure must adhere to the `.INCLUDE` statement specification in the implicit subcircuit. You can use this statement to specify the directory that contains the `subname.inc` subcircuit file, and then reference the `subname` in each subcircuit call.

The component naming conventions for each subcircuit is:

`subname.inc`

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

Create subcircuit libraries in a hierarchy. Typically, the top-level subcircuit fully describes the input/output buffer; any hierarchy is buried inside. The buried hierarchy can include model statements, lower-level components, and

parameter assignments. Your library cannot use `.LIB` or `.INCLUDE` statements anywhere in the hierarchy.

Chapter 5: Input Netlist and Data Entry
Subcircuit Call Statement Discrete Device Libraries

Using Interactive Mode

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

Note: HSPICE RF does not support interactive mode.

HSPICE ships hundreds of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

The following sections discuss these topics:

- [Invoking Interactive Mode](#)
- [Using Interactive Mode on the Windows Platform](#)
- [Examples](#)

Invoking Interactive Mode

Start interactive mode by using the `hspice` executable in the `$installdir/arch` directory (for example, `$installdir/sparcOS5`).

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

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

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

The following sections discuss these topics:

- [Quitting Interactive Mode](#)
- [Executing an Interactive Script](#)

Quitting Interactive Mode

You quit an interactive mode session by entering:

```
HSPICE > quit
```

Executing an Interactive Script

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

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

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

Using Interactive Mode on the Windows Platform

To start Windows Interactive Mode, open a Windows Command Prompt. Launch the command prompt window by entering `cmd` into **Start > Run** or through **Start > All Programs > Accessories**. From the command prompt, type:

```
C: \>hspice -I
```

The interactive environment functions from a special HSPICE shell. You can use the same commands as you do under UNIX. For example:

```
HSPICE > load case.sp  
HSPICE > run
```

To quit an interactive mode, enter:

```
HSPICE > quit
```

Examples

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

The following sections discuss these topics:

- [Getting Help](#)
- [Creating a Netlist](#)
- [Specifying an Analysis](#)
- [Running an Analysis](#)
- [Viewing a Netlist](#)
- [Loading and Running an Existing Netlist](#)
- [Using Environment Commands](#)
- [Recording and Saving Interactive Commands to a File](#)
- [Printing a Voltage Value During Simulation](#)
- [Using a Command File to Run in Interactive Mode](#)
- [Running Multiple Testcases](#)

Getting Help

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

```
% hspice -I
HSPICE > help

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

Creating a Netlist

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

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

Save the file and exit `vi`.

Specifying an Analysis

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

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

Running an Analysis

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

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

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

Viewing a Netlist

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

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

Loading and Running an Existing Netlist

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

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

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

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

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

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

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

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

Use the `run` command to simulate the netlist:

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

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

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

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

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

Chapter 6: Using Interactive Mode

Examples

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

```
HSPICE > quit
%
```

Using Environment Commands

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

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

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

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

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

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

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

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

Recording and Saving Interactive Commands to a File

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

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

Write all interactive commands entered to file int1.cmd:

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

Quit interactive mode and return to the system prompt.

Printing a Voltage Value During Simulation

This example prints a voltage value during simulation:

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

Chapter 6: Using Interactive Mode

Examples

```
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
v(1, 0) 0.000000
>info:          ***** hspice job concluded
HSPICE > quit
%
```

Using a Command File to Run in Interactive Mode

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

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

Running Multiple Testcases

To run multiple testcases and save the license check-in and check-out times, you can use the command file similar the following example. Use any text editor to create a command file named `bat.cmd` containing these entries:

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

Run HSPICE in interactive mode to execute `bat.cmd`:

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

HSPICE checks out a license only one time and then simulates both `tt.sp` and `qq.sp` netlists.

HSPICE GUI for Windows

Describes how to use the HSPICE GUI for Windows.

To open and install the GUI, click on the HSPUI icon. [Figure 13](#) shows the directory structure for the HSPICE GUI for Windows.

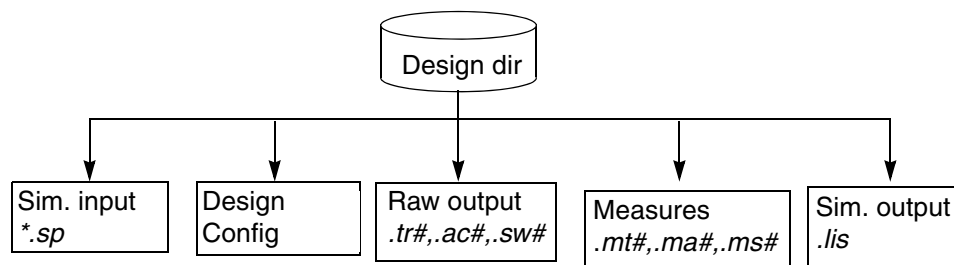


Figure 13 Directory Structure

This following section discuss these topics:

- [Working with Designs](#)
- [Configuring the HSPICE GUI for Windows](#)
- [Running Multiple Simulations](#)
- [SPutil, Metaencrypt and Converter Utilities, Client/Server Operation](#)
- [CMI Directory Structure](#)
- [Troubleshooting Guide](#)

Working with Designs

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

Select a file of the type `design.sp` to cause the Launcher to display the main form, which contains the following items:

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

Table 17 Design Commands in the Launcher

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

The Launcher checks the status of a given design when it opens. If the input file exists, the **Simulate** button is active. If the listing file exists for the design, the **Edit Listing** button is active. The **Edit Netlist** button is always active.

See [Figure 14 on page 131](#) for the main window of the Launcher.

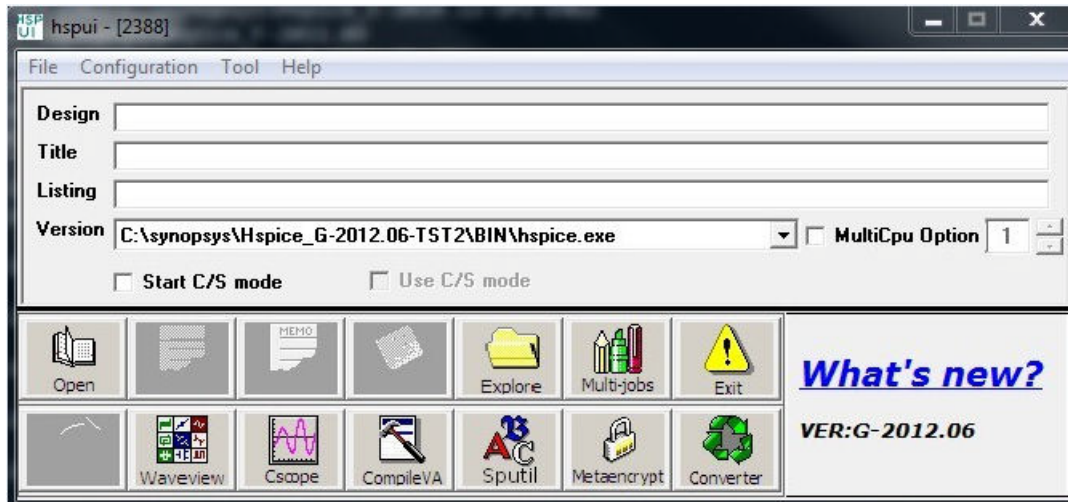


Figure 14 Launcher Main Window

Configuring the HSPICE GUI for Windows

Customize configurations by using the **Configuration** menu of the Launcher. To open the Launcher Options form (Figure 15) select **Configuration > Options**.

- The **Current Directory** defaults to the value of the *installdir* environment variable set up during HSPICE installation.
- The **Design (file) Suffix** defaults to `.sp`.

Chapter 7: HSPICE GUI for Windows

Configuring the HSPICE GUI for Windows



Figure 15 Launcher Options form

- The **Listing (file) Suffix** defaults to `.lis`.
- The **Default Editor** is `notepad.exe`.
- The **Browse** button enables navigation throughout the file tree.
- The check box enables: **Automatic save configuration when exit** (see below)
- The **Simulation window mode** panel presents radio button choices of:
 - **Popup** (default)
 - **Minimized**
 - **Silent**

If you change a value, the Launcher updates the `installdir/hspui.cfg` file. The location of the `hspui.cfg` file depends on which choices you make in the Launcher Options form. If `%APPDATA%/Synopsys/HSPICE/hspui.cfg` exists, HSPICE reads it, and ignores `installdir/hspui.cfg`. (HSPICE reads the `installdir/hspui.cfg` only if `%APPDATA%/Synopsys/HSPICE/hspui.cfg` does not exist.)

The `hspui.cfg` file location depends on the choice of **Save configuration at:**

1. If you select **Installed Path (root permission)**, the configuration is written to `installdir/hspui.cfg`.
2. If you select **User local HOME**, the configuration is written to `%APPDATA%/Synopsys/HSPICE/hspui.cfg`.

The configuration is written only if either:

1. If you select **Automatic save configuration when exit**, and the configuration is changed —or
2. You execute **File > Save Configuration**.

If you select **Configuration > Versions** you generate a list of the current executables for the path to the Launcher (HSPUI), HSPICE, and WaveView.

Note: The **Configuration > Versions** strings change from the main window **Version** selection box. You cannot change them here.

To associate your *design.sp* file with the Launcher, use the Launcher Options window. You can double-click an *.sp* file in the File Manager window to automatically invoke the HSPICE/Win Launcher.

For further configuration tips see [Setting the hspui.cfg File Values](#) and [Using Arguments on the Command Lines of the hspui.cfg](#)

Launching Waveview in HSPUI

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

1. Open the Versions form in the HSPUI.



Figure 16 Launcher Versions form

2. Configure the CustomWaveView executable (sx) in the Waveview field.

Chapter 7: HSPICE GUI for Windows

Configuring the HSPICE GUI for Windows

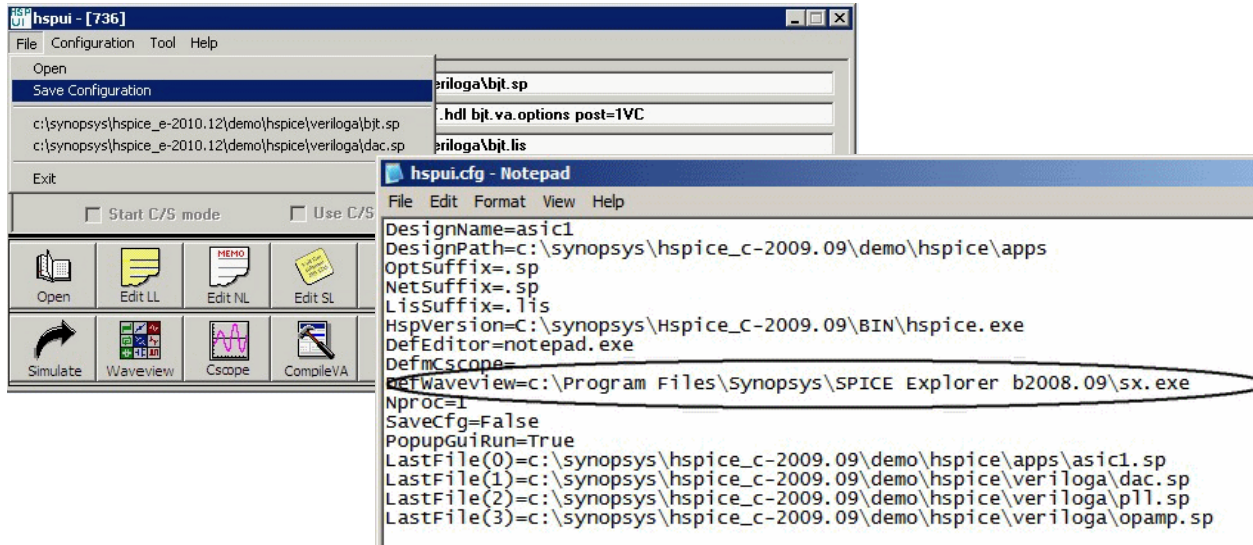


Figure 17 Opening the hspui.cfg file to confirm WaveView

3. Save configuration to hspui.cfg
4. Click the **Waveview** button in the HSPUI main window to launch Custom WaveView.

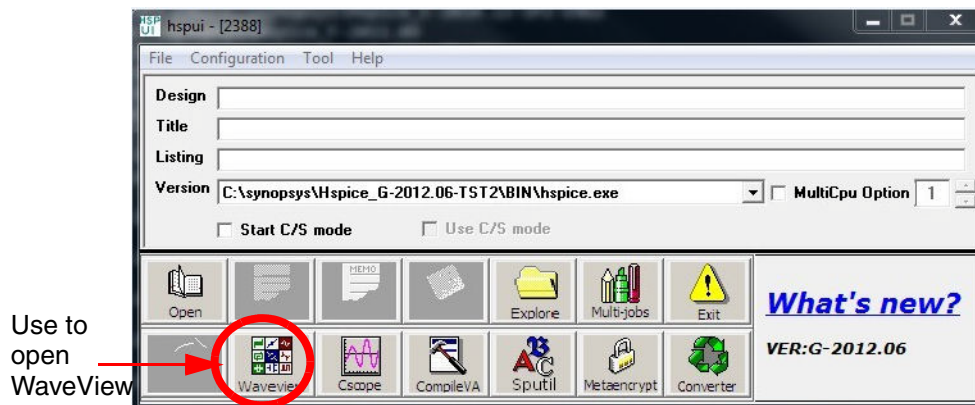


Figure 18 WaveView button

HSPICE automatically links existing simulation results into WaveView directly when you click to open the waveform viewer.

Note: Verify there is no whitespace in the path to the waveform file.

For HSPICE the UI checks for the following files:*.sw, *.ac, *.tr, *.ms, *.ma, *.mt, *.mcs, *.mca, *.mct, *.eps, *.epa, *.ept, and *.mex.

For HSPICE RF the UI checks for the following files:*.sw, *.ac, *.tr, *.ms, *.ma, *.mt, *.hb, *.sn, *.snf, *.hr, *.pn, *.snpn, *.snac, *.ls, *.ss, *.p2d, and *.ev.

Running Multiple Simulations

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

Click Multi-jobs in the main window to open the HSPICE Multi-jobs window (Figure 19 on page 135). Simulation files are chosen from the Drive/Directory list box and placed in the Files list box.

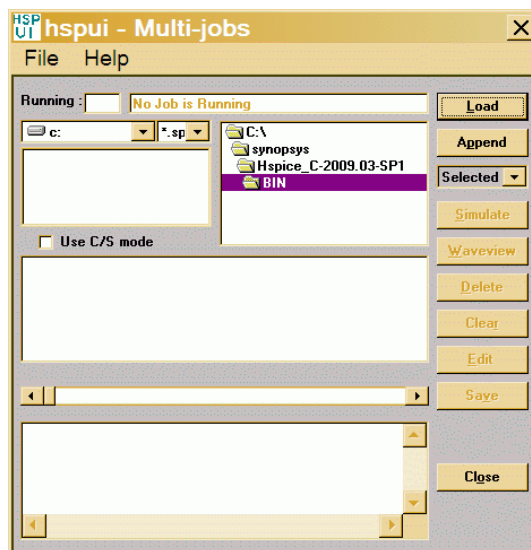


Figure 19 HSPUI Multi-jobs Window

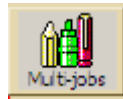
The following sections discuss these topics:

- [Building the Batch Job List](#)
- [Simulating a Batch Job](#)
- [Running Multithreading](#)

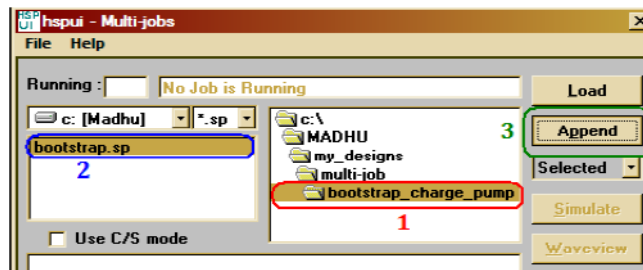
Building the Batch Job List

To build a batch job list:

1. Click Multi-jobs in the main window.



2. Using the Drive/Directory boxes, locate the directory of files that you wish to simulate.



1. Select directory of interest
2. Load file of interest
3. Click Append button

3. To load all files in the directory, click the **Load** button on the right side of the Multi-jobs window.

Note that any file names already in the list will be replaced.

4. To add additional files from other directories, repeat Step 2 and use the **Append** button.

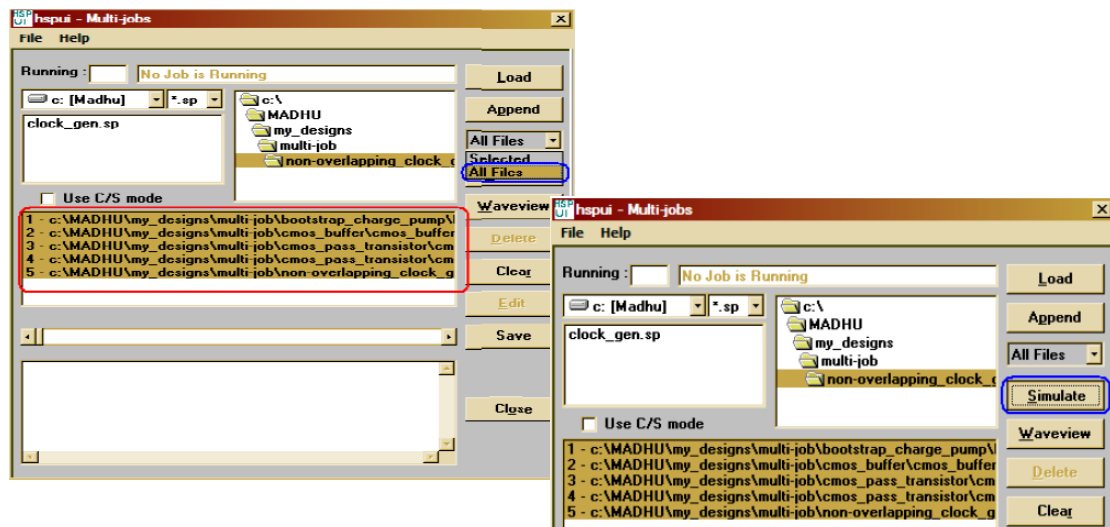
Simulating a Batch Job

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

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

To simulate a batch job list:

1. To simulate all of the files in the Batch Job list, set the pull-down menu to **All Files** and click the **Simulate** button.



2. To run simulation on a single file or a group of files, set the selection box option to **Selected** and select those files you wish to simulate from the batch job list box.

Use the left mouse button to select a single file.

- Press and hold the Control key and select another file with the left mouse button to add to the selected list.
- Press and hold the Shift key to select all files between the current file and the last selected file.

3. Click the **Simulate** button to start the consecutive simulations.

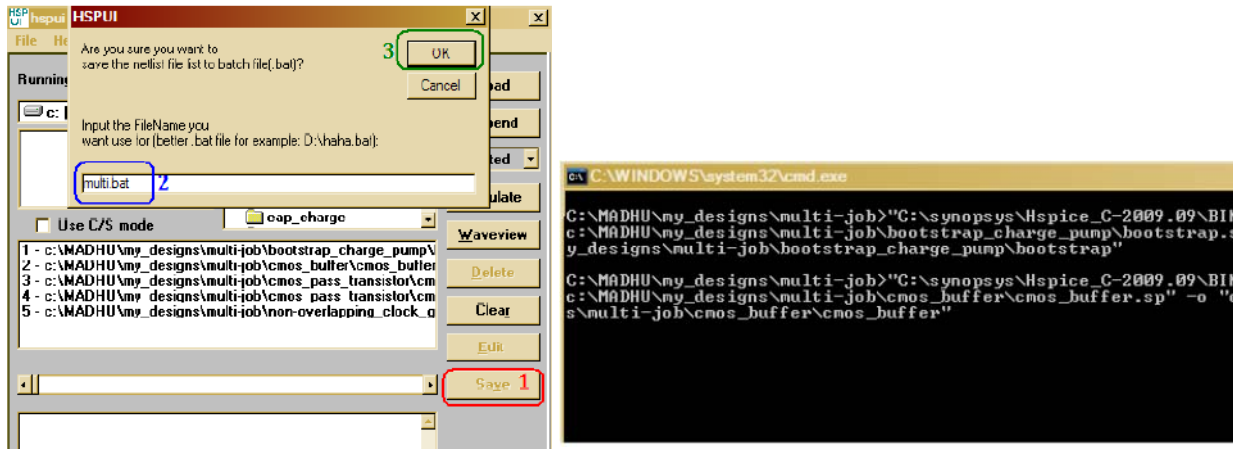
Sample Batch Work-Flow

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

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

Note: If you set the filter to *.* , file types other than HSPICE input decks are likely to be included.

2. Add files to the run list using the **Load** and **Append** buttons: When you navigate to the folder containing the SPICE decks, they are displayed in the left-hand list window. Although you can select files in this window, clicking the **Load** button adds all the files in the left-hand list window, as well as those in subdirectories. These files go into a run list and are numbered sequentially. **Load** clears the contents of the run list, but you can navigate to another folder containing HSPICE sources files and click the **Append** button to add additional files to the run list.
3. Edit the run list: Select files one at a time (CTRL-click) or a range of files (SHIFT-click) and click the **Delete** button to remove them from the run list.
Note: **Delete** does not remove the file from your hard-drive, just the run list. **Clear** removes the entire contents of the run list and allows you to start over again.
4. Perform text edits on individual HSPICE jobs: Selecting a job and clicking **Edit** displays that HSPICE deck in the text editor you designated in the HSPICE UI Configuration > Options dialog. Save and close it to continue.
5. Simulate the jobs: Only selected jobs in the run list can be simulated. Select (highlight) files by clicking on them, or by choosing **All Files** from the pull-down menu below the **Append** button. Click **Simulate** to run the jobs in order, one at a time.
6. Save the run list for later use: Using the **File > Save** or **File > Save As** pull-down menu items, you can save the contents of the run list to a file and open it with **File > Open** to begin work at a later time.
7. Invoke a waveform viewer graphical waveform analysis.
8. Create a batch (* .bat) file with a list of jobs to be run without using the UI: Click the **Save** button (not File > Save) to create a DOS batch file containing the full path to the HSPICE executable and the design name for each job in the run list. This .bat file can be executed from a DOS command prompt or by double-clicking on it in Windows.



Running Multithreading

To run multithreading:

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

SPutil, Metaencrypt and Converter Utilities, Client/Server Operation

For information about use of the S-parameter utility (SPutil) refer to [S-parameter Standalone Manipulation Utility](#) in Chapter 2 of the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

For information about use of the metaencrypt utility for encryption of files, refer to [Chapter 8, Library and Data Encryption](#).

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

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

CMI Directory Structure

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

Troubleshooting Guide

The following sections discuss these topics:

- [Setting the hspui.cfg File Values](#)
 - [Text Editor Issues](#)
 - [Simulating a UNIX Netlist File](#)
-

Setting the hspui.cfg File Values

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

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

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

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

```
DesiDesignName=  
DesignPath=  
NetSuffix=.sp  
LisSuffix=.lis  
HspVersion=C:\synopsys\Hspice_C-2009.03\BIN\hspice.exe  
DefEditor=C:\Program Files\Windows NT\Accessories\wordpad.exe  
DefmCscope=  
Nproc=1  
LastFile(0) =  
LastFile(1) =  
LastFile(2) =  
LastFile(3) =  
LastFile(4) =
```

- `DesignName` is the name of the last saved netlist run through HSPICE
- `DesignPath` is the path to the above netlist
- `OptSuffix` along with `NetSuffix` set suffix for netlist
- `NetSuffix` default is `.sp`
- `LisSuffix` sets suffix for output listing file, default `.lis`
- `DefmCscope` defines path to `Cscope` or other waveform viewer executable
- `DefWaveview` configures `WaveView` button to point to and open `WaveView`
- `DefEditor` path to editor for netlist and output files, default is Notepad
- `HspVersion` points to the default HSPICE version when HSPUI is opened
- `Nproc` defines the number of processors used when running HSPICE
- `SaveCfg` controls whether to save configuration automatically on exit
- `SimWinMode=2` controls window mode of `*.st0` file while HSPICE runs; set to 0 to hide window, set to 1 to set minimize it, set to 2 for popup window.
- `LastFile(#)` sets names of previously simulated netlists; up to 5 are saved

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

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

Using Arguments on the Command Lines of the `hspui.cfg`

You can use arguments on the command lines of the file `hspui.cfg`. You are required to put double-quotes around the path. You may include the command-line argument(s) in the quotes. The following are acceptable ways to write the paths for `DefWaveview` and `DefmCscope` in the `hspui.cfg`:

```
DefWaveview="c:\Program Files\Synopsys\SPICE Explorer  
c2009.03\sx.exe" -k -w  
DefWaveview="c:\Program Files\Synopsys\SPICE Explorer  
c2009.03\sx.exe -k -w"  
DefmCscope="C:\Synopsys\c2009.03\ai_bin\cscope.exe"
```

Note: The hspui.cfg file is located in the folder: C:\Documents and Settings\user_name\Application Data\Synopsys\HSPICE

Text Editor Issues

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

- With Notepad open, click Format on the tool bar and uncheck Word Wrap.
- Configure the HSPUI to use another text editor to view the files:
- From the HSPUI click Configuration > Options.
- For the Default Editor field, click the Browse button.
- Navigate to the .exe for desired text editor.

Simulating a UNIX Netlist File

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

Running AvanWaves Using the Cscope Button

WaveView is now the preferred viewer for HSPICE output. However, if you have the INCREMENT metawaves key in your license, you can configure the CScope button to invoke Avanwaves.

In the HSPUI:

1. Click Configuration.
2. Select VERSIONS.

3. In the CScope field enter path to Avanwaves: For example:
`DefmCscope=C:\Synopsys\Hspice_C-2009.03\BIN\awaves.exe`
4. In the hspui.config file (C:\Documents and Settings\user\Application Data\Synopsys\HSPICE\hspui.cfg), edit the line `DefmCscope=to` point to `awaves.exe` in the HSPICE installation directory.
5. Click the Cscope button. You are now be able to launch AvanWaves.

Chapter 7: HSPICE GUI for Windows
Troubleshooting Guide

Library and Data Encryption

Describes the Synopsys library encryption methods and their use to protect your intellectual property.

HSPICE ships several suites of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files; for encryption demo files, see [Encryption Examples](#).

Organization

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

- [Library Encryption](#)
- [Three Encryption Methods](#)
- [Installing and Running metaencrypt](#)
- [Encryption Guidelines](#)
- [General Example](#)
- [Traditional Library Encryption](#)
- [8-Byte Key Encryption](#)
- [Triple DES Public and Random Keys](#)
- [Troubleshooting](#)

Library Encryption

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

The following topic discusses the metaencrypt utility:

- [Encrypting a Model Library Using the metaencrypt Utility](#)

Encrypting a Model Library Using the metaencrypt Utility

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

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

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

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

1. The modified structure should be as follows:

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

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


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

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

```
* top level netlist  
.inc mynetlist.sp.enc  
.lib mylib.lib tt  
...  
.end  
  
* modified library mylib.lib  
.lib tt  
.inc myparam.par.enc  
.inc mymodel.mdl.enc  
.end
```

Three Encryption Methods

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

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

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

Installing and Running metaencrypt

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

The following sections discuss these topics:

- [Installing metaencrypt](#)
- [Running metaencrypt](#)
- [Encryption Guidelines](#)

- [General Example](#)
- [Traditional Library Encryption](#)
- [Creating Files Using Traditional Encryption](#)
- [Example: Traditional \(freelib\) Encryption in an HSPICE Netlist](#)

Installing metaencrypt

The metaencrypt utility is part of the general HSPICE distribution and found in the `$installdir/bin` directory.

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

Running metaencrypt

Syntax

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

Argument	Description
-i <i>inptfileName</i>	Unencrypted input filename.
-pipe	Reads in data from a UNIX pipe.
-o <i>outfileName</i>	Encrypted output filename.
-t <i>encrypt_type</i>	Encryption method: <ul style="list-style-type: none">▪ Freelib—weak (low security) -t [ddl1 ddl2 custom freelib]▪ 8-byte—strong (medium security) -t <i>8-byte-string</i>▪ Triple DES—strongest (high security) -t [privkey [<i>key</i> or <i>file</i>] randkey]

Argument	Description
<code>-d encrypt_dir</code>	Valid when using DES or tripleDES to encrypt file. Data between <code>.PROT</code> and <code>.UNPROT</code> commands undergoes encryption and outputs to an encrypted output file in directory <code>encrypt_dir</code> .
<code>-r synopsys...tool...</code>	<p><code>synopsys_tool</code> can be: HSPICE, NanoSim, HSIM, CustomSim, or Synopsys. <code>access_control</code> can be 0 or 1. The default is 0.</p> <ul style="list-style-type: none"> ▪ 0: The simulators suppress any warning-related information to the encrypted block. The default value is ON for all simulators, i.e., if you do not use <code>-r</code>, or if you specify <code>-r</code> without a value after it, Synopsys tools (NanoSim, HSIM, XA, and HSPICE) can read the encrypted file. ▪ 1: Simulators do not suppress warning messages from the encrypted block. <p>Notes:</p> <p>HSPICE can always read an encrypted file, even if you have not specified <code>-r hspice</code>. If there are multiple settings for the same tool, HSPICE uses the last setting.</p> <p>Example 1: In this invocation, the <code>access_control</code> for NanoSim is 0 and the <code>access_control</code> for all other simulators is 1.</p> <pre>.metaencrypt -i test.sp -o test.spe -t randkey + -r synopsys:1 -r nanosim:0</pre> <p>Example 2: Limit the parsing of the encrypted files to HSPICE. Other simulators cannot parse the encrypted file</p> <pre>metaencrypt -i test.sp -o test.spe -t randkey + -r hspice</pre>

Encryption Guidelines

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

You can use any legal HSPICE command inside subcircuits that you encrypt. Refer to [Using Subcircuits](#) in [Chapter 5, Input Netlist and Data Entry](#) for more information about how to construct subcircuits. The structure of your libraries can affect how you encrypt them. If your library requires that you change the name of a subcircuit, you must encrypt that circuit again.

To encrypt more than one file in a directory, use the following shell script, which encrypts the files as a group. In this example, the script uses the traditional

Chapter 8: Library and Data Encryption

Installing and Running metaencrypt

encryption method. The script produces a `.inc` encrypted file, for each `.dat` file in the current directory. The `metaencrypt` command assumes that unencrypted files have a `.dat` suffix.

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

Use an encrypted file much the same way as you do before encryption. The name of the file may be different, however, and so you may need to update the `.include` and `.lib` commands.

Note: Verilog-A supports module files for encryption only when using 8byte key and tripleDES private or random key. The Freelib (traditional encryption method) does not support Verilog-A due to limitations in metaencrypt.

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

General Example

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

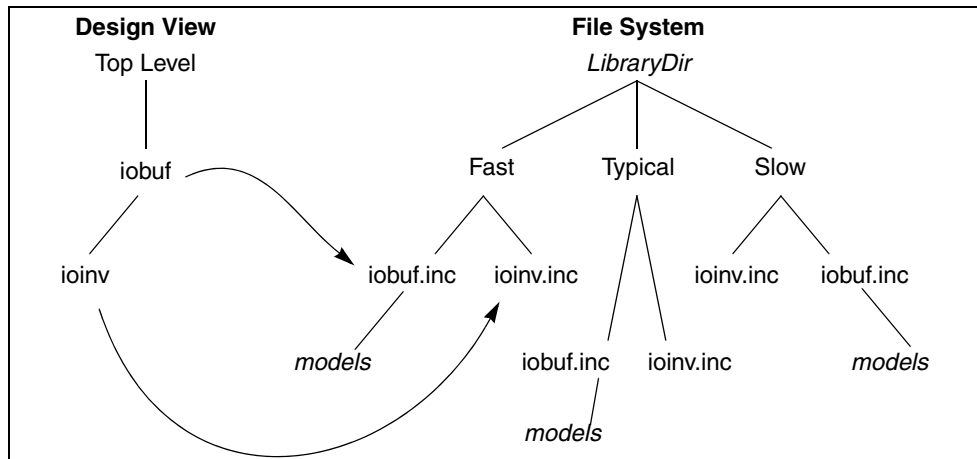


Figure 20 Encrypted Library Structure

The following example describes an encrypted I/O buffer library subcircuit. This subcircuit consists of several subcircuits and model commands that you need to protect with encryption. [Figure 20 on page 151](#) shows the organization of subcircuits and models, in the libraries used in this example.

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

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

The *LibraryDir/Fast/iobuf.inc* file contains:

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

Chapter 8: Library and Data Encryption

Installing and Running metaencrypt

The *LibraryDir/Fast/oinv.inc* file contains:

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

The encrypted file looks similar to the following:

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

.ENDS
```

Note: The `.UNPROTECT` statement also becomes encrypted during the encryption process.

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

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

Using the `-pipe` Option

With the 2012.06 release, you can use input from a UNIX pipe so you can read in from a command series such as the following:

```
FILE *pipe;
char buf[100];
pipe = popen("metaencrypt -o outfile ....", "w");
strcpy(buf, ".....");
fprintf(pipe, buf);
strcpy(buf, ".....");
fprintf(pipe, buf);
pclose(pipe);
```

After the pipe closes, HSPICE generates the encrypted file.

Traditional Library Encryption

A 5-rotor Enigma machine is the basis for the traditional library encryption algorithm (Freelib). You can specify which portions of subcircuits to encrypt.

The `.PROT/.UNPROT` commands designate encrypted portions. HSPICE does not encrypt any other netlist entries. Library encryption uses a key value, which HSPICE reconstructs for decryption.

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

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

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

Example

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

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

Creating Files Using Traditional Encryption

The following sections describe:

- [Non-Library Encrypted Portions](#)
- [*.lib File Encryption](#)

Non-Library Encrypted Portions

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

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

The following is an example of an `.sp` file:

Chapter 8: Library and Data Encryption

Installing and Running metaencrypt

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

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

*.lib File Encryption

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

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

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

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

```
*sample.lib*
.lib test1 $ .prot , .unpr should be put between
* .lib and .endl
.prot
..... $ data to be encrypted
.unpr
..... $ data not to be encrypted
.end1 test1

.lib test2 $ .prot , .unpr should be put between
* .lib and .endl
.prot
..... $ data to be encrypted
.unpr
.end1 test2

.lib test3
..... $ data
.end1 test3
```

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


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

Example: Traditional (freelib) Encryption in an HSPICE Netlist

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

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

```
file mm.spe:
.prot CUSTOM
-hs#y1B]*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$0) :P.#$<) :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
```

Chapter 8: Library and Data Encryption

8-Byte Key Encryption

```
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
HO[I: [
.ENDL

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

8-Byte Key Encryption

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

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

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

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

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

- 8-byte key encryption supports only .inc encryption.
- 8-byte encryption does NOT support .LIB, .LOAD, or .OPTION SEARCH encryption. Choose another form of encryption for these types of files.
- If keyname is an 8-byte string (combination of characters and numbers), then metaencrypt performs the 8-byte key encryption.
- In a .sp file, you cannot encrypt the first line because it is the title. You also cannot encrypt the last line because it marks the end of the file.

The following sections discuss these topics:

- [Creating 8-byte key Encryption](#)
- [Placing an 8-byte key Encrypted File into a HSPICE Netlist](#)

Creating 8-byte key Encryption

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

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

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

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

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

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

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

Triple DES Public and Random Keys

Note: Older releases of HSPICE cannot read files encrypted with the Triple DES algorithm.

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

- 256-bit public key

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

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

The 256-bit public key supports multiple encrypted files. You have to put all relevant public key files in the directory from which the simulation originates. You can also generate these encrypted files with the same private key or

different private keys. The actual “key” needs to be a user-generated 192-bit string. For example, the following two files receive encryption with the same key:

```
metaencrypt -i a.dat -o a.inc -t privkey  
metaencrypt -i b.dat -o b.inc -t privkey
```

The encryption creates two public key files: a.inc.key, and b.inc.key. These files are different even when the same private key generates the encryption. A simulation run with a netlist file containing `.include a.inc` and `.include b.inc` commands requires that both key files be in the simulation directory.

- 192-bit random key
With this option, HSPICE does not need an additional public key. Usage is consistent with other encryption options in previous releases of HSPICE. The encrypted file is different every time you run an encryption.
- Embedded `.INC` encryption, which causes confusion in decryption, is not supported by 3DES.

The following sections discuss these topics:

- [Creating 3DES Encrypted Files](#)
- [Placing 3DES Encryption Files into a HSPICE Netlist](#)

Creating 3DES Encrypted Files

Observe these rules when creating TripleDES encrypted files:

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

Random Key Example

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

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

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

Chapter 8: Library and Data Encryption

Triple DES Public and Random Keys

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

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

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

Public Key Example

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

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

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

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

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

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

Placing 3DES Encryption Files into a HSPICE Netlist

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

Example 1 Random TripleDES

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

Example 2 Private TripleDES

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

Troubleshooting

The following sections discuss these issues:

- [**warning** parameters... as an expression containing output signals](#)
- [Encrypting S-parameter files](#)
- [Freelib Issue with Equations](#)

****warning** parameters... as an expression containing output signals**

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

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

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

Note: If you use `.prot/ .unprot` in a library or file that is unencrypted you get warnings that the file is an encrypted file and the file or library is a “black box.”

Encrypting S-parameter files

While S-parameters do not convey any IP information and encryption you do not need to add encryption, you can encrypt S-parameters and use in HSPICE simulations if the file is in the SELEM format (`*.sc0`).

To encrypt a SELEM-formatted S-parameter file and use it in HSPICE, follow this procedure:

1. Generate a `*.sc0` file. Use the `.LIN` command to extract the S-parameters from a circuit. The `.LIN` command creates a `.sc0` file. If you already have a TouchStone 1.0/2.0 or CITI formatted file, you can use also use the `.LIN` command to convert the file to an `*.sc0` formatted file.
2. Edit the `.sc0` file so that it is a `.lib` file. For example,

```
.lib s
.protect
*| NPort=2 DATA=1000 COMPLEX_DATAFORMAT=RI NOISE=0
GROUPDELAY=0
*| NumOfBlock=1 NumOfParam=0
*
.MODEL filter S
+ N=2 FQMODEL=SFQMODEL TYPE=S Zo= 50.0000 50.0000
* + FBASE= FMAX=
.MODEL SFQMODEL SP N=2 SPACING=POI INTERPOLATION=LINEAR
MATRIX=NONSYMMETRIC
+ DATA=1000
+ ...
<S-parameter data>
.unprotect
.endl
```

3. Encrypt the file. Use either 8-bit encryption or Triple DES encryption.
4. To use the encrypted file in the netlist you need to call the S-parameter data as a `.lib` in addition to defining the S-element.


```
* Encrypted S-parameter Example
...
.lib 'filter.sdt' s          $ encrypted S-parameter library file
                           $ contains the model 'filter'
S1 in out 0 mname=filter    $ S-element with model name 'filter'
...
.end
```

Freelib Issue with Equations

When you run encrypted files using the freelib method, you may generate error messages such as:

```
**error** Left parenthesis improperly placed in column 90
**error** Characters after end of expression in column 148
```

The error messages only occur when using freelib encryption because this method does not support the mathematical operator caret (^) used for power in equations. If you are using freelib encryption, use '**' as an operator for power in equations.

HSPICE recommends that you use either 8-byte or Triple DES encryption as both the 8-byte and Triple DES encryption are much stronger encryption methods than the freelib encryption method.

Chapter 8: Library and Data Encryption
Troubleshooting

Part 2: Elements and Devices

The following chapters/topics are included in this Part:

- [Chapter 9, Elements](#)
- [Chapter 10, Sources and Stimuli](#)
- [Chapter 11, Parameters and Functions](#)
- [Chapter 12, Simulation Output](#)

Elements

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

HSPICE ships hundreds of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files. See [Benchmark Examples](#) and [Applications of General Interest Examples](#) for demo files showing usage of elements.

Elements are local and sometimes customized instances of a device model that you specified in your design netlist. Elements (instances) use standard device models as their bases. For descriptions of the standard device models, see the [HSPICE Reference Manual: Elements and Device Models](#) and the [HSPICE Reference Manual: MOSFET Models](#). For signal integrity applications see the [HSPICE User Guide: Signal Integrity Modeling and Analysis](#).

Element names can be up to 1024 characters.

The following sections discuss these topics:

- [Passive Elements](#)
- [Multi-Terminal Linear Elements](#)
- [Port Element](#)
- [Active Elements](#)
- [IBIS Buffers \(HSPICE Only\)](#)

Passive Elements

This section describes the passive elements: resistors, capacitors, and inductors. See [Multi-Terminal Linear Elements](#) for discussion of the W-, U-, and

S-elements. See also, [T-element \(Ideal Transmission Lines\)](#) in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

The content of this section includes:

- [Values for Elements](#)
- [Resistor Elements in a HSPICE or HSPICE RF Netlist](#)
- [Capacitors](#)
- [Inductors](#)

Values for Elements

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

Resistor Elements in a HSPICE or HSPICE RF Netlist

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

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

Parameter	Description
Rxxx	Resistor element name. Must begin with R, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
n1	Positive terminal node name.
n2	Negative terminal node name.
mname	Resistor model name. Use this name in elements, to reference a resistor model.

Parameter	Description
TC	TC1 alias. The current definition overrides the previous definition.
TC1	First-order temperature coefficient for the resistor. See the Passive Device Models chapter in the <i>HSPICE Elements and Device Models Manual</i> for temperature-dependent relations.
TC2	Second-order temperature coefficient for the resistor.
SCALE	Element scale factor; scales resistance and capacitance by its value. Default=1.0.
R=resistance	Resistance value at room temperature. This can be: <ul style="list-style-type: none"> ▪ a numeric value in ohms ▪ a parameter in ohms
M	Multiplier to simulate parallel resistors. For example, for two parallel instances of a resistor, set M=2, to multiply the number of resistors by 2. Default=1.0.
AC	Resistance for AC analysis. Default=Reff.
DTEMP	Temperature difference between the element and the circuit, in degrees Celsius. Default=0.0. To modify the temperature for a particular element, use the DTEMP parameter in an instance line.
L	Resistor length in meters. Default=0.0, if you did not specify L in a resistor model.
W	Resistor width. Default=0.0, if you did not specify W in the model.
C	Capacitance connected from node n2 to bulk. Default=0.0, if you did not specify C in a resistor model
user-defined equation	Can be a function of any node voltages, element currents, temperature, frequency, or time
NOISE	<ul style="list-style-type: none"> ▪ NOISE=0, do not evaluate resistor noise. ▪ NOISE=1, evaluate resistor noise (default).
R= 'equation'	<ul style="list-style-type: none"> ▪ a function of any node voltages ▪ a function of branch currents ▪ any independent variables such as time, hertz, and temper

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

For output templates for resistor models, see [Table 32 on page 438](#).

HSPICE Examples

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

```
R1 Rnode1 Rnode2 100
```

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

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

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

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

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

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

HSPICE RF Examples

Some basic examples for HSPICE RF include:

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

```
R1 1 0 'v(c)'
```
- R2 is a resistor whose resistance is the sum of the absolute values of nodes c and d.

```
R2 1 0 'abs(v(c)) + abs(v(d))'
```
- R3 is a resistor whose resistance is the sum of the `rconst` parameter, and 100 times `tx1` for a total of 1100 ohms.

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

Linear Resistors

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


+ [C=val] [DTEMP=val] [SCALE=val]

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

Example

```
R1 1 2 10.0
Rload 1 GND RVAL

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

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

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

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

Behavioral Resistors

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

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

Example

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

Frequency-Dependent Resistors

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

Parameter	Description
CONVOLUTION	<p>Indicates which method to use (at the instance level only).</p> <ul style="list-style-type: none"> ▪ 0: Acts the same as the conventional method. This is the default. ▪ 1 : Applies recursive convolution, and if the rational function is not accurate enough, it switches to linear convolution. ▪ 2 : Applies linear convolution.
FBASE	<p>Specifies the lower bound of the transient analysis frequency. For CONVOLUTION=1 mode, HSPICE starts sampling at this frequency. For CONVOLUTION=2 mode, HSPICE uses this value as the base frequency point for Inverse Fourier Transformation.</p> <p>For recursive convolution, the default value is 0Hz, and for linear convolution, HSPICE uses the reciprocal of the transient period.</p>
FMAX	<p>Specifies the possible maximum frequency of interest. The default value is the frequency point where the function reaches close enough to infinity value. This assumes that frequency point reaches 10THz and the monotonous function is approaching the infinity value.</p> <p>The equation should be a function of HERTZ. If CONVOLUTION turns on when a HERTZ keyword is not in the equation, it is automatically turns off to let the resistor behave conventionally. The equation can be a function of temperature, but it cannot be node voltage or branch current and time.</p>

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

You can model a frequency-dependent resistor and use it in the time domain only when CONVOLUTION=1.

Example

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

Skin Effect Resistors

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

The R_s indicates the skin effect coefficient of the resistor.

[Equation 1](#) shows the expression of the complex impedance of the resistor:

Equation 1

$$R(f) = R_0 + (1 + j) \cdot R_s \cdot \sqrt{f}$$

Where, R_0 is the DC resistance, j is the imaginary term and f is the frequency. The imaginary part of the equation represents the frequency-dependent inductance.

In the time domain, HSPICE uses the following rational function to represent [Equation 1](#):

Equation 2

$$H(\omega) \cong \sum_k \frac{\alpha_k}{\omega_k + j\omega}$$

Recursive convolution evaluates the rational function skin effect resistor. An advantage of using the skin effect resistor is that it automatically guarantees the causality.

Capacitors

For a full demonstration example of voltage variable capacitance see the path to the `calg2.sp` netlist noted in the section [Behavioral Application Examples](#).

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

Polynomial form:

Chapter 9: Elements

Passive Elements

```
Cxxx n1 n2 POLY c0 c1... [IC=val] [M=val]
```

Parameter	Description
Cxxx	Capacitor element name. Must begin with C, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
n1	Positive terminal node name.
n2	Negative terminal node name.
mname	Model name; mname must refer to a capacitor model defined in the netlist using a .MODEL construct.
C=capacitance	Capacitance at room temperature—a numeric value or a parameter in farads.
TC1	First-order temperature coefficient for the capacitor. See the Passive Device Models chapter in the <i>HSPICE Elements and Device Models Manual</i> for temperature-dependent relations.
TC2	Second-order temperature coefficient for the capacitor.
SCALE	Element scale parameter, scales capacitance by its value. Default=1.0.
IC	The initial voltage across the capacitor, in volts when the UIC parameter in the .TRAN statement is used. When the UIC parameter is used, HSPICE does not calculate the initial DC operating point and the voltage will be across the capacitor at transient simulation <code>time=0</code> and will be released at <code>time>0</code> to allow the capacitor voltage to vary with circuit operation. If you use an .IC statement to set an initial voltage across the capacitor, then the .IC statement overrides the IC value set on the instance.
M	Multiplier, used to simulate multiple parallel capacitors. Default=1.0
W	Capacitor width, in meters. Default=0.0, if you did not specify W in a capacitor model.
L	Capacitor length, in meters. Default=0.0, if you did not specify L in a capacitor model.
DTEMP	Element temperature difference from the circuit temperature, in degrees Celsius. Default=0.0.
C='equation'	Capacitance at room temperature, specified as a function of <ul style="list-style-type: none"> ■ any node voltages ■ any branch currents ■ any independent variables such as <code>time</code>, <code>hertz</code>, and <code>temper</code>

Parameter	Description
CTYPE	Determines capacitance charge calculation for elements with capacitance equations. If the C capacitance is a function of $V(n1[,n2])$, set CTYPE=0. Use this setting correctly, to ensure proper capacitance calculations, and correct simulation results. Default=0.
POLY	Keyword, to specify capacitance as a non-linear polynomial.
c0 c1...	Coefficients of a polynomial, described as a function of the voltage across the capacitor. c0 represents the magnitude of the 0th order term, c1 represents the magnitude of the 1st order term, and so on. You cannot use parameters as coefficient values.

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

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

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

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

For output templates for capacitor models, see [Table 33 on page 439](#).

Example 1

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

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

Example 2

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

```
C1 1 2 20p
```

Chapter 9: Elements

Passive Elements

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

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

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

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

The C99 capacitor connects from the in node to the out node. The polynomial $C=c_0 + c_1*v + c_2*v*v$, where v is the voltage across the capacitor, determines the capacitance.

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

Linear Capacitors

```
Cxxx node1 node2 [modelname] [C=] val [TC1=val]  
+ [TC2=val] [W=val] [L=val] [DTEMP=val]  
+ [M=val] [SCALE=val] [IC=val] [SHRINK=val]
```

The value of a linear capacitor can be a constant, or an expression of parameters.

Parameter	Description
Cxxx	Name of a capacitor. Must begin with C, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
node1 and node2	Names or numbers of connecting nodes.
value	Nominal capacitance value, in Farads.
modelname	Name of the capacitor model.
C	Capacitance, in Farads, at room temperature.
TC1, TC2	Specifies the temperature coefficient.
W	Capacitor width.
L	Capacitor length.
M	Multiplier to simulate multiple parallel capacitors.
DTEMP	Temperature difference between element and circuit.

Parameter	Description
SCALE	Scaling factor.
IC	Initial capacitor voltage across the capacitor.
SHRINK	Local shrink factor; a value of shrink=1 disables shrinking. See .OPTION SHRINK for more information.

Example

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

In this example:

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

Frequency-Dependent Capacitors

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

Syntax

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

Parameter	Description
n1 n2	Names or numbers of connecting nodes.

Parameter	Description
equation	Expressed as a function of HERTZ. If CONVOLUTION=1 or 2 and HERTZ is not in the equation, CONVOLUTION turns off and the capacitor behaves conventionally. The equation can be a function of temperature, but it does not support variables of node voltage, branch current, or time. If these variables exist in the expression and CONVOLUTION=1 or 2, then the simulation considers only their values at the operating point in calculation.
CONVOLUTION	Specifies the method used. <ul style="list-style-type: none"> ▪ 0 (default): HERTZ=0 in time domain analysis. ▪ 1 or 2: performs Inverse Fast Fourier Transformation (IFFT) linear convolution.
FBASE	Base frequency to use for transient analysis. This value becomes the base frequency point for Inverse Fast Fourier Transformation (IFFT) when CONVOLUTION=1 or 2. If you do not set this value, the base frequency is a reciprocal value of the transient period.
FMAX	Maximum frequency to use for transient analysis. Used as the maximum frequency point for Inverse Fourier Transformation. If you do not set this value, the simulation takes the reciprocal value of RISETIME.

Example

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

Behavioral Capacitors in HSPICE or HSPICE RF

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

Parameter	Description
CTYPE	Determines the calculation mode for elements that use capacitance equations. Set this parameter carefully, to ensure correct simulation results. HSPICE RF extends the definition and values of CTYPE, relative to HSPICE: <ul style="list-style-type: none"> ▪ CTYPE=0, if C depends only on its own terminal voltages—that is, a function of V(n1[, n2]). ▪ CTYPE=1, if C depends only on outside voltages or currents. ▪ CTYPE=2, if C depends on both its own terminal and outside voltages. This is the default for HSPICE RF. HSPICE does not support CTYPE=2.

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

Example

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


DC Block Capacitors

```
Cxxx node1 node2 [C=] INFINITY [IC=val]
```

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

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

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

DC Block and Choke Elements

In HSPICE RF, you can specify an `INFINITY` value for capacitors and inductors to model ideal DC block and choke elements. The following input syntax is for the DC block (ideal infinite capacitor):

Syntax

```
Cxxx node1 node2 [C=] INFINITY [IC=val]
```

HSPICE RF does not support any other capacitor parameters for DC block elements, because HSPICE RF assumes that the infinite capacitor value is independent of temperature and scaling factors. The DC block acts as an open circuit for all DC analyses. HSPICE RF calculates the DC voltage across the circuit’s nodes. In all other (non-DC) analyses, a DC voltage source of this value represents the DC block (that is, HSPICE RF does not then allow dv/dt variations).

The following input syntax is for the Choke (ideal infinite inductor):

Syntax

```
Lxxx node1 node2 [L=] INFINITY [IC=val]
```

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

Charge-Conserved Capacitors

`Cxxx node1 node2 q='expression'`

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

The `expression` supports the following parameters and variables:

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

Note: Transient analyses do not support the `hertz` variable.

You must use parameters directly in an equation. HSPICE does not support parameters that represent an equation containing variables.

Error Handling

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

Limitations

The following syntax does not support charge-conserving capacitors:

`Cxx node1 node2 C='expression'`

HSPICE does not implicitly convert capacitor equations to charge equations.

Example 1: Capacitance-based Capacitor

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

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

Example 2: Charge-based Capacitor

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

Example 3: Capacitance-based Capacitor

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

Example 4: Charge-based Capacitor

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

Inductors

For demonstration examples of magnetics netlists, see [Magnetics Examples](#) demo files for magnetic cores, L-elements and K-elements. This link provides paths to several files available from the HSPICE installation directory.

General form:

```
Lxxx n1 n2 [L=inductance] [TC1=val]
+ [TC2=val] [SCALE=val] [IC=val] [M=val]
+ [DTEMP=val] [R=val]
Lxxx n1 n2 L='equation' [LTYPE=val] [above_options...]
```

Polynomial form:

```
Lxxx n1 n2 POLY c0 c1... [above_options...]
```

Magnetic winding form:

```
Lxxx n1 n2 NT=turns [above_options...]
```

Parameter	Description
Lxxx	Inductor element name. Must begin with L, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .

Chapter 9: Elements

Passive Elements

Parameter	Description
n1	Positive terminal node name.
n2	Negative terminal node name.
TC1	First-order temperature coefficient for the inductor. See the Passive Device Models chapter in the <i>HSPICE Reference Manual: Elements and Device Models</i> 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	The current forced through the inductor for the duration of the DC operating point calculation, in amperes. HSPICE uses this value as the DC operating point current only. If the <code>.TRAN</code> statement uses UIC, then HSPICE does not calculate the DC operating point. For this case, the current will flow through the inductor at transient simulation <code>time=0</code> and will be released at <code>time>0</code> to allow the inductor current to vary with circuit operation.
L=inductance	Inductance value. This can be: <ul style="list-style-type: none"> ▪ a numeric value, in henries ▪ a parameter in henries
M	Multiplier, used to simulate parallel inductors. Default=1.0.
DTEMP	Temperature difference between the element and the circuit, in degrees Celsius. Default=0.0.
R	Resistance of the inductor, in ohms. Default=0.0.
L='equation'	Inductance at room temperature, specified as: <ul style="list-style-type: none"> ▪ a function of any node voltages ▪ a function of branch currents ▪ any independent variables such as <code>time</code>, <code>hertz</code>, and <code>temper</code>
LTYPE	Calculates inductance flux for elements, using inductance equations. If the L inductance is a function of I(Lxxx), then set LTYPE=0. Otherwise, specify LTYPE=1. Use this setting correctly, to ensure proper inductance calculations, and correct simulation results. Default=0.
POLY	Keyword that specifies the inductance, calculated by a polynomial.
c0 c1...	Coefficients of a polynomial in the current, describing the inductor value. c0 is the magnitude of the 0th order term, c1 is the magnitude of the 1st order term, and so on.
NT=turns	Number of turns of an inductive magnetic winding.

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

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

For a listing of output templates for inductor (L-element) models, see [Table 34 on page 439](#).

Example 1

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

```
L1 coilin coilout 100n
```

Example 2

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

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

Example 3

The Lcoil inductor connects from the input node to ground. The product of the current through the inductor, and 1E-6, determines its inductance.

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

Example 4

The L99 inductor connects from the in node to the out node. The polynomial $L=c_0 + c_1*i + c_2*i^2$, where i is the current through the inductor, determines its inductance. The inductor also has a specified DC resistance of 10 ohms.

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

Example 5

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

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

Linear Inductors

```
Lxxx node1 node2 [L =] inductance [TC1=val] [TC2=val]
```

Chapter 9: Elements

Passive Elements

+ [M=val] [DTEMP=val] [IC=val]

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

Example:

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

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

Frequency-Dependent Inductors

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

Syntax

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

Parameter	Description
Lxxx	Inductor element name. Must begin with L, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70
n1 n2	Positive and negative terminal node names.

Parameter	Description
equation	The equation should be a function of HERTZ. If CONVOLUTION turns on when a HERTZ keyword is not in the equation, CONVOLUTION automatically turns off and the inductor behaves conventionally. The equation can be a function of temperature, but it does not support variables of node voltage, branch current, or time. If these variables exist in the equation with CONVOLUTION on, the calculation considers only their values at the operating point.
CONVOLUTION	Indicates which method to use: <ul style="list-style-type: none"> ▪ 0 (default): Acts the same as the conventional method. ▪ 1 : Applies recursive convolution, and if the rational function is not accurate enough, it switches to linear convolution. ▪ 2 : Applies linear convolution.
FBASE	Specifies the lower bound of the transient analysis frequency. <ul style="list-style-type: none"> ▪ For CONVOLUTION=1 mode, HSPICE starts sampling at this frequency. ▪ For CONVOLUTION=2 mode, HSPICE uses this value as the base frequency point for Inverse Fourier Transformation. ▪ For recursive convolution, the default value is 0Hz. ▪ For linear convolution, HSPICE uses the reciprocal of the transient period.
FMAX	Specifies the possible maximum frequency of interest. The default value is the frequency point where the function reaches close enough to infinity value. This assumes that the frequency reaches 10THz and that the monotonous function is approaching the infinity value.

Example

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

AC Choke Inductors

Syntax

```
Lxxx node1 node2 [L=] INFINITY [IC=val]
```

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

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

Reluctors

Syntax

Reluctance Inline Form

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

Reluctance External File Form

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

Parameter	Description
Lxxx	Name of a reluctor. Must begin with L, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
n1p n1n ... nNp nNn	Names of the connecting terminal nodes. The number of terminals must be even. Each pair of ports represents the location of an inductor.
RELUCTANCE	Keyword to specify reluctance (inverse inductance).
r1, c1, val1, r2, c2, val2, ... rm, cm, valm	Reluctance matrix data. In general, K is sparse and only non-zero values appear in the matrix. A triplet (r,c,val) represents each matrix entry. The value r and c are integers referring to a pair of inductors from the list of terminal nodes. If there are 2*N terminal nodes, there are N inductors, and the r and c values must be in the range [1,N]. The val value is a reluctance value for the (r,c) matrix location, and the unit for reluctance is the inverse Henry (H ⁻¹). The reluctance_matrix specifies only terms along and above the diagonal. The simulator fills in the lower triangle to ensure symmetry. If you specify lower diagonal terms, the simulator converts that entry to the appropriate upper diagonal term. If you supply multiple entries for the same (r,c) location, it only uses the first one, and ignores the others. A warning advises of the ignored entries. You must assign a positive value all diagonal entries of the reluctance matrix. The reluctance matrix should be positive definite.
FILE="filename1"	For the external file format, the data files should contain three columns of data. Each row should contain an (r,c,val) triplet separated by white space. The r, c, and val values may be expressions surrounded by single quotes. You can specify multiple files to spread the reluctance data over several files, if necessary.
SHORTALL	<ul style="list-style-type: none"> ▪ SHORTALL=yes: Converts all inductors in this model to short circuits, and ignores all reluctance matrix values. ▪ SHORTALL=no (default): Does not convert inductors to short circuits, and does not ignore reluctance matrix values.

Parameter	Description
IGNORE_COUPLING	<ul style="list-style-type: none"> ▪ IGNORE_COUPLING=yes: Ignores all off-diagonal terms (that is, re-sets them to zero). ▪ IGNORE_COUPLING=no (default): Does not do off-diagonal terms.

Example

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

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

Alternatively, you can specify the same element by using:

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

Where reluctance.dat contains:

Chapter 9: Elements

Passive Elements

```

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

```

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

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

Mutual Inductors

General form:

`Kxxx Lyyy Lzzz [K=coupling | coupling]`

Mutual core form:

`Kaaa Lbbb [Lccc ... Lddd] mname [MAG=magnetization]`

Parameter	Description
Kxxx	Mutual inductor element name. Must begin with K, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
Lyyy	Name of the first of two coupled inductors.
Lzzz	Name of the second of two coupled inductors.
K= <i>coupling</i>	Coefficient of mutual coupling. K is a unitless number, with magnitude > 0. If K is negative, the direction of coupling reverses. This is equivalent to reversing the polarity of either of the coupled inductors. Use the K= <i>coupling</i> syntax when using a parameter value or an equation, and the keyword "k=" can be omitted.

Parameter	Description
Kaaa	Saturable core element name. Must begin with K, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
Lbbb, Lccc, Lddd	Names of the windings about the Kaaa core. Requires one winding element, and each winding element must use the magnetic winding syntax. Write all winding elements with the same magnetic core model in one mutual inductor statement in the netlist.
mname	Saturable core model name. (See the Passive Device Models chapter in the <i>HSPICE Elements and Device Models Manual</i> for more information.)
MAG= magnetization	Initial magnetization of the saturable core. You can set this to +1, 0, or -1, where +/- 1 refer to positive and negative values of the BS model parameter. (See the Passive Device Models chapter in the <i>HSPICE Elements and Device Models Manual</i> for more information.)

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

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

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

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

For a listing of output templates for mutual inductor (K-element) models, see [Table 35 on page 439](#).

Example 1

This example couples the Lin and Lout inductors, with a coefficient of 0.9.

```
K1 Lin Lout 0.9
```

Example 2

This example couples the Lhigh and Llow inductors, with a coefficient equal to the value of the COUPLE parameter.

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

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

Example 3

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

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

Ideal Transformer

```
Kxxx Li Lj [k=IDEAL | IDEAL]
```

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

[Equation 3](#) presents multiple coupled inductors. The `IDEAL` keyword activates the equation set for non-DC values. I_j is the current into the first terminal of L_j .

Equation 3

$$\frac{v1}{\sqrt{L1}} = \frac{v2}{\sqrt{L2}} = \frac{v3}{\sqrt{L3}} = \frac{v4}{\sqrt{L4}} = \dots$$

Equation 4

$$0 = (i1 \cdot \sqrt{L1}) + (i2 \cdot \sqrt{L2}) + (i3 \cdot \sqrt{L3}) + (i4 \cdot \sqrt{L4}) + \dots$$

HSPICE can solve any I or V in terms of L ratios and treats DC as expected—while it treats inductors as short circuits. The simulation ignores mutual coupling for DC.

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

$$v2 = v1 \\ i2 + i1 = 0$$

Example 1

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

```

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

```

Example 2

This example is a 2-pin ideal 4:1 step-up balun transformer subcircuit with shared DC path (no DC isolation). Input and output have a common pin, and both inductors have the same value. Note that $R_{load}=4 \cdot R_{in}$.

```

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

```

Example 3

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

Chapter 9: Elements

Passive Elements

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

For a description of the S-parameter (SP) model syntax, see the [S-parameter Modeling Using the S-element](#) chapter in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

Ideal Transformer Format in HSPICE RF

The ideal transformer format simplifies modeling of baluns. They previously used mutual inductors (K elements) with the `IDEAL` keyword. A given balun model required Multiple L and K elements. The ideal transformer model allows modeling of a balun using a single L element.

In the ideal transformer format, you do not specify no absolute inductance or reluctance values. Instead, you specify the transformer's coupling characteristics using inductor number-of-turns values. The behavior of the ideal transformer depends on ratios of the inductors' number of turns.

Syntax

```
Lxxx n1p n1n ... nNp nNn TRANSFORMER_NT=(nt1, ... , ntN)
```

Parameter	Description
<i>Lxxx</i>	Inductor element name. Must begin with L, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
<i>n1p n1n ... nNp nNn</i>	Positive and negative terminal node names. The number of terminals must be even. Each pair of reports represents the location of an inductor.

Parameter	Description
TRANSFORMER_NT	Number of turns values. These parameters must match the number of inductors.

The ideal transformer element obeys the standard ideal transformer equations:

$$\frac{v_1}{nt_1} = \frac{v_2}{nt_2} = \dots = \frac{v_N}{nt_N}$$

$$i_1nt_1 + i_2nt_2 + \dots + i_Nnt_N = 0$$

Example

```
L1 1 0 0 2 3 0 transformer_nt=(1,2,2)
```

Multi-Terminal Linear Elements

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

Examples of transmission lines include:

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

The following sections discuss:

- [S-element \(Scattering Parameter Data\)](#)
- [W-element \(Distributed Transmission Lines\)](#)
- [U-element \(Lumped Transmission Lines\)](#)

S-element (Scattering Parameter Data)

All HSPICE and HSPICE RF analyses can use the S-element. For more information about S-parameters, see [S-parameter Modeling Using the S-element](#) in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

Frequency-Dependent Multi-Terminal (S-element)

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

The S-element describes a linear time-invariant system, and provides a series of data that describe the frequency response of the system. The S-element is particularly useful for high-frequency characterization of distributed passive structures. A common use of the S-element is in microwave circuits such as spiral inductors, because electronic devices in this frequency domain no longer act as they do in low frequencies. In this case, you must consider distributed system parameters. See the example below for an application of the state space stamping to generate a frequency invariant modified nodal analysis (NMA) matrix from frequency-dependent characteristics with the Shooting Newton (`.SN`) algorithm.

For scattering parameter element and model syntax, see [S-element Syntax](#) and [S Model Syntax](#) in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

Example

The following netlist and data file (`test.rfm`) show how the S-element “S1” uses the “STAMP=YSTS” configuration which invokes the state space stamping to generate a frequency invariant modified nodal analysis (NMA) matrix from frequency-dependent characteristics. This stamping method allows the Shooting-Newton algorithm (`.SN`) to obtain the steady state. Note that unless there is RFM file input, the S-element first applies the rational function approximation (equivalent behavior to `RATIONAL_FUNCTION=1`) to the original S-parameters to generate the state space stamping.


```
=====  
main netlist =====  
*** .SN with s-element example  
P1 n1 gnd port=1 dc=1v ac=1v pulse(1 0 1n 1n 1n 10n 20n)  
P2 n2 gnd port=2 dc=1v ac=1v pulse(1 0 1n 1n 1n 10n 20n)  
  
S1 n1 n2 0 mname=s_model  
.model s_model S n=2  
+ rfmfile='test.rfm'  
+ STAMP=YSTS  
.SN tone=0.05Ghz nharms=32  
.option post accurate  
.end
```

The following is from the `.lis` file for this netlist.

Chapter 9: Elements

Multi-Terminal Linear Elements

```
==== rational function matrix file (test.rfm) =====
VERSION 200600 NPORT 2 MATRIX_TYPE Y SYMMETRIC PRECFAC 0.75 Z0 50 50
BEGIN 1 1

BEGIN_REAL 9
DC 2.10290261e-02
2.80562648113e+07  1.791888661818e+00
1.36806220992e+08 -5.313505935943e+01
1.16867967247e+09  2.840375731037e+06
1.23552099406e+09 -4.257158329976e+06
1.92568095149e+09  3.038955064913e+06
4.15005808751e+09 -8.058749095413e+06
1.00149288271e+10  3.846931398394e+06
2.27536895845e+10  1.702938150800e+05
3.54118199282e+10 -1.243885701867e+07

BEGIN_COMPLEX 5
5.53251427579e+05  1.28282249537e+06 -3.17377193705e-03 -
1.20935639131e-03
2.39642428296e+09  1.39710928734e+08 -1.99538130185e+07 -
6.93072640638e+07
2.41275272760e+09  4.88535891322e+09  2.92904966609e+04
4.08311621367e+04
9.49575839142e+08 -2.82753080087e+10 -1.69178467311e+05 -
1.42790736653e+04
    3.74702282735e+10  2.26461714292e+1
6.18960971035e+06  2.73309486084e+05 END BEGIN 2 2 DC 2.10290261e-
02

BEGIN_REAL 9
2.80562648113e+07  1.79188866181e+00
1.36806220992e+08 -5.31350593594e+01
1.16867967247e+09  2.84037573103e+06
1.23552099406e+09 -4.25715832997e+06
1.92568095149e+09  3.03895506491e+06
4.15005808751e+09 -8.05874909541e+06
1.00149288271e+10  3.84693139839e+06
2.27536895845e+10  1.70293815080e+05
3.54118199282e+10 -1.24388570186e+07

BEGIN_COMPLEX 5
5.53251427579e+05  1.28282249537e+06 -3.17377193705e-03 -
1.20935639131e-03
2.39642428296e+09  1.39710928734e+08 -1.99538130185e+07 -
6.93072640638e+07
2.41275272760e+09  4.88535891322e+09  2.92904966609e+04
4.08311621367e+04
9.49575839142e+08 -2.82753080087e+10 -1.69178467311e+05 -
1.42790736653e+04  3.74702282735e+10  2.26461714292e+10
```

```
6.18960971035e+06 2.73309486084e+05  
END
```

W-element (Distributed Transmission Lines)

The W-element supports 5 formats to specify the transmission line properties:

- Model 1: RLGC-Model specification
 - Internally specified in a .model statement
 - Externally specified in a different file
- Model 2: U-Model specification
 - RLGC input for up to five coupled conductors
 - Geometric input (planer, coax, twin-lead)
 - Measured-parameter input
 - Skin effect
- Model 3: Built-in field solver model
 - Standard format (using geometric data with the W-element)
 - Tabular format
- Model 4: Frequency-dependent tabular model
- Model 5: S-parameter Model

W-element Statement

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

RLGC input form:

```
Wxxx in1 [in2 [...inx]] refin out1 [out2 [...outx]]  
+ refout [RLGCfile=filename | RLGCMODEL=name] N=val L=val
```

U Model form:

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

Field solver form:

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

Chapter 9: Elements

Multi-Terminal Linear Elements

Table Model form:

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

S Model form:

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

Parameter	Description
Wxxx	Lossy (W-element) transmission-line element name. Must start with W, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
inx	Signal input node for x th transmission line (Requires in1).
refin	Ground reference for input signal
outx	Signal output node for the x th transmission line (each input port must have a corresponding output port).
refout	Ground reference for output signal.
N	Number of conductors (excluding the reference conductor).
L	Physical length of the transmission line, in units of meters.
RLGCfile=filename	File name reference for the file containing the RLGC information for the transmission lines (for syntax, see Using the W-element in the <i>HSPICE User Guide: Signal Integrity Modeling and Analysis</i>).
RGLCMODEL=modelname	RLGC model file
Umodel=modelname	U-model lossy transmission-line model reference name. A lossy transmission line model, used to represent the characteristics of the W-element transmission line.
FSmodel=modelname	Internal field solver model name. References the PETL internal field solver as the source of the transmission-line characteristics (for syntax, see Using the Field Solver to Extract Tx Line Parameters in the <i>HSPICE User Guide: Signal Integrity Modeling and Analysis</i>).
Smodel=modelname	S Model name reference, which contains the S-parameters of the transmission lines (for the S Model syntax, see the HSPICE User Guide: Signal Integrity Modeling and Analysis).
TABLEMODEL=modelname	Name of the frequency-dependent tabular model.

Parameter	Description
NODEMAP	<p>String that assigns each index of the S parameter matrix to one of the W-element terminals. This string must be an array of pairs that consists of a letter and a number, (for example, Xn), where</p> <ul style="list-style-type: none"> ▪ X= l, i, N, or n to indicate near end (input side) terminal of the W-element ▪ X= O, o, F, or f to indicate far end (output side) terminal of the W-element. <p style="text-align: center;">The default value for NODEMAP is "l1I2I3...lnO1O2O3...On"</p>

The number of ports on a single transmission line is unlimited. You must provide one input and output port, the ground references, a model or file reference, a number of conductors, and a length.

Example 1

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

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

Where,

- Both signal references are `gnd`
- The RLGC file name: `cable.rlgc`
- The transmission line is 5 meters long.

Example 2

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

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

Where,

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

Example 3

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

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

Where,

Chapter 9: Elements

Multi-Terminal Linear Elements

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

Example 4: S Model Example

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

Where,

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

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

You can specify only one of the RLGCFfile, FSmodel, Umodel, or Smodel models, in a single W-element card.

[Figure 21 on page 201](#) shows node numbers for the element syntax.

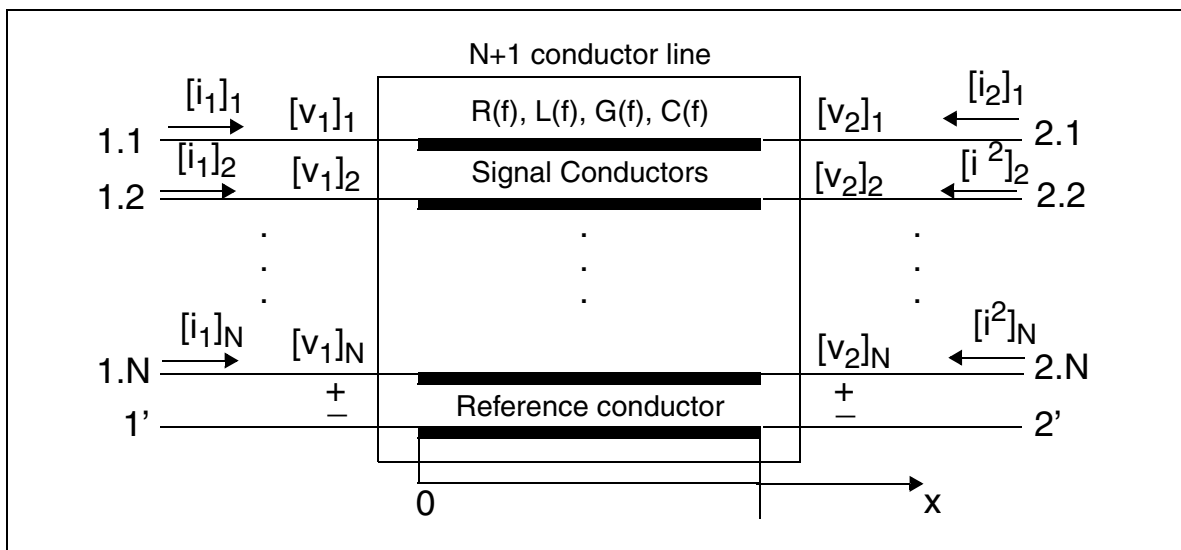


Figure 21 Terminal Node Numbering for the W-element

For additional information about the W-element, see [W- and T-elements Modeling Lossless and Lossy Transmission Lines](#) in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

U-element (Lumped Transmission Lines)

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

Parameter	Description
Uxxx	Lossy (U-element) transmission line-element name. Must begin with U, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
inx	Requires signal input node for the x^{th} transmission-line ($in1$).
refin	Ground reference for the input signal.
outx	Signal output node for the x^{th} transmission line (each input port must have a corresponding output port).
refout	Ground reference for the output signal.

Parameter	Description
mname	Model reference name for the U-model lossy transmission-line.
L	Physical length of the transmission line, in units of meters.
LUMPS	Number of lumped-parameter sections used to simulate the element.

In this syntax, there is a limit to the number of ports on a single transmission line — five in and five out. These are the required parameters: one input and output port, the ground references, a model reference, and a length.

Example 1

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

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

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

Example 2

The U`cable` transmission line connects the in1 and in2 input nodes to the out1 and out2 output nodes:

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

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

Example 3

The U`net1` element is a five-conductor lossy transmission line:

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

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

Using the Scattering Parameter Element in HSPICE

The S- (scattering) element gives you a convenient way to describe a multi-terminal network. You can use the S-element in conjunction with the generic frequency-domain model (.MODEL SP), or data files that describe frequency-varying behavior of a network, and provide discrete frequency-dependent data such as a Touchstone file and a Common Instrumentation Transfer and Interchange (CITI) file. See the [HSPICE User Guide: Simulation and Analysis](#) for more information.

In particular, the S-parameter in the S-element represents the generalized scattering parameter (S) for a multi-terminal network.

The S-parameter and the Y-parameter satisfy the following relationship:

Equation 5

$$Y = Y_{rs}(I - S)(I + S)^{-1}Y_{rs}$$

where Y_r is the characteristic admittance matrix of the reference system. The following formula relates Y_r to the Z_r characteristic impedance matrix:

Equation 6

$$Y_r = Z_r^{-1}Y_{rs}Y_{rs} = Y_r Z_{rs} Z_{rs} = Z_r$$

Similarly, you can convert the Y-parameter to the S-parameter as follows:

Equation 7

$$S = (I + Z_{rs}YZ_{rs})^{(-1)}(I - Z_{rs}YZ_{rs})$$

The follow sections discuss these topics:

- [S-element \(Generic Multiport\)](#)
- [S-element Syntax](#)
- [Frequency-Dependent Multi-Terminal \(S-element\) in HSPICE RF](#)

S-element (Generic Multiport)

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

Chapter 9: Elements

Using the Scattering Parameter Element in HSPICE

- S (scattering)
- Y (admittance)

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

- DC
- AC
- Transient
- Small Signal

For a description of the S-parameter and SP model analysis, see the [S-parameter Modeling Using the S-element](#) chapter in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

S-element Syntax

Use the following S-element syntax to show the connections within a circuit:

```
Sxxx nd1 nd2 ... ndN ndRef
+ [ENFORCE_PASSIVE=0|1]
+ [MNAME=Smodel_name] [FQMODEL=sp_model_name]
+ [TYPE=[s|y]] [Z0=[value | vector_value]]
+ [FBASE = base_frequency] [FMAX=maximum_frequency]
+ [PRECFACT=val] [DELAYHANDLE=[1|0|ON|OFF]]
+ [DELAYFREQ=val]
+ [INTERPOLATION=STEP|LINEAR|SPLINE|HYBRID]
+ [INTDATATYP=[RI|MA|DBA]] [HIGHPASS=[1|2|3|4]]
+ [LOWPASS=[0|1|2|3]] [MIXEDMODE=[0|1]]
+ [DATATYPE=data_string]
+ [NOISE=[1|0]] [NoiPassiveChk=1|0] [DTEMP=val]
+ [PASSIVE=[0|1]]
+ [RATIONAL_FUNC=[0|1]] [RATIONAL_FUNC_REUSE=[0|1]]
+ [STAMP=[S|Y|YSTS|SSTS]] [M=int]
```

Parameter	Description
nd1 nd2...ndN	<p>Nodes of an S-element (see Figure 22 on page 209) and Node Example. Three kinds of definitions are present:</p> <ul style="list-style-type: none"> ▪ With no reference node ndRef, the default reference node is GND. Each node ndi (i=1~N) and GND construct one of the N ports of the S-element. ▪ Defines ndRef with one reference node. Each node ndi (i=1~N) and the ndRef construct one of the N ports of the S-element. ▪ With an N reference node, each port has its own reference node. You can write the node definition in a clearer way as: nd1+ nd1- nd2+ nd2- ... ndN+ ndN- Each pair of the nodes (ndi+ and ndi-, i=1~N) constructs one of the N ports of the S-element.
ndRef	Reference node
ENFORCE_PASSIVE	With the ENFORCE_PASSIVE=1 keyword, the S-element statement checks passivity of all the given frequency sampling points. Once HSPICE finds passivity violations, the S-element seeks a minimum amount of loss property which restores passivity of all the violated points then adds the loss to all the given frequency points.
MNAME	Name of the S model; (Supports string parameters in calling an MNAME.)
FQMODEL	Frequency behavior of the parameters. .MODEL statement of sp type, which defines the frequency-dependent matrices array
TYPE	<p>Parameter type:</p> <ul style="list-style-type: none"> ▪ S: (scattering) (default) ▪ Y: (admittance)
Z0 (or Zo)	Characteristic impedance value for the reference line (frequency-independent). For multiple terminals (N>1), HSPICE assumes that the characteristic impedance matrix of the reference lines is diagonal, and that you set diagonal values to Z0. Default=50 Ω .
FBASE	<p>Base frequency to use for transient analysis. This value becomes the base frequency point for Inverse Fast Fourier Transformation (IFFT).</p> <ul style="list-style-type: none"> ▪ If you do not set this value, the base frequency is a reciprocal value of the transient period. ▪ If you set a frequency that is smaller than the reciprocal value of the transient, then transient analysis performs circular convolution, and uses the reciprocal value of FBASE as its base period.
FMAX	Maximum frequency use in transient analysis. Used as the maximum frequency point for Inverse Fast Fourier Transformation (IFFT).

Chapter 9: Elements

Using the Scattering Parameter Element in HSPICE

Parameter	Description
PRECFAF	In almost all cases, you do not need to specify a value for this parameter. This parameter specifies the precondition factor keyword used for the precondition process of the S-parameter. A precondition avoids an infinite admittance matrix. The default is 0.75, which is good for most cases.
DELAYHANDLE	DELAYHANDLE in S-element simulation extracts a system delay before constructing the system impulse response. This may help to improve transient accuracy when the system does have delay, such as transmission line system. Recommendation: Because S-parameters represent a system which has delay, turn DELAYHANDLE on. When DELAYHANDLE is ON (or 1) the S-element extracts propagation delay to simplify transfer functions, then proceeds to approximation. In the time domain, HSPICE handles the extracted delay separately.
DELAYFREQ	Delay frequency for transmission-line type parameters. The default is FMAX. If the DELAYHANDLE is OFF, but DELAYFREQ is nonzero, HSPICE still simulates the S-element in delay mode.
INTERPOLATION	The interpolation method: <ul style="list-style-type: none">▪ STEP: piecewise step▪ SPLINE: b-spline curve fit▪ LINEAR: piecewise linear (default)▪ HYBRID: HSPICE combines different interpolation/extrapolation methods, and switches automatically between them to get the best accuracy. If needed, it also does causality correction down to DC. It is most useful for the S-parameters showing local resonances, and provides the proper interpolation and low-frequency extrapolation method for each entry of the S matrix, which shows different behaviors. For best accuracy, provide low frequency examples.
INTDATTYP	Data type for the linear interpolation of the complex data. <ul style="list-style-type: none">▪ RI: real-imaginary based interpolation▪ DBA: dB-angle based interpolation▪ MA: magnitude-angle based interpolation (default)
HIGHPASS	Method to extrapolate higher frequency points. <ul style="list-style-type: none">▪ 0: cut off▪ 1: use highest frequency point▪ 2: perform linear extrapolation with the highest 2 points▪ 3: apply the window function to gradually approach the cut-off level (default)▪ 4: Estimates average derivatives of the phase and magnitude from highest 10% of sampling points. Extrapolation uses the highest sampling point and these derivatives.

Parameter	Description
LOWPASS	<p>Method to extrapolate lower frequency points.</p> <ul style="list-style-type: none"> ▪ 0: Cut off. ▪ 1: Make use of the S matrix at the magnitude of the lowest given frequency point; Set the magnitude value of each entry as the element of DC matrix. The real part of the extrapolated value at DC point determines the sign of each value (default). ▪ 2: Perform linear extrapolation with the magnitude of the lowest two points. ▪ 3: Perform rational function approximation based on low end frequency extrapolation.
MIXEDMODE	Set to =1 if the parameters are in the mixed mode.
DATATYPE	<p>A string used to determine the order of the indices of the mixed-signal incident or reflected vector. The string must be an array of a letter and a number (Xn) where:</p> <ul style="list-style-type: none"> ▪ X = D to indicate a differential term = C to indicate a common term = S to indicate a single (grounded) term ▪ n = the port number
NOISE	<p>Activates thermal noise.</p> <ul style="list-style-type: none"> ▪ 1 (default): Element generates thermal noise ▪ 0: Considers element noiseless
NoiPassiveChk	<p>Checks S-parameter for passivity in noise analysis (only).</p> <ul style="list-style-type: none"> ▪ 1 (default): Checks for passivity; if it fails at any frequency, thermal noise is <i>off</i> for the specific frequency point. ▪ 0: Disables the passivity checker; thermal noise is always <i>on</i>.
DTEMP	<p>Temperature difference between the element and the circuit, expressed in $\times C$. The default is 0.0.</p> <p>Calculates element temperature as:</p> $T = \text{Element temperature } (\times K) \\ = 273.15 (\times K) + \text{circuit temperature } (\times C) \\ + \text{DTEMP } (\times C)$ <p>Where you specify circuit temperature by using either the <code>.TEMP</code> statement, or by sweeping the global <code>TEMP</code> variable in <code>.DC</code>, <code>.AC</code>, or <code>.TRAN</code> statements.</p> <p>When a you do not use a <code>.TEMP</code> statement or <code>TEMP</code> variable, <code>.OPTION TNOM</code>, sets the circuit temperature, which defaults to 25 $\times C$ unless you use <code>.OPTION SPICE</code>, which raises the default to 27 $\times C$.</p>

Chapter 9: Elements

Using the Scattering Parameter Element in HSPICE

Parameter	Description
PASSIVE	<p>Activates passive checker to help debug passive models. The default is 0 for the S-element where 0=deactivate and 1=activate. The default tolerance value is $TOL=1e-2$. The eigenvalue vector of matrix $(I-S*S')$ is "ev". Each of the elements of the eigenvalue vector is $ev[i]$.</p> <p>If $RE(ev[i]) < -(TOL*0.1)$, HSPICE issues this warning message: **warning** [model_name] passivity warning, real part of eigenvalue of $(I-S*S')$ is smaller than $< -1e-3$ at $F=xxxx$. Simulation results may not be accurate.</p> <p>If $RE(ev[i]) < -(TOL)$, HSPICE issues the following error message:</p> <p>**error** [model_name] passivity violation, real part of eigenvalue of $(I-S*S')$ is smaller than $< -1e-2$ at $F=xxxx$.</p>
RATIONAL_FUNC	<ul style="list-style-type: none">0: (default) Performs the same as the conventional S-element. (Performs FBASE/FMAX-based linear convolution.)1: Performs rational function approximation then recursive convolution; also handles non-causal S-parameters
RATIONAL_FUNC_REUSE	<p>The S-element rational function approximation process stores the fitting data into a binary file named MODEL_NAME.yrf (DEHAYHANDLE=0) or MODEL_NAME.yrfd (DELAYHANDLE=1). The S-element seeks these files and reuse when available, if RATIONAL_FUNC_REUSE=1 (default). Reusing rational function data increases efficiency especially for large systems.</p> <ul style="list-style-type: none">0: Discard previously extracted rational function data and re-run the rational function approximation.1: (default) Reuse rational function data if available.
STAMP	<ul style="list-style-type: none">Y: Conventional admittance based stampS: Scattering parameter based stamp (Note 1)YSST: Admittance parameter based state space stamp (Note 2)SSST: Scattering parameter based state space stamp (Note 2) <p>Note 1: Although Y and S stamp types behave mathematically equivalent, if you select the S type, the S-element activates a procedure to reduce memory consumption by taking matrices' sparseness into account.</p> <p>Note 2: You can activate YSTS and SSTS stamp methods when you specify RATIONAL_FUNC=1. The state space stamping embeds all the state variables for extracted rational function matrix into the modified nodal analysis (NMA) matrix instead of performing recursive convolution integration. Although this stamping method may incur additional computational cost, since it produces frequency invariant NMA matrix, it enables time domain steady state (called .SN in HSPICE RF) analysis to handle frequency-dependent S-parameter blocks.</p>
M	<p>S-element multiplier; replicates element <i>int</i> times, in parallel; default is 1. Do not assign a negative value or zero as the M value.</p>

The nodes of the S-element must come first. If you do not declare the MNAME, you must specify the FQMODEL. You can specify all the optional parameters in both the S-element and S model statements, except for the MNAME argument.

You can enter the optional arguments in any order, and the parameters specified in the element statement have a higher priority.

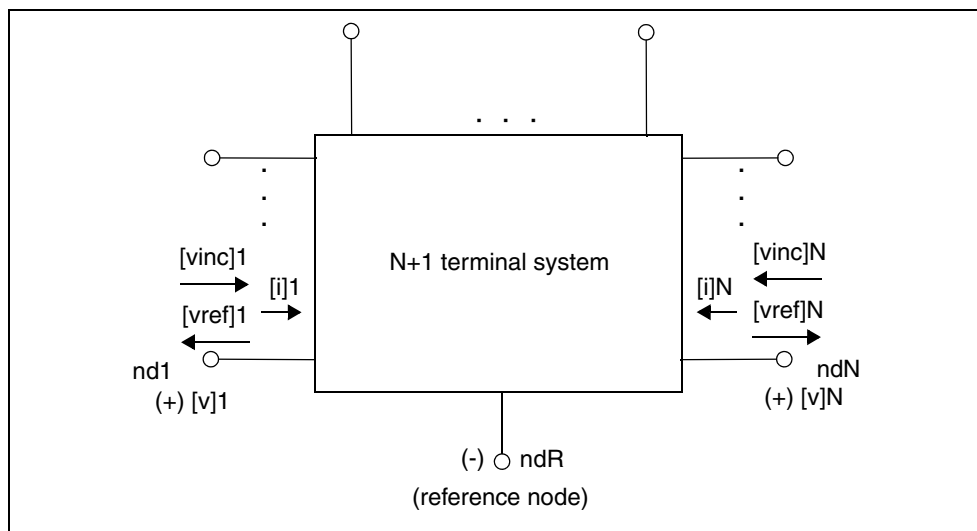


Figure 22 Terminal Node Notation

Node Example

The following example illustrates the $nd1$ $nd2$. . . ndN —*no reference*, *single reference*, and *multi-reference* parameters.

Chapter 9: Elements

Using the Scattering Parameter Element in HSPICE

```
**S-parameter example

.opt post
.ac lin 500 1Hz 30MegHz
.tran 0.1ns 10ns

V1 n1 0 ac=1v PULSE 0v 5v 5n 0.5n 0.5n 25n

* no reference
S_no_ref n1 n2 mname=s_model

* single reference
S_one_ref n1 n3 gnd mname=s_model

*multi-reference
S_multi_ref n1 gnd n4 gnd mname=s_model
Rt1 n2 0 50
Rt2 n3 0 50
Rt3 n4 0 50

* 50 ohm resistor
.MODEL s_model S
+ N=2 FQMODEL=SFQMODEL TYPE=S Z0=50 50
.MODEL SFQMODEL SP N=2 SPACING=POI INTERPOLATION=LINEAR
+ MATRIX=NONSYMMETRIC
+ DATA=1
+ 1.0 0.333333333 0.0 0.666666667 0.0 0.666666667 0.0
0.333333333 0.0

.end
```

The S-element must have a call to one of the supported S-parameter file formats (Touchstone, Citi or .SC#). HSPICE gets the number of ports from the S-parameter file. You can also explicitly specify $N=n$ where 'n' is the number of ports.

- For n terminals, the S-element assumes no reference node.
- For n+1 terminals, the S-element assumes one reference node.
- For 2n terminals, the S-element assumes signal nodes and n reference nodes. Each pair of nodes is a signal and a reference node.

Frequency-Dependent Multi-Terminal (S-element) in HSPICE RF

All HSPICE and HSPICE RF analyses can use the S-element. When used with the generic frequency-domain model (`.MODEL SP`), an S-element is a convenient way to describe the behavior of a multi-terminal network.

The S-element describes a linear time-invariant system, and provides a series of data that describe the frequency response of the system. The S-element is particularly useful for high-frequency characterization of distributed passive structures. A common use of the S-element is in microwave circuits such as spiral inductors, because electronic devices in this frequency domain no longer act as they do in low frequencies. In this case, you must consider distributed system parameters. See the example below for an application of the state space stamping to generate a frequency invariant modified nodal analysis (NMA) matrix from frequency-dependent characteristics by using the Shooting Newton (`.SN`) algorithm.

For scattering parameter element/model, see [S-element Syntax](#) and [S Model Syntax](#) in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

Example

The following netlist and data file (`test.rfm`) show how the S-element “S1” uses the “STAMP=YSTS” configuration which invokes the state space stamping to generate a frequency invariant modified nodal analysis (NMA) matrix from frequency-dependent characteristics. This stamping method allows the Shooting-Newton algorithm (`.SN`) to obtain the steady state. Note that without RFM file input, the S-element first applies the rational function approximation (equivalent behavior to `RATIONAL_FUNCTION=1`) to the original S-parameters to generate the state space stamping.

```
===== main netlist =====  
*** .SN with s-element example  
P1 n1 gnd port=1 dc=1v ac=1v pulse(1 0 1n 1n 1n 10n 20n)  
P2 n2 gnd port=2 dc=1v ac=1v pulse(1 0 1n 1n 1n 10n 20n)  
  
S1 n1 n2 0 mname=s_model  
.model s_model S n=2  
+ rfmfile='test.rfm'  
+ STAMP=YSTS  
.SN tone=0.05Ghz nharms=32  
.option post accurate  
.end
```

The following is an excerpt from the `.lis` file for this netlist.

Chapter 9: Elements

Using the Scattering Parameter Element in HSPICE

```
==== rational function matrix file (test.rfm) =====
VERSION 200600 NPORT 2 MATRIX_TYPE Y SYMMETRIC PRECFAC 0.75 Z0 50 50
BEGIN 1 1
BEGIN_REAL 9
DC 2.10290261e-02
2.80562648113e+07  1.791888661818e+00
1.36806220992e+08 -5.313505935943e+01
1.16867967247e+09  2.840375731037e+06
1.23552099406e+09 -4.257158329976e+06
1.92568095149e+09  3.038955064913e+06
4.15005808751e+09 -8.058749095413e+06
1.00149288271e+10  3.846931398394e+06
2.27536895845e+10  1.702938150800e+05
3.54118199282e+10 -1.243885701867e+07
BEGIN_COMPLEX 5
5.53251427579e+05  1.28282249537e+06 -3.17377193705e-03 -
1.20935639131e-03
2.39642428296e+09  1.39710928734e+08 -1.99538130185e+07 -
6.93072640638e+07
2.41275272760e+09  4.88535891322e+09  2.92904966609e+04
4.08311621367e+04
9.49575839142e+08 -2.82753080087e+10 -1.69178467311e+05 -
1.42790736653e+04
    3.74702282735e+10  2.26461714292e+1
6.18960971035e+06  2.73309486084e+05 END BEGIN 2 2 DC 2.10290261e-
02
BEGIN_REAL 9
2.80562648113e+07  1.79188866181e+00
1.36806220992e+08 -5.31350593594e+01
1.16867967247e+09  2.84037573103e+06
1.23552099406e+09 -4.25715832997e+06
1.92568095149e+09  3.03895506491e+06
4.15005808751e+09 -8.05874909541e+06
1.00149288271e+10  3.84693139839e+06
2.27536895845e+10  1.70293815080e+05
3.54118199282e+10 -1.24388570186e+07
BEGIN_COMPLEX 5
5.53251427579e+05  1.28282249537e+06 -3.17377193705e-03 -
1.20935639131e-03
2.39642428296e+09  1.39710928734e+08 -1.99538130185e+07 -
6.93072640638e+07
2.41275272760e+09  4.88535891322e+09  2.92904966609e+04
4.08311621367e+04
9.49575839142e+08 -2.82753080087e+10 -1.69178467311e+05 -
1.42790736653e+04  3.74702282735e+10  2.26461714292e+10
6.18960971035e+06  2.73309486084e+05
END
```

Port Element

The port element (P-element) identifies the ports used in .LIN analysis and in other all other analyses behaves as either a noiseless impedance or a voltage source in series with the port impedance (DC, AC, or TRAN). Each port element requires a unique port number. Each port has an associated system impedance, Z0. If you do not explicitly specify the system impedance, the default is 50 ohms.

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

The port element accepts transient waveforms AM, EXP, PULSE, PWL, SFFM, SIN, LFSR and, for signal integrity usage, the PAT source.

The mixed-mode port element has an additional reference pin that allows further flexibility in creating, detecting, and separating common-mode and differential-mode signals. It is useful for measuring mixed-mode S-parameters (see [Using the P-element for Mixed-Mode Measurement](#)).

Syntax

```
Pxxx p n port=portnumber
+ [Z0=val]                               $Port Impedance
+ [DC mag] [AC mag phase] [HBAC mag phase] $Voltage/Power Info
+ [HB mag phase harm tone modharm modtone]
+ [transient_waveform] [ENCODE=DW8B10B] [RD_INIT=0|1]
+ [TRANFORHB=[0|1]] [DCOPEN=[0|1]]
+ [power=[0|1|2|W|dbm]]                   $Power Switch
+ [RDC=val] [RAC=val]                     $ Source Impedance Overrides
+ [RHBAC=val] [RHB=val] [RTRAN=val]
+ [Emphasis_Level=val] [Emphasis_Time=val] $ Emphasis
+ [DCD=val] [DCD_TYPE=0|1|2|3]           $ Duty Cycle Distortion
+ [PJ=val] [PJ_TYPE=0|1|2|3]             $ Period Jitter
+ [RJ=val1, [val2, ... valN]] [VN=val] $ Rand Jitter Stateye
+ [AMI_OBJ=filename] [AMI_PARAM=filename] [AMI_CDR=0|1] $AMI
```

Parameter	Description
port=portnumber	The port number. Numbered sequentially beginning with 1 with no shared port numbers.

Chapter 9: Elements
Port Element

Parameter	Description
<code>z0=val</code> (or <code>Zo=val</code>)	Port impedance (Ohms). (Default: 50). Sets port characteristic impedance used for .LIN analysis, sets port termination impedance for other analyses, and also sets source impedance when you use the port element as a signal source.
<code>DC mag</code>	DC voltage or power source value.
<code>AC mag phase</code>	AC voltage or power source value.
<code>HBAC mag phase</code>	(HSPICE RF) HBAC voltage or power source value.
<code>HB mag phase harm tone</code> <code>modharm modtone</code>	(HSPICE RF) HB voltage, current, or power source value. Allows multiple HB specifications with different harm, tone, modharm, and modtone values. <ul style="list-style-type: none"> ▪ phase is in degrees ▪ harm and tone are indices corresponding to the tones specified in the .HB statement. Indexing starts at 1 (corresponding to the first harmonic of a tone). ▪ modtone and modharm specify sources for multi-tone simulation. A source specifies a tone and a harmonic, and up to 1 offset tone and harmonic (modtone for tones and modharm for harmonics). Then it describes the signal as: $V(\text{or } I) = \text{mag} * \cos(2 * \pi * (\text{harm} * \text{tone} + \text{modharm} * \text{modtone}) * t + \text{phase})$
<code>transient_waveform</code>	(Transient analysis) Voltage or power source waveform. Any one of waveforms: AM, EXP, PULSE, PWL, SFFM, SIN, or PRBS. Does <i>not</i> allow multiple transient descriptions.
<code>ENCODE=DW8b10b</code>	Keyword to specify 8b/10b encoding.
<code>RD_INIT=0 1</code>	Initial value of Running Disparity. <i>Running Disparity.1</i> is the name of the one-bit memory that recalls the bias of the last unbalanced code word: Specifies that a Running Disparity value of zero is synonymous with negative Running Disparity (?). <ul style="list-style-type: none"> ▪ 0: Specifies that a Running Disparity value of one is synonymous with negative Running Disparity (-) ▪ 1: Specifies that a Running Disparity value of one is synonymous with positive Running Disparity (+).

Parameter	Description
TRANFORHB=[0 1]	<ul style="list-style-type: none"> ▪ (HSPICE RF) 0 (default): Ignores the transient description if you specify an HB value or a DC value. If you do not provide a DC or HB value and TRANFORHB=0, then HB analysis treats the source as a DC source, and the DC source value is the time=0 value. ▪ 1: HB analysis uses the transient description if its value is VMRF, SIN, PULSE, PWL, or LFSR. If the type is a non-repeating PWL source, the simulation treats the time=infinity value as a DC analysis source value. For example, the simulation treats the the following statement as a DC source with value=1 for HB analysis: <pre>v1 1 0 PWL (0 0 1n 1 1u 1) + TRANFORHB=1</pre> In contrast, the following statement is a 0V DC source: <pre>v1 1 0 PWL (0 0 1n 1 1u 1) + TRANFORHB=0</pre> The simulation treats the following statement as a periodic source with a 1u period that uses PWL values: <pre>v1 1 0 PWL (0 0 1n 1 0.999u 1 1u 0) R + TRANFORHB=1</pre> To override the global TRANFORHB option, explicitly set TRANFORHB for a voltage or current source.
DCOPEN	<p>Switch for open DC connection if you have not set DC mag.</p> <ul style="list-style-type: none"> ▪ 0 (default): P-element behaves as an impedance termination. ▪ 1: P-element behaves as an open circuit in DC operating point analysis. .LIN analysis mainly uses DCOPE=1 so the P-element does not affect the self-biasing device under test by opening the termination at the operating point.
RDC= <i>val</i>	(DC analysis) Series resistance (overrides <i>z0</i>).
RAC= <i>val</i>	(AC analysis) Series resistance (overrides <i>z0</i>).
RHBAC= <i>val</i>	(HSPICE RF HBAC analysis) Series resistance (overrides <i>z0</i>).
RHB= <i>val</i>	(HSPICE RF HB analysis) Series resistance (overrides <i>z0</i>).
RTRAN= <i>val</i>	(Transient analysis) Series resistance (overrides <i>z0</i>).
power=[0 1 2 W dbm]	<p>Power switch. Causes treatment of the signal amplitude quantities specified with the port element to as power levels instead of voltage levels. The port element then becomes a power source, realized as a voltage source in series with the port impedance (Z0). The simulation calculates the voltage source value internally as that is necessary to realize the appropriate RMS available power. For transient analysis, this is only appropriate for DC and SIN sources.</p> <ul style="list-style-type: none"> ▪ When 0 (default), power entry disabled. ▪ When 1 or W, power given in units of Watts. ▪ When 2 or dBm, power given in units of dBm (dB relative to 1 mW).
Emphasis_Level	Aids in .STATEYE analysis pre-emphasis and de-emphasis

Chapter 9: Elements

Port Element

Parameter	Description
Emphasis_Time	Aids in .STATEYE analysis pre-emphasis and de-emphasis
DCD	Aids in .STATEYE analysis; specifies peak percentage of the duty cycle distortion (DCD). Default value is zero.
DCD_TYPE	Aids in .STATEYE analysis; specifies variation type. Default type for non-zero DCD is 1 (constant). <ul style="list-style-type: none"> ▪ 0: no DCD ▪ 1: constant DCD ▪ 2: uncorrelated triangular DCD variation ▪ 3: uncorrelated sinusoidal DCD variation
PJ	Aids in .STATEYE analysis; specifies periodic jitter (voltage) magnitude. Default value is zero.
PJ_TYPE	Aids in .STATEYE analysis; specifies variation type. Default type for non-zero DCD is 1 (constant). <ul style="list-style-type: none"> ▪ 0: no periodic jitter ▪ 1: constant voltage shift ▪ 2: uncorrelated triangular jitter variation ▪ 3: uncorrelated sinusoidal jitter variation
RJ	An array of the real numbers to specify the standard deviation of the Gaussian random jitter. The array must be in the order of the port element index. By default, the simulation does not add random jitter.
VN	(Voltage noise); aids in .STATEYE analysis. Specify the standard deviation of the Gaussian random voltage noise. By default, the simulation does not add random voltage noise.
AMI_OBJ	Aids in .STATEYE analysis; specifies an Algorithmic Modeling Interface (AMI) shared object (typically, lib*.so for UNIX, *.dll for Windows)
AMI_PARAM	Aids in .STATEYE analysis; specifies an AMI parameter file (*.ami)
AMI_CDR	Aids in .STATEYE. Switch to use AMI_GetWave's clock data recovery (CDR) output in eye diagram generation. Default value is 0 (disabled state).

Example

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

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

Computing scattering parameters requires z_0 reference impedance values. The order of the *port* parameters (in the P-element) determines the order of the S, Y, and Z parameters. Unlike the .NET command, the .LIN command does

not require you to insert additional sources into the circuit. To calculate the requested transfer parameters, HSPICE automatically inserts these sources as needed at the port terminals. You can define an unlimited number of ports.

Using the Port Element for Mixed-Mode Measurement

To measure mixed mode S-parameters you can use a port element with three terminals. Except for the number of external terminals, the syntax of the port element remains the same. The LIN analysis function internally sets the necessary drive mode (common/differential) of these mixed mode port elements. For analyses other than the LIN analysis (such as DC, AC, TRAN, and so on), the mixed-mode P-element acts as a differential driver that drives positive nodes with half of their specified voltage and the negative nodes with a negated half of the specified voltage. [Figure 23 on page 217](#) shows the block diagram of the mixed mode port element.

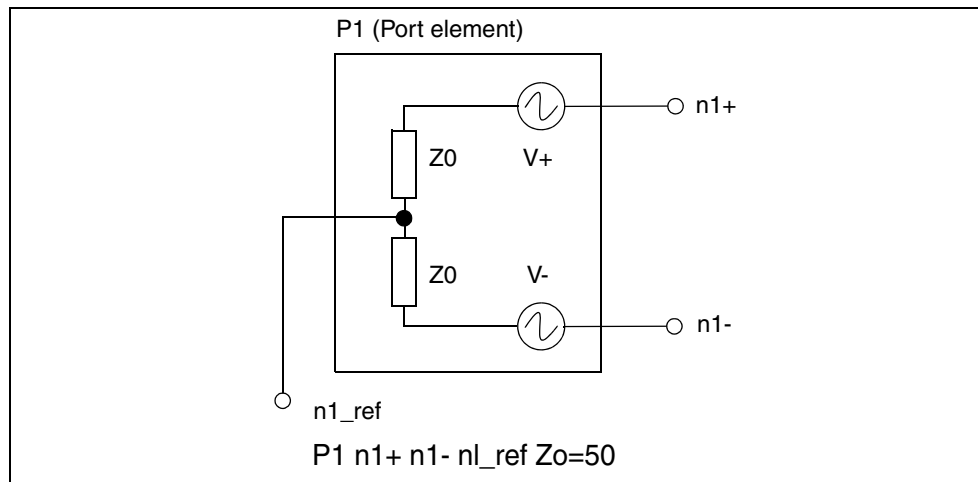


Figure 23 Mixed Mode Port Element

Active Elements

- [Diode Element](#)
- [Bipolar Junction Transistor \(BJT\) Element](#)
- [JFETs and MESFETs](#)

- [MOSFETs](#)
- [Extended MOSFET Element Support Using .OPTION MACMOD](#)

Diode Element

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

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

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

Fowler-Nordheim (LEVEL=2) form:

```
Dxxx nplus nminus mname [W=val] [L=val] [WP=val]
+ [OFF] [IC=vd] [M=val]
```

Parameter	Description
Dxxx	Diode element name. Must begin with D, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
nplus	Positive terminal (anode) node name. The series resistor for the equivalent circuit attaches to this terminal.
nminus	Negative terminal (cathode) node name.
mname	References diode model name.
AREA	Area of the diode (unitless for LEVEL=1 diode, and square meters for LEVEL=3 diode). This affects saturation currents, capacitances, and resistances (diode model parameters are IK, IKR, JS, CJO, and RS). The SCALE option does not affect the area factor for the LEVEL=1 diode. Default=1.0. Overrides AREA from the diode model. If you do not specify the AREA, HSPICE calculates it from the width and length.
PJ	Periphery of junction (unitless for LEVEL=1 diode, and meters for LEVEL=3 diode). Overrides PJ from the diode model. If you do not specify PJ, HSPICE calculates it from the width and length specifications.
WP	Width of polysilicon capacitor, in meters (for LEVEL=3 diode only). Overrides WP in the diode model. Default=0.0.
LP	Length of polysilicon capacitor, in meters (for LEVEL=3 diode only). Overrides LP in the diode model. Default=0.0.

Parameter	Description
WM	Width of metal capacitor, in meters (for LEVEL=3 diode only). Overrides WM in the diode model. Default=0.0.
LM	Length of metal capacitor, in meters (for LEVEL=3 diode only). Overrides LM in the diode model. Default=0.0.
OFF	Sets the initial condition for this element to OFF, in DC analysis. Default=ON.
IC=vd	Initial voltage, across the diode element. Use this value when you specify the UIC option in the .TRAN statement. The .IC statement overrides this value.
M	Multiplier, to simulate multiple diodes in parallel. The M setting affects all currents, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.
W	Width of the diode, in meters (LEVEL=3 diode model only)
L	Length of the diode, in meters (LEVEL=3 diode model only)

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

For a listing of output templates for diode (D-element) models, see [Table 42 on page 441](#).

Example 1

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

```
D1 1 2 diode1
```

Example 2

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

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

Example 3

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

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

Bipolar Junction Transistor (BJT) Element

For a full demonstration file of a BJT element bipolar analog test case, see the path to `senseamp.sp` in [Benchmark Examples](#) in this user guide. See also [BJT and Diode Examples](#).

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

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

Parameter	Description
Qxxx	BJT element name. Must begin with Q, then up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
nc	Name of collector terminal node.
nb	Name of case terminal node.
ne	Name of emitter terminal node.
ns	Name of substrate terminal node, which is optional. You can also use the BULK parameter to set this name in the BJT model.
mname	BJT model name reference.
area, AREA=area	Emitter area multiplying factor, which affects currents, resistances, and capacitances. Default=1.0.
OFF	Sets initial condition for this element to OFF, in DC analysis. Default=ON.
IC=vbeval, vceval, VBE, VCE	Initial internal base-emitter voltage (vbeval) and collector-emitter voltage (vceval). HSPICE uses this value when the .TRAN statement includes UIC. The .IC statement overrides it.
M	Multiplier, to simulate multiple BJTs in parallel. The M setting affects all currents, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.
AREAB	Base area multiplying factor, which affects currents, resistances, and capacitances. Default=AREA.

Parameter	Description
AREAC	Collector area multiplying factor, which affects currents, resistances, and capacitances. Default=AREA.

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

For a listing of output templates for BJT (Q element) models, see [Table 43 on page 442](#).

Example 1

In the Q1 BJT element:

```
Q1 1 2 3 model_1
```

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

Example 2

In the following, Qopamp1 BJT element:

```
Qopamp1 c1 b3 e2 s Mod1stagepnp AREA=1.5 AREAB=2.5  
AREAC=3.0
```

- The collector connects to the c1 node.
- The base connects to the b3 node.
- The emitter connects to the e2 node.
- The substrate connects to the s node.
- Mod1stagepnp references the BJT model.
- The AREA area factor is 1.5.
- The AREAB area factor is 2.5.
- The AREAC area factor is 3.0.

Example 3

In the Qdrive BJT element:

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

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

JFETs and MESFETs

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

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

Parameter	Description
Jxxx	JFET or MESFET element name. Must begin with J, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
nd	Name of drain terminal node
ng	Name of gate terminal node
ns	Name of source terminal node
nb	Name of bulk terminal node, which is optional.
mname	JFET or MESFET model name reference
area, AREA=area	Area multiplying factor that affects the BETA, RD, RS, IS, CGS, and CGD model parameters. Default=1.0, in units of square meters.
W	FET gate width in meters
L	FET gate length in meters
OFF	Sets initial condition to OFF for this element, in DC analysis. Default=ON.
IC=vdsval, vgsval, VDS, VGS	Initial internal drain-source voltage (vdsval) and gate-source voltage (vgsval). Use this argument when the .TRAN statement contains UIC. The .IC statement overrides it.

Parameter	Description
M	Multiplier to simulate multiple JFETs or MESFETs in parallel. The M setting affects all currents, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.

The syntax only requires drain, gate, and source nodes, and model name fields. Node and model names must precede other fields.

For a listing of output templates for JFET (J-element) models, see [Table 44 on page 444](#).

Example 1

In the J1 JFET element:

```
J1 1 2 3 model_1
```

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

Example 2

In the following Jopamp1 JFET element:

```
Jopamp1 d1 g3 s2 b Mod1stage AREA=100u
```

- The drain connects to the d1 node.
- The source connects to the g3 node.
- The gate connects to the s2 node.
- Mod1stage references the JFET model.
- The area is 100 microns.

Example 3

In the Jdrive JFET element:

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

- The drain connects to the driver node.
- The source connects to the in node.
- The gate connects to the output node.

- `model_jfet` references the JFET model.
- The width is 10 microns.
- The length is 10 microns.

MOSFETs

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

Parameter	Description
Mxxx	MOSFET element name. Must begin with M, followed by up to 1023 alphanumeric and special characters as defined in Table 9 on page 70 .
nd	Name of drain terminal node.
ng	Name of gate terminal node.
ns	Name of source terminal node.
nb	Name of bulk terminal node, which is optional.
mname	MOSFET model name reference or subckt name if you set <code>.OPTION MACMOD</code> .
L	MOSFET channel length, in meters. This parameter overrides <code>.OPTION DEFL</code> , with a maximum value of 0.1m. Default=DEFL.
W	MOSFET channel width, in meters. This parameter overrides <code>.OPTION DEFW</code> . Default=DEFW.
AD	Drain diffusion area. Overrides <code>.OPTION DEFAD</code> . Default=DEFAD, if you set the <code>ACM=0</code> model parameter.
AS	Source diffusion area. Overrides <code>.OPTION DEFAS</code> . Default=DEFAS, if you set the <code>ACM=0</code> model parameter.
PD	Perimeter of drain junction, including channel edge. Overrides <code>.OPTION DEFPPD</code> . Default=DEFAD, if you set the <code>ACM=0</code> or <code>1</code> model parameter. Default=0.0, if you set <code>ACM=2</code> or <code>3</code> .

Parameter	Description
PS	Perimeter of source junction, including channel edge. Overrides .OPTION DEFPS. Default=DEFAS, if you set the ACM=0 or 1 model parameter. Default=0.0, if you set ACM=2 or 3.
NRD	NRD (Number of squares of drain diffusion for resistance calculations) overrides .OPTION DEFNRD. For nonCMI models such as BSIM3 etc... Default=DEFNRD, if you set ACM=0 or 1 model parameter. Default=0.0, if you set ACM=2 or 3 For CMI models such as BSIM4 etc... Default=1.0
NRS	NRS (Number of squares of source diffusion for resistance calculations) overrides .OPTION DEFNRS. For nonCMI models such as BSIM3 etc... Default=DEFNRS, if you set ACM=0 or 1 model parameter. Default=0.0, if you set ACM=2 or 3 For CMI models such as BSIM4 etc... Default=1.0
RDC	Additional drain resistance due to contact resistance, in units of ohms. This value overrides the RDC setting in the MOSFET model specification. Default=0.0.
RSC	Additional source resistance due to contact resistance, in units of ohms. This value overrides the RSC setting in the MOSFET model specification. Default=0.0.
OFF	Sets initial condition for this element to OFF, in DC analysis. Default=ON. This command does not work for depletion devices.
IC=vds, vgs, vbs	Initial voltage across external drain and source (vds), gate and source (vgs), and bulk and source terminals (vbs). Use these arguments with .TRAN UIC. .IC statements override these values.
M	Multiplier, to simulate multiple MOSFETs in parallel. Affects all channel widths, diode leakages, capacitances, and resistances. Default=1.
DTEMP	The difference between the element temperature and the circuit temperature, in degrees Celsius. Default=0.0.
GEO	Source/drain sharing selector for a MOSFET model parameter value of ACM=3. Default=0.0.
DELVTO	Zero-bias threshold voltage shift. Default=0.0.

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

For a full listing of output templates for MOSFET models, see [MOSFET Output Templates](#) in the *HSPICE Reference Manual: MOSFET Models*.

Example

In the following `M1` MOSFET element:

```
M1 1 2 3 model_1
```

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

In the following `Mopamp1` MOSFET element:

```
Mopamp1 d1 g3 s2 b Mod1stage L=2u W=10u
```

- The drain connects to the `d1` node.
- The gate connects to the `g3` node.
- The source connects to the `s2` node.
- `Mod1stage` references the MOSFET model.
- The length of the gate is 2 microns.
- The width of the gate is 10 microns.

In the following `Mdrive` MOSFET element:

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

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

Extended MOSFET Element Support Using .OPTION MACMOD

Use option MACMOD to enable HSPICE to access a subckt definition for MOSFETs, diodes, and BJTs, when no model reference exists. MACMOD=1 treats subcircuits as primitive models with an “M” instantiation. .OPTION MACMOD syntax is:

```
.OPTION MACMOD= [1|2|3|0]
```

When `macmod=1`, HSPICE seeks a subckt definition for the M/Q/D*** element if no model reference exists. The desired subckt name must match (case insensitive) the `mname` field in the M/Q/D*** instance statement. In addition, the number of terminals of the subckt must match with the M/Q/D*** element which references it; otherwise HSPICE aborts the simulation based on no definition for the M/Q/D*** element.

The following limitations apply when `macmod=1`:

1. Element template output does not support elements which use subckt definitions.
2. This feature does not support a element if a string parameter defines the `mname`.
3. The number of terminals for a HSPICE element must be within the range of 3-7; any number of terminals that is out of this range causes the simulation to fail.

When `macmod=2`, HSPICE seeks a MOSFET/BJT/Diode model definition when it cannot find matching subckt or Verilog-A definition for an X-element. The targeted MODEL card could be either a HSPICE built-in model or CMI model. If the model card that matched with the X-element reference name is not a type of MOSFET/BJT/Diode model, the simulator errors out with message of reference "not found."

The following limitations apply when `macmod=2`:

1. The feature of “string parameter supported in model name” does not apply to X-elements that are mapped to the model cards; i.e., reference name of the X-element must be constant string characters.
2. Subckt direct port probing command, `isub()` does not support X-elements mapped to the model cards.
3. HSPICE MOSRA analysis does not work on the X-elements, even when they directly map to compact model cards.

Chapter 9: Elements

Active Elements

When `macmod=3`, HSPICE enables both of the above features; HSPICE seeks a `.subckt` definition for an M/Q/D-element if there is no matching model reference; HSPICE seeks a `.model` definition for an X-element if there is no matching `.subckt` or Verilog-A definition. Usage considerations and limitations remain the same for both features, respectively. When `.OPTION TMIFLAG ≥ 1`, `.OPTION MACMOD` automatically equals 3.

The case of no `.OPTION MACMOD` in the input files or `MACMOD=0` disables the above two features; HSPICE ignores the `MACMOD` option if you set any value other than 1 | 2 | 3 | 0.

The `MACMOD` option is a global option; if there are multiple `MACMOD` options in one simulation, HSPICE uses the value of the last `MACMOD` option.

Example 1

```
**
.option MACMOD=1
M1 net1 net2 net3 net4 nch l=0.2u w=0.2u p1=1
.model nch nmos level=49 ...
.subckt nch d g s b w=1 l=1 p1=gp1
.if (p1 > 0)
Mnch d g s b model_1 w=w l=1
.else
Mnch d g s b model_2 w=w l=1
.endif
.ends
```

In Example 1, extended MOSFET is on. However, because the `mname` in a `.MODEL` statement matches the `mname` of the M1 element, element M1 uses model `nch` rather than the `subckt` definition. The simulation ignores the extra instance parameter `p1`.

Example 2

```

**
.option MACMOD
.param gp1=1 gp2=2 gp3=3
M1 net1 net2 net3 net4 nch l=0.2u w=0.2u p1=gp1 p2=gp2 p3=gp3
.subckt nch d g s b w=1 l=1 p1=gp1 p2=gp2 p3=gp3
.if (p1 > 0 && p2==1 && p3 ==1)
Mnch d g s b model_1 w=w l=1
.else if ( p1 == 0 && p2 ==1 && p3 ==1)
Mnch d g s b model_2 w=w l=1
.else
Mnch d g s b model_3 w=w l=1
.endif
.ends
.model model_1 nmos level=49 ...
.model model_2 nmos level=53 ...
.model model_3 nmos level=54 ...

```

In Example 2, extended MOSFET element support is on; since there is no matching `.MODEL` statement, M1 uses subckt definition `nch`; after evaluation, M1 results in a MOSFET element that species MOSFET model `model_3`.

Example 3

```

**
.option MACMOD
.param gp1=1 gp2=2 gp3=3
M1 net1 net2 net3 net4 nch l=0.2u w=0.2u p1=gp1 p2=gp2 p3=gp3
.subckt nch d g s b w=1 l=1 p1=gp1 p2=gp2 p3=gp3
.if (p1 > 0 && p2==1 && p3 ==1)
Mnch d g s b model_1 w=w l=1
.else if ( p1 == 0 && p2 ==1 && p3 ==1)
Mnch d g s b model_2 w=w l=1
.else
Mnch d g s b model_3 w=w l=1
.endif
C1 g 0 1p
.ends

```

Example 3 shows extended MOSFET element support turned on. Instance M1 uses macro model `nch`, which is a subckt definition that consists of one MOSFET device and one capacitor.

MACMOD Option Limitations

- The number of terminals for M*** must be within the range of 3 to 7. A number of terminals outside of that range cause the simulation to fail.
- A string parameter must not define a MOSFET element's `mname`.

Chapter 9: Elements

Active Elements

- The MACMOD option only applies to HSPICE MOSFET elements.
- Element template output does not support MOSFET elements which use subckt definitions.

For example, if Example 3 includes the output command:

```
.PRINT LX8 (M1) LV9 (M1)
```

—then HSPICE ignores the above output command. Because M1 is using a subckt definition, it is no longer a HSPICE primitive MOSFET device.

- The desired subckt name must match the `mname` field in the M*** instance statement. The match of subckt name and the `mname` field is case insensitive.
- The `.MODEL` definition takes precedence over a subckt definition even when `.OPTION MACMOD` is on.

Direct X-Element Mapping to a MOSFET Model Card

HSPICE seeks a MOSFET model definition when it cannot find a matching `subckt` or Verilog-A definition for an X-element. This applies mainly to a custom model, which can only become enabled by the `.option CMIFLAG` or `.option TMIFLAG`; Subckt or Verilog-A module definitions always take preference over the CMI model card.

HSPICE considers the X-elements that are mapped to model cards as MOSFET devices with certain. The following sections describe these usage considerations and limitations. For information about HSPICE CMI and TMI, contact the Synopsys support team.

Considerations

Syntax check rules of MOSFET devices apply to the X-elements directly mapping to MOSFET model cards, such as:

1. The valid instance parameter list of that X-element becomes the valid instance parameter list of the particular MOSFET model in the mapping model card; HSPICE ignores invalid instance parameter of that X-element; default values of the mapping MOSFET model apply to any undefined instance parameter by comparing with the corresponding MOSFET model.
2. The `.PARAM` netlist commands cannot override MOSFET instance parameters which are device native parameters; such instance parameter rules apply to X-elements with direct mapping to MOSFET model cards, i.e., instance parameters of such an X-element become native device parameters, thus netlist `.PARAM` commands cannot be override them.

3. The number of terminals of the X-element must be in the valid range of its mapping MOSFET model; or simulator exits with an error message.

Limitations

1. The feature of “string parameter supported on MOSFET model name” does not apply to X-elements that are mapped to MOSFET model cards. In other words, you must write a reference name of the X-element as constant string characters.
2. The Subckt direct port probing command, `Isub()` does not support X-elements mapped to MOSFET model cards.

IBIS Buffers (HSPICE Only)

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

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

Parameter	Description
bxxx	Buffer name; starts with the letter B followed by up to 1023 alphanumeric and special characters, which Table 9 on page 70 defines.
node_1 node_2 ... node_N	List of I/O buffer external nodes. The number of nodes and their meaning are specific to different buffer types.
file='filename'	Name of the IBIS file.
model='model_name'	Name of the model.
keyword_i=value_i	Assigns a value of value_i to the keyword_i keyword. Specify optional keywords in brackets ([]). For more information about IBIS keywords, see Specifying Required and Optional Common Keywords in the <i>HSPICE User Guide: Signal Integrity Modeling and Analysis</i> .

Example 1

The Input buffer name is B1. The four terminals names are nd_pc, nd_gc, nd_in and nd_out_of_in. The IBIS model named IBIS_IN is in the file named test.ibs. The values specified in IBIS_IN guide HSPICE to connect

Chapter 9: Elements

IBIS Buffers (HSPICE Only)

`nd_pc` and `nd_gc` to the voltage sources. Do *not* manually connect voltage sources to these nodes.

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

Example 2

The output buffer name is B2. The six terminals names are `nd_pu`, `nd_pd`, `nd_out`, `nd_in`, and `nd_pc`, `nd_gc`. Here, `nd_pc`, `nd_gc` are optional terminals. If you do not set these names, the netlist uses the optional terminal names instead. The IBIS model named `IBIS_OUT` in the file named `test.ibs`. The values specified in `IBIS_OUT` guide HSPICE to connect `nd_pc` and `nd_gc` to the voltage sources. Do *not* manually connect voltage sources to these nodes.

```
B2 nd_pu nd_pd nd_out nd_in [nd_pc nd_gc]
+ file = 'test.ibs'
+ model = 'IBIS_OUT'
```

Example 3

The IO buffer name is B3. The eight terminals names are `nd_pu`, `nd_pd`, `nd_out`, `nd_in`, `nd_en`, `nd_out_of_in`, and `nd_pc`, `nd_gc`. Here, `nd_pc`, `nd_gc` are optional terminals. If you do not set these names, the netlist uses the optional terminal names instead. The IBIS model named `IBIS_IO` is in the IBIS file named `test.ibs`. The values specified in `IBIS_IO` guide HSPICE to connect `nd_pc` and `nd_gc` to the voltage sources. Do *not* manually connect voltage sources to these nodes.

```
B3 nd_pu nd_pd nd_out nd_in nd_en nd_out_of_in [nd_pc nd_gc]
+ file = 'test.ibs'
+ model = 'IBIS_IO'
```

For more examples, see the [Modeling Input/Output Buffers Using IBIS Files](#) chapter in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

Sources and Stimuli

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

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

HSPICE ships over a dozen sources examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files and [Sources Examples](#).

The following section discuss these topics:

- [Independent Source Elements](#)
- [Independent Source Functions](#)
- [Voltage and Current Controlled Elements](#)
- [Power Sources](#)
- [Voltage-Dependent Voltage Sources — E-elements](#)
- [Current-Dependent Current Sources — F-elements](#)
- [Voltage-Dependent Current Sources — G-elements](#)
- [Current-Dependent Voltage Sources — H-elements](#)
- [HSPICE RF-Specific Sources and Stimuli](#)
- [Specifying a Digital Vector File and Mixed Mode Stimuli](#)

Independent Source Elements

Use independent source element (current, I-element, or voltage, V-element) statements to specify DC, AC, transient, and mixed independent voltage and current sources. These elements are also The associated analysis sources depend on the analysis you perform. The zero time value of the transient source of a calculated transient operating point overrides the value of the DC source. (For discussion of P-elements, see Chapter 8, [Port Element](#), in this user guide.)

The following sections discuss these topics:

- [Source Element Conventions](#)
- [Independent Source Element Syntax](#)
- [DC Sources](#)
- [AC Sources](#)
- [Transient Sources](#)
- [Mixed Sources](#)

Source Element Conventions

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

You can use parameters as values in independent sources. Do not use any of the following reserved keywords to identify these parameters: AC, ACI, AM, DC, EXP, PAT, PE, PL, PU, PULSE, PWL, R, RD, SFFM, POWER, or SIN

Independent Source Element Syntax

```
Vxxx n+ n- [[DC=] dcval tranfun [AC=acmag acphase]]  
+ POWER=[0 | 1 | 2 | w | dbm]
```

```
Ixxx n+ n- [[DC=] dcvaltranfun [AC=acmag acphase]]  
+ POWER=[0 | 1 | 2 | w | dbm] [M=val]
```


Parameter	Description
Vxxx	Independent voltage-source element name. Must begin with V, followed by up to 1023 alphanumeric and special characters.
Ixxx	Independent current source element name. Must begin with I, followed by up to 1023 alphanumeric and special characters.
n+	Positive node.
n-	Negative node.
DC=dcval	DC source keyword and value in volts. Used for the operating point calculation for all simulations except transient. Transient analysis calculates an additional operating point with the tranfun value at time zero. Default=0.0.
tranfun	Transient source function (one or more of: AM, DC, EXP, PAT, PE, PL, PU, PULSE, PWL, SFFM, SIN). The functions specify the characteristics of a time-varying source. See the individual functions for syntax.
AC	AC source keyword for use in AC small-signal analysis.
acmag	Magnitude (RMS) of the AC source, in volts.
acphase	Phase of the AC source, in degrees. Default=0.0.
M	Multiplier, to simulate multiple parallel current sources. HSPICE multiplies source current by M. Default=1.0.
power=[0 1 2 W dbm]	<p>Power switch</p> <ul style="list-style-type: none"> ▪ When 0 (default), the element is treated as a voltage or current source. ▪ When 1 or W, the element is treated as a power source, realized as a voltage source with a series impedance or a current source with a parallel impedance. In this case, the source value is interpreted as RMS available power in units of Watts. ▪ When 2 or dbm, the element is treated as a voltage source with a series impedance or a current source with a parallel impedance. Values are in dbms. You can use this parameter for transient analysis if the power source is either DC or SIN.

Chapter 10: Sources and Stimuli

Independent Source Elements

For a listing of output templates for independent voltage source (V-element) models (HSPICE only), see [Table 40 on page 441](#).

For a listing of output templates for independent current source (I-element) models (HSPICE only), see [Table 41 on page 441](#).

For a listing of output templates for independent current (I-element) models, see [Table 41 on page 441](#).

Example 1

```
VX 1 0 5V
```

where:

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

Example 2

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

where:

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

Example 3

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

where:

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

Example 4

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

where:

- The piecewise-linear relationship defines the time-varying response for the IG current source, which is 1 milliamp at time=0, and 5 milliamps at 25 milliseconds.

- The positive terminal connects to node 8.
- The negative terminal connects to node 7.

Example 5

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

where:

- The `DC=0` keyword specifies the initial DC bias value for the `VCC` voltage source.
- The piecewise-linear relationship defines the time-varying response for the `VCC` voltage source, which is 0 volts at time=0, `VCC` from 10 to 15 nanoseconds, and back to 0 volts at 20 nanoseconds.
- The positive terminal connects to the `in` node.
- The negative terminal connects to the `out` node.
- HSPICE will determine the operating point using the specified DC bias value. If no initial DC bias value is specified, HSPICE determines the operating point using the initial transient source value and outputs the following informational message: `DC voltage reset to initial transient source value.`

Example 6

```
Vin1 in 0 DC=0 PULSE(-1 1 10n 1n 1n 20n 40n)
```

where:

- The `DC=0` specifies the DC bias for the `Vin1` voltage source.
- The positive terminal connects to the `in` node.
- The negative terminal connects to the ground (`0`) node.
- The output high voltage is 1 V.
- The output low voltage is -1 V.
- The delay is 10 ns.
- The rise and fall times are each 1 ns.
- The high pulse width is 20 ns.
- The period is 40 ns.

Chapter 10: Sources and Stimuli

Independent Source Elements

- HSPICE will determine the operating point using the specified DC bias value. If no initial DC bias value is specified, HSPICE determines the operating point using the initial transient source value and output the following informational message: DC voltage reset to initial transient source value.

Example 7

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

where:

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

Example 8

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

where:

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

Example 9

```
VMEAS 12 9
```

where:

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

DC Sources

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

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

- The first two examples specify a DC voltage source of 5 V, connected between node 1 and ground.
- The third and fourth examples specify a 5 mA DC current source, between node 1 and ground.
- The direction of current in both sources is from node 1 to ground.

AC Sources

AC analysis uses AC current and voltage sources as impulse functions. To specify the magnitude and phase of the impulse, use the `AC` keyword.

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

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

Transient Sources

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

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

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

Mixed Sources

Mixed sources specify source values for more than one type of analysis. For example, you can specify a DC source, an AC source, and a transient source, all of which connect to the same nodes. In this case, when you run specific analyses, HSPICE selects the appropriate DC, AC, or transient source. In a DC analysis, if the mixed source DC value is missing, the simulation replaces it with the zero-time value of its transient source by default. Otherwise, for DC analysis, HSPICE uses the DC source value for operating point calculation for all analyses except TRAN; TRAN analysis calculates an additional operating point with the zero-time source transient value.

Example

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

Where,

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

Each source connects between nodes 13 and 2.

For DC analysis, HSPICE uses its dc value 0.5v, and selects this operating point for the AC analysis to come. In transient analysis, HSPICE calculates another operating point by using zero source value because the sinusoidal source is zero at time zero.

Independent Source Functions

HSPICE uses the following types of independent source functions:

- Trapezoidal pulse (`PULSE` function)
- Sinusoidal (`SIN` function)
- Exponential (`EXP` function)
- Piecewise linear (`PWL` function)

- Single-frequency FM (`SFFM` function)
- Single-frequency AM (`AM` function)
- Pattern (`PAT` function)
- Pseudo Random-Bit Generator Source (`PRBS` function)

HSPICE also provides a data-driven version of `PWL` (not supported in HSPICE RF). If you use the data-driven `PWL`, you can reuse the results of an experiment or of a previous simulation, as one or more input sources for a transient simulation. See [Sources Examples\(/datadriven_pwl.sp\)](#) for syntax of a data-driven sweep.

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

The following sections discuss these topics:

- [Trapezoidal Pulse Source](#)
- [Sinusoidal Source Function](#)
- [Exponential Source Function](#)
- [Repeat Option in Sinusoidal and Exponential Sources](#)
- [Piecewise Linear Source](#)
- [PWLZ High Impedance State](#)
- [Single-Frequency FM Source](#)
- [Single-Frequency AM Source](#)
- [Pattern Source](#)
- [Pseudo Random-Bit Generator Source](#)

Trapezoidal Pulse Source

HSPICE provides a trapezoidal pulse source function, which starts with an initial delay from the beginning of the transient simulation interval, to an onset ramp. During the onset ramp, the voltage or current changes linearly from its initial value to the pulse plateau value. After the pulse plateau, the voltage or

Chapter 10: Sources and Stimuli
Independent Source Functions

current moves linearly along a recovery ramp back to its initial value. The entire pulse repeats, with a period named *per*, from onset to onset.

```
Vxxx n+ n- PU[LSE] [(] v1 v2 [td [tr [tf [pw [per]]]]] [)]
+ [PERJITTER=val [SEED=val]]
Ixxx n+ n- PU[LSE] [(] v1 v2 [td [tr [tf [pw [per]]]]] [)]
+ [PERJITTER=val [SEED=val]]
```

Parameter	Description
Vxxx, Ixxx	Independent voltage/ current source, which exhibits the pulse response.
PULSE	Keyword for a pulsed time-varying source. The short form is PU.
v1	Initial value of voltage or current before the pulse onset (units: volts/amps).
v2	Pulse plateau value (units of volts or amps).
td	Delay (propagation) time in seconds from the beginning of the transient interval to the first onset ramp. Default=0.0
tr	Duration of the onset ramp (in seconds) from the initial value to the pulse plateau value (reverse transit time). default=TSTEP
tf	Duration of the recovery ramp (in seconds) from the pulse plateau back to the initial value (forward transit time). default=TSTEP
pw	Pulse width (the width of the plateau portion of the pulse), in seconds. Default=TSTOP.
per	Pulse repetition period, in seconds. default=TSTEP
perjitter	RMS value for period jitter, adjusts the magnitude of the random time.
seed	Used to generate random number sequences with different seed value. The value is a negative integer, defaults to -1.

Table 18 Time-Value Relationship for a PULSE Source

Time	Value
0	v1
td	v1

Table 18 Time-Value Relationship for a PULSE Source (Continued)

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

Linear interpolation determines the intermediate points.

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

Effect of Jitter on the PULSE Source

The effect of jitter on the PULSE source results in random shifts of the rise and fall transitions that normally take place at:

RISE edge: $td + n \cdot T_0 \leq t \leq td + n \cdot T_0$

FALL edge: $td + pw + n \cdot T_0 \leq t \leq td + tr + pw + tf + n \cdot T_0$

The jitter effect is equivalent to introducing random shifts in the period $0T$ consistent with the 1st order jitter model based on Period Jitter, also according to the expression for period variations $T_0 \rightarrow T_j = T_0 + \Delta T(t)$. The specified syntax determines the first period of the PULSE source, yet all subsequent pulse periods are random according to $T_0 + \Delta T(t)$.

The PULSE source with jitter still maintains constant rise and fall times. This creates some uncertainty in how the pulse width (pw) varies as the period varies due to jitter.

HSPICE RF uses a special calculation that holds the rise and fall times (tr and tf) constant, and also holds the 50% Duty Cycle constant. The halfway points on the rise and fall times determines the 50 percent duty cycle definition. This results in the pulse width (pw) change due to jitter variations as:

Chapter 10: Sources and Stimuli

Independent Source Functions

Equation 8

$$pw_j = pw \cdot \left(\frac{T_j}{T_0} \right) + \frac{1}{2} \cdot (tr + tf) \cdot \left(\frac{T_j}{T_0} - 1 \right)$$

Also, this sets the minimum period for a PULSE source with jitter to be $tr + tf$, resulting in the extreme case of a sawtooth waveform.

A Gaussian random number generator computes the random $\Delta T(t)$ variations after each leading edge of the clock sources. For flexibility, the SEED parameter (integer) generates different random number sequences when you specify different SEED integers for initialization.

Example 1

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

- The output high voltage is 1 V.
- The output low voltage is -1 V.
- The delay is 2 ns.
- The rise and fall time are each 2 ns.
- The high pulse width is 50 ns.
- The period is 100 ns.
- The RMS value for period jitter is 10ns.
- The seed is -1.

```
VIN 3 0 PULSE (-1 1 2NS 2NS 2NS 50NS 100NS) perjitter=10ns seed=-1
```

Example 2

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

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

Example 3

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

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

- The pulse width is 20 nanoseconds.
- The pulse period is 50 nanoseconds.

This example uses demonstration netlist `pulse.sp`, which is available in directory `$installdir/demo/hspice/sources`:

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

Example 24 on page 245 shows the result of simulating this netlist, in HSPICE.

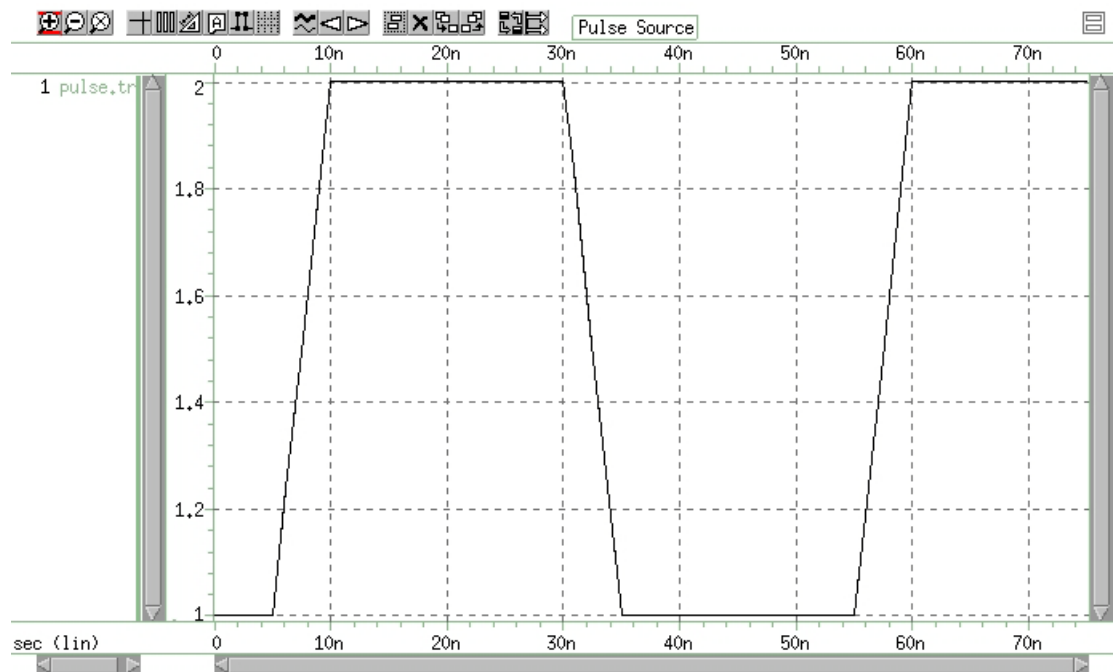


Figure 24 Pulse Source Function

Sinusoidal Source Function

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

Chapter 10: Sources and Stimuli

Independent Source Functions

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

```
Vxxx n+ n- SIN [(] vo va [freq [td [q [j]]]] [)]  
+ [[PERJITTER=val] [SEED=val]]  
Ixxx n+ n- SIN [(] vo va [freq [td [q [j]]]] [)]  
+ [[PERJITTER=val] [SEED=val]]
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source that exhibits the sinusoidal response.
SIN	Keyword for a sinusoidal time-varying source.
vo	Voltage or current offset, in volts or amps.
va	Voltage or current peak value (vpeak), in volts or amps.
freq	Source frequency in Hz. Default=1/TSTOP.
td	Time (propagation) delay before beginning the sinusoidal variation, in seconds. Default=0.0. Response is 0 volts or amps, until HSPICE reaches the delay value, even with a non-zero DC voltage.
q	Damping factor, in units of 1/seconds. Default=0.0.
j	Phase delay, in units of degrees. Default=0.0.
perjitter	RMS value for period jitter, used to adjust the magnitude of the random time.
seed	Used to generate random number sequences with different seed value. The value is a negative integer, defaults to -1.

The following table of expressions defines the waveform shape:

Table 19 Waveform Shape Expressions

Time	Value
0 to t_d	$v_o + v_a \cdot \text{SIN}\left(\frac{2 \cdot \Pi \cdot \phi}{360}\right)$
t_d to t_{stop}	$v_o + v_a \cdot \text{Exp}[-(\text{Time} - t_d) \Rightarrow q] \cdot \text{SIN}\left\{2 \cdot \Pi \cdot \left[\text{freq} \cdot (\text{time} - t_d + x(t)) + \frac{j}{360}\right]\right\}$

Where q and j are the damping factor and phase delay in the syntax.

In these expressions, TSTOP is the final time.

Example

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

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

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

See [Figure 25 on page 248](#) for a plot of the source output.

Chapter 10: Sources and Stimuli
Independent Source Functions

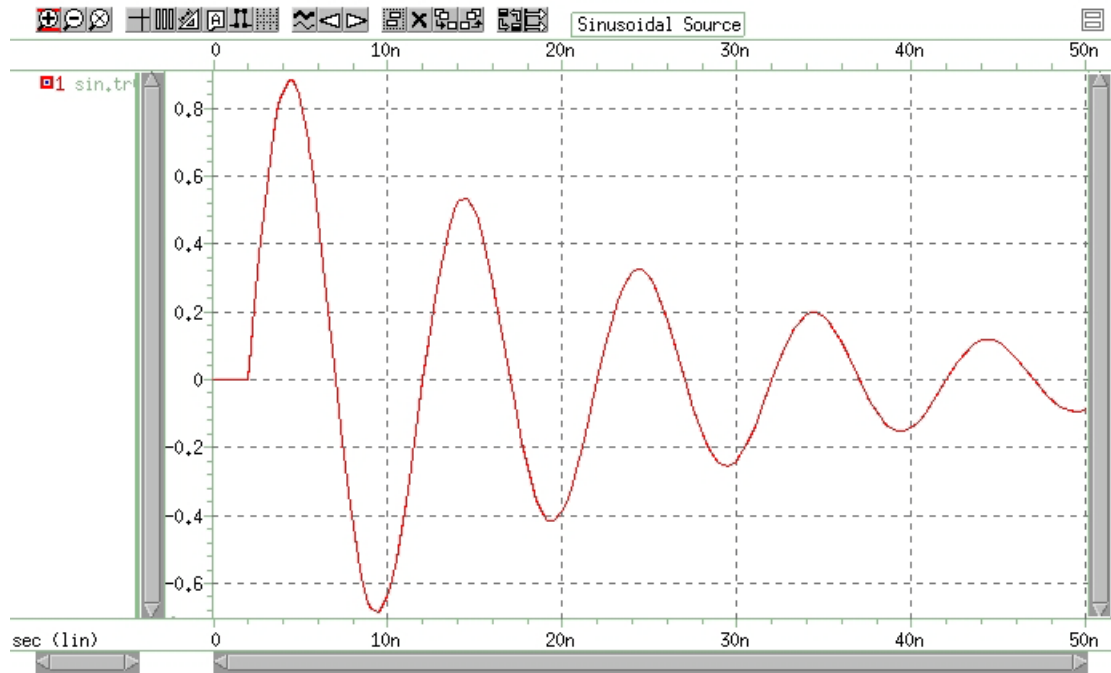


Figure 25 Sinusoidal Source Function

This example uses demonstration netlist `sin.sp`, which is available in directory `$installdir/demo/hspice/sources`:

```
*file: sin.sp
sinusoidal source
.options post

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

Table 20 SIN Voltage Source

Parameter	Value
initial voltage	0 volts
pulse voltage	1 volt
delay time	2 nanoseconds

Table 20 SIN Voltage Source

Parameter	Value
frequency	100 MHz
damping factor	50 MHz

Exponential Source Function

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

```
Vxxx n+ n- EXP [(] v1 v2 [td1 [t1 [td2 [t2]]]] [)]
Ixxx n+ n- EXP [(] v1 v2 [td1 [t1 [td2 [t2]]]] [)]
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source, with an exponential response.
EXP	Keyword for an exponential time-varying source.
v1	Initial value of voltage or current, in volts or amps.
v2	Pulsed value of voltage or current, in volts or amps.
td1	Rise delay time, in seconds. Default=0.0.
td2	Fall delay time, in seconds. Default=td1+TSTEP.
t1	Rise time constant, in seconds. default=TSTEP
t2	Fall time constant, in seconds. default=TSTEP

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

Chapter 10: Sources and Stimuli
Independent Source Functions

The following table of expressions defines the waveform shape:

Table 21 Waveform Shape Definitions

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

Example

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

The above example describes an exponential transient source, which connects between nodes 3 and 0. In this source:

- Initial t=0 voltage is -4 V.
- Final voltage is -1 V.
- Waveform rises exponentially from -4 V to -1 V with a time constant of 30 ns.
- At 60 ns, the waveform starts dropping to -4 V again, with a time constant of 40 ns.

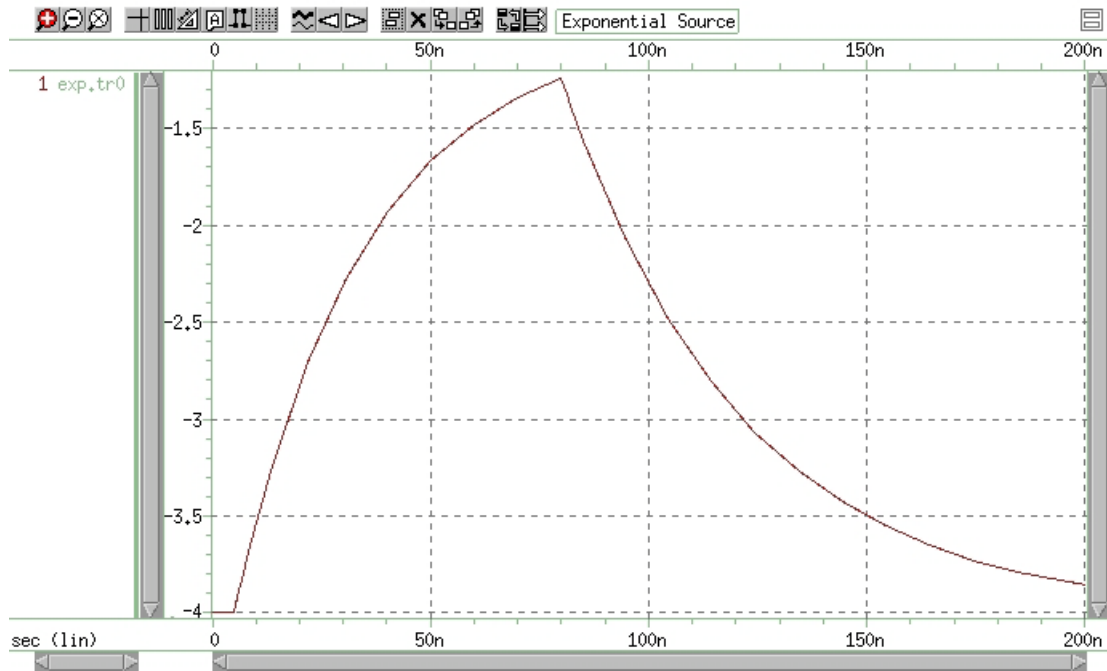


Figure 26 Exponential Source Function

This example uses demonstration netlist `exp.sp`, which is available in directory `$installdir/demo/hspice/sources`:

```
*file: exp.sp      exponential independant source
.options post
.param v0=-4 va=-1 td1=5n tau1=30n tau2=40n td2=80n
v 1 0 exp(v0 va td1 tau1 td2 tau2)
r 1 0 1
.tran .05n 200n
.end
```

This example shows an entire netlist, which contains an `EXP` voltage source. In this source:

- Initial $t=0$ voltage is -4 V.
- Final voltage is -1 V.
- Waveform rises exponentially from -4 V to -1 V with a time constant of 30 ns.
- At 80 ns, the waveform starts dropping to -4 V again, with a time constant of 40 ns.

Repeat Option in Sinusoidal and Exponential Sources

You can use the `R-repeat` option in `SIN` (damping) and `EXP` (damping) sources. The syntax is:

```
sin (0 1 300meg 10n 1e8 R=val)
```

or

```
exp (1 2 10n 30n 60n 20n R=val)
```

Where `val` can be a number specified or a parameter. A syntax of `R=R` is acceptable. For `SIN/EXP`, the `R` option means the length of each repeated period. For example:

```
VIN1 in out SIN (1.2 0.11 4G 5.5NS 2e9 180 R=1.2n)
```

This `SIN` function repeats from the beginning of the `SIN` waveform to `R=1.2n`.

Note: The `SIN/EXP R-option` is not fully consistent with `PWL`. For `PWL`, the `R-option` specifies the start point of the waveform to repeat. In the example below, this `pwl` function repeats from `r=2n` to the last defined points (`5n, 0`).

```
v1 1 0 pwl 0 0 0.5n 0.5 2n 0.8 5n 0 r=2n
```

Piecewise Linear Source

HSPICE provides a piecewise linear source function in an independent voltage or current source.

Note: For controlled sources (E, F, G and H), the '(1)' or '(2)' after the `PWL` syntax means that the transfer function described by the piecewise-linear function is a 1-dimensional or 2-dimensional `PWL` function.

The following sections discuss these topics:

- [General PWL Form](#)
- [MSINC and ASPEC Form](#)
- [Data-Driven Piecewise Linear Source](#)
- [File-Driven PWL Source](#)

General PWL Form

```
Vxxx n+ n- PWL [(] t1 v1 [t2 v2 t3 v3...] [R= [repeat]]
+ [TD=delay] [)]
Ixxx n+ n- PWL [(] t1 v1 [t2 v2 t3 v3...] [R= [repeat]]
+ [TD=delay] [)]
```

MSINC and ASPEC Form

```
Vxxx n+ n- PL [(] t1 v1 [t2 v2 t3 v3...] [R= [repeat]]
+ [TD=delay] [)]
Ixxx n+ n- PL [(] t1 v1 [t2 v2 t3 v3...] [R= [repeat]]
+ [TD=delay] [)]
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source; uses a piecewise linear response.
PWL	Keyword for a piecewise linear time-varying source.
v1 v2 ... vn	Current or voltage values at the corresponding timepoint.
t1 t2 ... tn	Timepoint values, where the corresponding current or voltage value is valid.
R=repeat	Keyword and time value to specify a repeating function. With no argument, the source repeats from the beginning of the function. <i>repeat</i> is the time, in units of seconds, which specifies the start point of the waveform to repeat. This time needs to be less than the greatest time point, <i>tn</i> .
TD=delay	Time, in units of seconds, which specifies the length of time to delay (propagation delay) the piecewise linear function.

- Each pair of values (*t1*, *v1*) specifies that the value of the source is *v1* (in volts or amps), at time *t1*.
- Linear interpolation between the time points determines the value of the source, at intermediate values of time.
- The PL form of the function accommodates ASPEC style formats, and reverses the order of the time-voltage pairs to voltage-time pairs.
- If you do not specify a time-zero point, HSPICE uses the DC value of the source, as the time-zero source value.

HSPICE does not force the source to terminate at the `TSTOP` value, specified in the `.TRAN` statement.

If the slope of the piecewise linear function changes below a specified tolerance, the timestep algorithm might not choose the specified time points as simulation time points. To obtain a value for the source voltage or current, HSPICE extrapolates neighboring values. As a result, the simulated voltage might deviate slightly from the voltage specified in the `PWL` list. To force HSPICE to use the specified values, use `.OPTION SLOPETOL`, which reduces the slope change tolerance.

`R` causes the function to repeat. You can specify a value after this `R`, to indicate the beginning of the function to repeat. The repeat time must equal a breakpoint in the function. For example, if $t_1=1$, $t_2=2$, $t_3=3$, and $t_4=4$, then the repeat value can be 1, 2, or 3.

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

Example

This example uses demonstration netlist `pwl.sp`, which is available in directory `$installdir/demo/hspice/sources`:

```
file pwl.sp repeated piecewise linear source
.option post
.tran 5n 500n
v1 1 0 pwl 60n 0v, 120n 0v, 130n 5v, 170n 5v, 180n 0v, r
r1 1 0 1

v2 2 0 pl 0v 60n, 0v 120n, 5v 130n, 5v 170n, 0v 180n, r 60n
r2 2 0 1
.end
```

This example shows an entire netlist, which contains two piecewise linear voltage sources. The two sources have the same function:

- First is in normal format. The repeat starts at the beginning of the function.
- Second is in ASPEC format. The repeat starts at the first timepoint.

See [Figure 27](#) for the difference in responses.

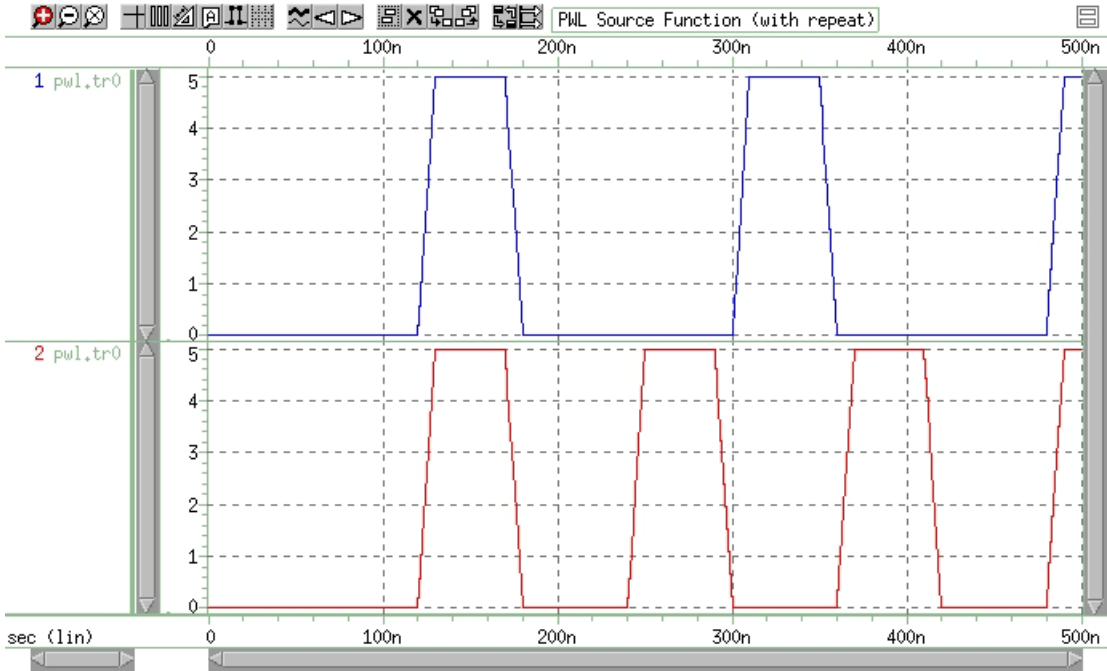


Figure 27 Results of Using the Repeat Function

Data-Driven Piecewise Linear Source

HSPICE provides a data-driven piecewise linear source function in an independent voltage or current source.

```
Vxxx n+ n- PWL (TIME, PV)
Ixxx n+ n- PWL (TIME, PV)
.DATA dataname
TIME PVt1 v1t2 v2t3 v3t4 v4. . . .
.ENDDATA
.TRAN DATA=datanam
```

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

Use with a .DATA statement that contains time-value pairs. For each t_n-v_n (time-value) pair that you specify in the .DATA block, the data-driven PWL function outputs a current or voltage of the specified t_n duration and with the

specified *vn* amplitude. This source enables simulation results reuse as an input source in another simulation. Transient analysis must be data-driven.

Example

This example uses demonstration netlist `datadriven_pwl.sp`, which is available in directory `$installdir/demo/hspice/sources`:

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

This example is an entire netlist, containing two data-driven, piecewise linear voltage sources. The `.DATA` statement contains the two sets of values referenced in the `pv1` and `pv2` sources. `.TRAN` references the data name; there should be no time in `.TRAN` because `DATA` includes time.

File-Driven PWL Source

You can specify a text file containing time, voltage (or current) pairs into a PWL source. The text file could be either a table,

```
0 0
1n 0.1
2n 0.2
```

or a list (commas optional) `0 0, 1n 0.1, 2n 0.2, ...`

```
Vxxx n1 n2 PWL PWLFILE='filename' [col1, [col2]] [R [=repeat]]
+ [TD=delay]
Ixxx n1 n2 PWL PWLFILE='filename' [col1, [col2]] [R [=repeat]]
+ [TD=delay]
```

Here, `PWLFILE` can use a string parameter.

Example

```
Vit n1 n2 PWL PWLFILE='Imod.dat'
.parameter pwl_data_file=str('pwl.dat')
V1 in out PWL PWLFILE=str(pwl_data_file)
```

PWLZ High Impedance State

The high Z state of the PWLZ source adds the capability to disconnect the PWL source for time periods marked with the keyword Z.

```
Vxxx n+ n- PWLZ [(] t1 val1 [t2 val2 t3 z t4 val4 ...]
+ [R [=repeat]] [TD=delay] [)]
```

You can use the keyword `z` in place of the source value. You can disconnect the voltage source for time periods you mark with the keyword `z`.

Parameters	Description
Vxxx	Independent voltage source; capable of having high impedance during periods of times.
PWLZ	Keyword for a piecewise linear time-varying source with high impedance.
val1 val2... valn	Voltage values at the corresponding time points.
t1 t2 ... tn	Time point values.
R= <i>repeat</i>	Keyword and time value to specify a repeating function. With no argument, the source repeats from the beginning of the function. <i>repeat</i> is the time, in units of seconds, which specifies the start point of the waveform to repeat. This time needs to be less than the greatest time point, <i>tn</i> .
TD = <i>delay</i>	Time, in units of seconds, which specifies the length of time to delay (propagation delay) the piecewise linear function.

Example

In this example, node 5 connects to a 0V source at time 0, and rises from 0V to 0.75V in 2 ns. Between 2 ns and 10 ns, the voltage-source value rises from 0.75V to 1.5V. The voltage-source value stays at 1.5V between 10 ns and 50

ns. At 50 ns, node 5 disconnects from the voltage source until 60 ns. It re-connects to a 0.75V voltage source after 60 ns.

```
VXD 5 0 pwlz (0 0 2N 0.75 10N 1.5 50N z 60N 0.75)
```

Single-Frequency FM Source

HSPICE provides a single-frequency FM source function, in an independent voltage or current source.

```
Vxxx n+ n- SFFM [(] vo va [fc [mdi [fs]] (]]
Ixxx n+ n- SFFM [(] vo va [fc [mdi [fs]] (]]
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source, which exhibits the frequency-modulated response.
SFFM	Keyword for a single-frequency, frequency-modulated, time-varying source.
vo	Output voltage or current offset, in volts or amps.
va	Output voltage or current amplitude, in volts or amps.
fc	Carrier frequency, in Hz. Default=1/TSTOP.
mdi	Modulation index, which determines the magnitude of deviation from the carrier frequency. Values normally lie between 1 and 10. Default=0.0.
fs	Signal frequency, in Hz. Default=1/TSTOP.

The following expression defines the waveform shape:

Equation 9

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

Example

This example uses demonstration netlist `sffm.sp`, which is available in directory `$installdir/demo/hspice/sources`:


```
*file: sffm.sp  
frequency modulation source  
.options post  
vsff1 15 0 dc 3v sffm(0v 1v 20k 10 5k)  
rssf1 15 0 1  
.tran .001ms .5ms  
.probe tran v(15)  
.end
```

This example shows an entire netlist, which contains a single-frequency, frequency-modulated voltage source. In this source.

- The offset voltage is 0 volts.
- The maximum voltage is 1 millivolt.
- The carrier frequency is 20 kHz.
- The signal is 5 kHz, with a modulation index of 10 (the maximum wavelength is roughly 10 times as long as the minimum).

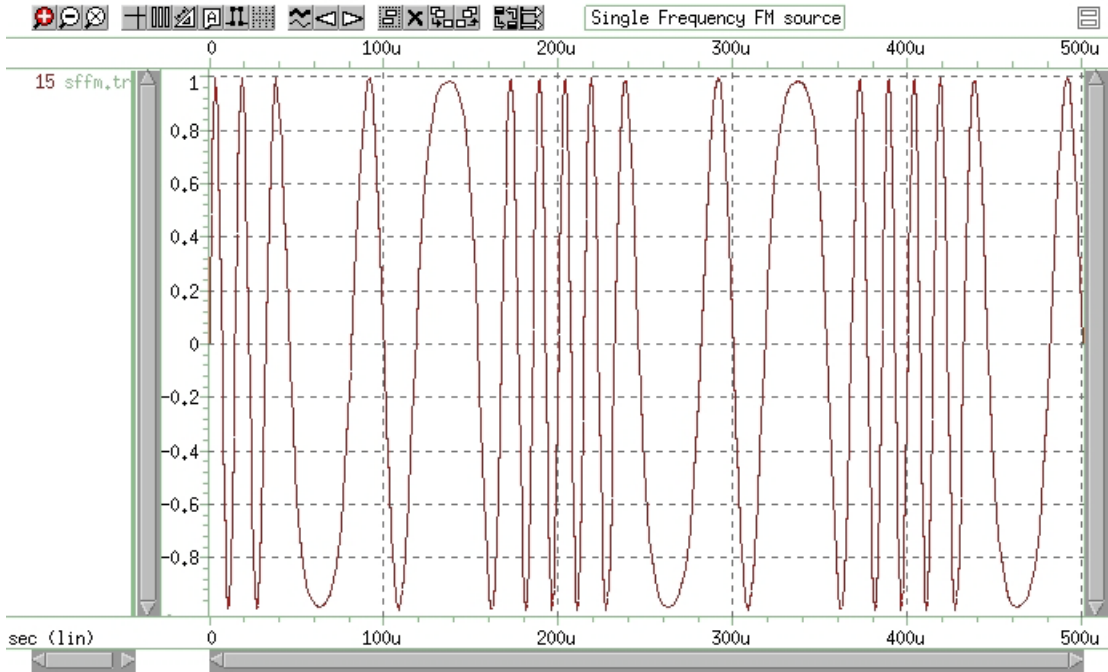


Figure 28 Single Frequency FM Source

Single-Frequency AM Source

HSPICE provides a single-frequency AM source function in an independent voltage or current source.

```
Vxxx n+ n- AM [ ( ) sa oc fm fc [td] [ ] ]
Ixxx n+ n- AM [ ( ) sa oc fm fc [td] [ ] ]
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source, which exhibits the amplitude-modulated response.
AM	Keyword for an amplitude-modulated, time-varying source.
sa	Signal amplitude, in volts or amps. Default=0.0.
fc	Carrier frequency, in hertz. Default=0.0.
fm	Modulation frequency, in hertz. Default=1/TSTOP.
oc	Offset constant, a unitless constant that determines the absolute magnitude of the modulation. Default=0.0.
td	Delay time (propagation delay) before the start of the signal, in seconds. Default=0.0.

The following expression defines the waveform shape:

Equation 10

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

Example

This example uses demonstration netlist `amsrc.sp`, which is available in directory `$installdir/demo/hspice/sources`:

```
*file amsrc.sp amplitude modulation
.option post
.tran .01m 20m

v1 1 0 am(10 1 100 1k 1m)
r1 1 0 1

v2 2 0 am(2.5 4 100 1k 1m)
r2 2 0 1

v3 3 0 am(10 1 1k 100 1m)
r3 3 0 1
.end
```

This example shows an entire netlist, which contains three amplitude-modulated voltage sources.

- In the first source:
 - Amplitude is 10.
 - Offset constant is 1.
 - Carrier frequency is 1 kHz.
 - Modulation frequency of 100 Hz.
 - Delay is 1 millisecond.
- In the second source, only the amplitude and offset constant differ from the first source:
 - Amplitude is 2.5.
 - Offset constant is 4.
 - Carrier frequency is 1 kHz.
 - Modulation frequency of 100 Hz.
 - Delay is 1 millisecond.
- The third source exchanges the carrier and modulation frequencies, compared to the first source:
 - Amplitude is 10.
 - Offset constant is 1.
 - Carrier frequency is 100 Hz.
 - Modulation frequency of 1 kHz.

- Delay is 1 millisecond.

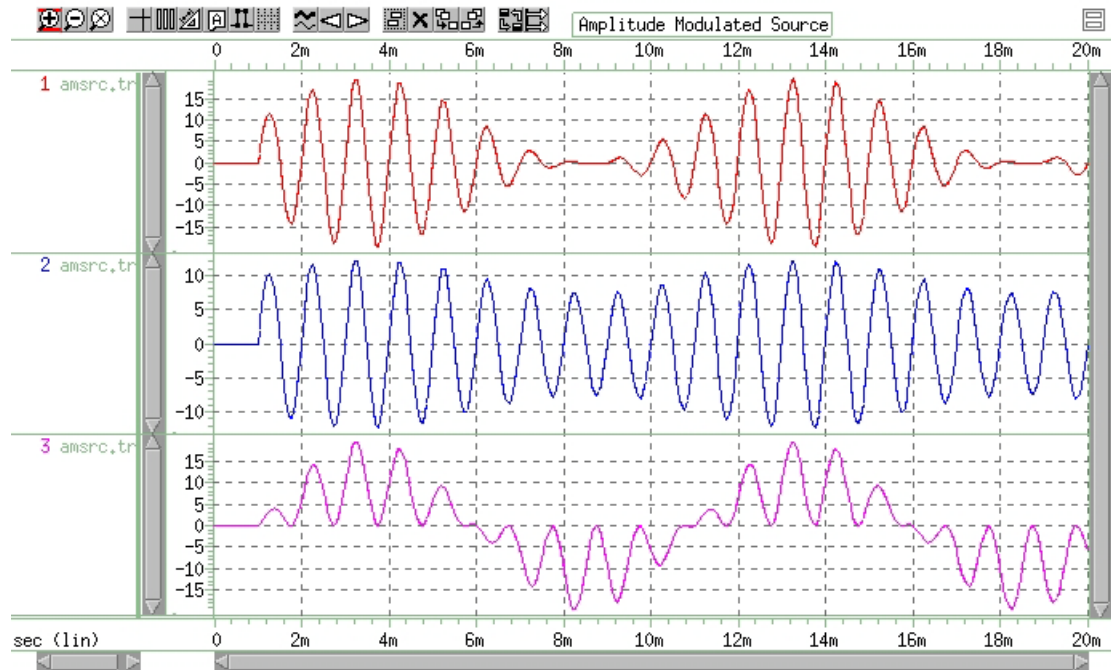


Figure 29 Amplitude Modulation Plot

Pattern Source

HSPICE provides a pattern source function, in an independent voltage or current source. The pattern source function uses four states, '1','0','m', and 'z', which represent the high, low, middle voltage, or current and high impedance state respectively. The series of these four states is a “b-string.” (HSPICE supports eight-bit data byte conversion to a 10-bit transmission character—8B/10B encoding for the PAT keyword.)

```
Vxxx n+ n- PAT [(] vhi vlo td tr tf tsample data [RB=val]
+ [R=repeat] [ENCODE=DW8b10b] [RD_INIT=0|1] [)]
Ixxx n+ n- PAT [(] vhi vlo td tr tf tsample data [RB=val]
+ [R=repeat] [ENCODE=DW8b10b] [RD_INIT=0|1] [)]
```

Parameter	Description
Vxxx, Ixxx	Independent voltage source that exhibits a pattern response.

Parameter	Description
PAT	Keyword for a pattern time-varying source.
vhi	High voltage or current value for pattern sources (units of volts or amps).
vlo	Low voltage or current value for pattern sources (units of volts or amps).
td	Delay (propagation) time in seconds from the beginning of the transient interval to the first onset ramp. It can be negative. The state in the delay time is the same as the first state specified in data.
tr	Duration of the onset ramp (in seconds) from the low value to the high value (reverse transit time).
tf	Duration of the recovery ramp (in seconds) from the high value back to the low value (forward transit time).
tsample	Time spent at '0' or '1' or 'M' or 'Z' pattern value (in seconds).
data	String of '1', '0', 'M', 'Z' representing a pattern source, or 'K' representing 8b/10b encoding control string (or control word). If the first alphabetic character is 'B', which represents it as a binary bit stream, the series is a b-string. '1' represents the value for high voltage or current, '0' is the value for low voltage or current, 'M' represents the value which is equal to $0.5 \cdot (v_{hi} + v_{lo})$. If the first alphabetic character is 'K', which represents the control string in the 8b10b encoder. Then a series binary bit stream (such as k00010001, similar to the original b-string) follows the k-string. In a K-string, the bit stream must be '1' or '0'. You can not use 'M' and 'Z'. The definition of K-string follows the form of "DW_8b10b_enc" in the Synopsys DesignWare Building Block IP. If you do not apply <code>encode=dw8b10b</code> in the netlist, then the K-string takes the same function as B-string.
RB	Keyword to specify the starting bit when repeating. The repeat data starts from the bit indicated by RB. RB must be an integer. HSPICE reports an error if the value is larger than the length of the b-string. If the value is less than 1, HSPICE resets it to 1 automatically.
R=repeat	Keyword to specify the number of times to execute the repeating operation. With no argument, the source repeats from the beginning of the b-string. If R=-1, it means the repeating operation continues forever. R must be an integer and if it is less than -1, HSPICE resets it to 0 automatically.
ENCODE=DW8b10b	Keyword to specify 8b/10b encoding.

Parameter	Description
RD_INIT=0 1	<p>Initial value of Running Disparity. The one bit memory that recalls the bias of the last unbalanced code word is the Running Disparity. 1: Specifies that a Running Disparity value of zero is synonymous with negative Running Disparity (?).</p> <ul style="list-style-type: none"> ▪ 0: Specifies that a Running Disparity value of one is synonymous with negative Running Disparity (-) ▪ 1: Specifies that a Running Disparity value of one is synonymous with positive Running Disparity (+).

The time from 0 to the first transition is:

$$t_{\text{delay}} + N \cdot t_{\text{sample}} - t_r(t_f) / 2$$

- N is the number of the same bit from the beginning.
- If the first transition is rising, this equation uses t_r .
- If the first transition is falling, it uses t_f .

Example

The following example shows a pattern source with two b-strings:

```
*FILE: pattern source general form
v1 1 0 pat (5 0 0n 1n 1n 5n b1011 r=1 rb=2 b0m1z)
r1 1 0 1
```

In this pattern:

- High voltage is 5 v
- Low voltage is 0 v
- Time delay is 0 n
- Rise time is 1 n
- Fall time is 1 n
- Sample time is 5 n

The first b-string is 1011, which repeats once and then repeats from the second bit, which is 0. The second b-string is 0m1z. Since the example specifies neither R and RB here, HSPICE sets them to the default values, R=0, RB=1.

Example

The following b-string and its repeat time R and repeating start bit RB cannot use a parameter — HSPICE considers it as a undivided unit which only the .PAT command can define.

```
*FILE:pattern source using parameter
.param td=40ps tr=20ps tf=80ps tsample=400ps
VIN 1 0 PAT (2 0 td tr tf tsample b1010110 r=2)
r1 1 0 1
```

In this pattern:

- High voltage is 2 V.
- Low voltage is 0 V.
- Time delay is 40 ps.
- Rise time is 20 ps.
- Fall time is 80 ps.
- Sample time is 400 ps.
- Data is 1010110.

The following sections discuss these topics:

- [Nested-Structure Pattern Source](#)
- [Pattern-Command Driven Pattern Source](#)
- [Workaround to 1024 Character Limitation for Long Pattern Sources](#)

Nested-Structure Pattern Source

HSPICE provides Nested Structure (NS) for the pattern source function to construct complex waveforms. NS is a combination of a b-string and other nested structures defined in a .PAT command (see [Pattern-Command Driven Pattern Source on page 266](#)).

The following general syntax is for an NS pattern source.

```
Vxxx n+ n- PAT [(] vhi vlo td tr tf tsample
+ [component 1 ... component n] [RB=val] [R=repeat] [)]
Ixxx n+ n- PAT [(] vhi vlo td tr tf tsample
```

Chapter 10: Sources and Stimuli

Independent Source Functions

```
+ [component 1 ... component n] [RB=val] [R=repeat] [ ]]
```

Parameter	Description
component	Component is the element that makes up NS, which can be a b-string or a patname defined in other PAT commands. You must use Brackets ([]).
RB=val	Keyword to specify the starting component when repeating. The repeat data starts from the component indicated by RB. RB must be an integer. HSPICE reports an error if RB is larger than the length of the NS. If RB is less than 1, HSPICE resets it automatically to 1.
R=repeat	Keyword to specify number of executions of the repeating operation. With no argument, the source repeats from the beginning of the NS. If R=-1, the repeating operation continues indefinitely. R must be an integer, and if it is less than -1, HSPICE resets it automatically to 0.

If the component is a b-string, you can follow it with *R=repeat* and *RB=val* to specify the repeat time and repeating start bit.

Example

```
*FILE: Pattern source using nested structure
v1 1 0 pat (5 0 0n 1n 1n 5n [b1011 r=1 rb=2 b0m1z] r=2 rb=2)
r1 1 0 1
```

When expanding the nested structure, you generate the pattern source as:

```
'b1011 r=1 rb=2 b0m1z b0m1z b0m1z'
```

The whole NS repeats twice, and each time it repeats from the second b0m1z component.

Pattern-Command Driven Pattern Source

The following general syntax is for including a pattern-command driven pattern source in an independent voltage or current source. You can reset the RB and R of a b-string or NS in an independent source. With no argument, the R and RB are the same when you define it in the pattern command.

```
Vxxx n+ n- PAT [(] vhi vlo td tr tf tsample PatName [RB=val]
+ [R=repeat] [)]
Ixxx n+ n- PAT [(] vhi vlo td tr tf tsample Patname [RB=val]
+ [R=repeat] [)]
```

Additional syntax applies to the .PAT command-driven pattern source:


```
.PAT PatName=data [RB=val] [R=repeat]  
.PAT PatName=[component 1...component n] [RB=val] [R=repeat]
```

The `PatName` is the pattern name that has an associated b-string or nested structure.

Example 1

```
v1 1 0 pat (5 0 0n 1n 1n 5n a1 a2 r=2 rb=2)  
.PAT a1=b1010 r=1 rb=1  
.PAT a2=b0101 r=1 rb=1
```

The final pattern source is:

```
b1010 r=1 rb=1 b0101 r=2 rb=2
```

When the independent source uses the `.PAT` command to specify its pattern source, you can reset `r` and `rb`.

Example 2

```
*FILE 2: Pattern source driven by pattern command  
v1 1 0 pat (5 0 0n 1n 1n 5n [a1 b0011] r=1 rb=1)  
.PAT a1=[b1010 b0101] r=0 rb=1
```

The final pattern source is:

```
b1010 b0101 b0011 b1010 b0101 b0011
```

A pattern source can reference the `a1` which is a predefined NS.

Workaround to 1024 Character Limitation for Long Pattern Sources

HSPICE limits pattern sources to 1024 characters, including the `.PAT` statement. To work around this limitation, you can define multiple pattern statements and combine them using another pattern statement.

Example

To combine the patterns, first define several separate patterns such as:

```
.pat p1=b10000000001000100001100100001010011000111  
.pat p2=b0100001001010100101101100011010111001111  
.pat p3=b1000010001100101001110100101011011010111
```

Next, define another pattern source that combines the previous ones

```
.pat p4=p1 p2 p3
```

Finally, use the combined pattern source in your source statement:

V1 1 0 pat (5 0 0n 20p 30p 200p p4)

Pseudo Random-Bit Generator Source

HSPICE Pseudo Random Bit Generator Source (PRBS) function, in an independent voltage or current source. You can use this function in several applications from cryptography and bit-error-rate measurement, to wireless communication systems employing spread spectrum or CDMA techniques. PRBS uses a Linear Feedback Shift Register (LFSR) to generate a pseudo random bit sequence. (HSPICE supports eight-bit data byte conversion to a 10-bit transmission character—8B/10B encoding for the LFSR keyword.)

```
Vxxx n+ n- LFSR [(] vlow vhigh tdelay trise tfall rate seed
+ [taps] [rout=val] [ENCODE=DW8b10b] [RD_INIT=0|1] [(]
Ixxx n+ n- LFSR [(] vlow vhigh tdelay trise tfall rate seed
+ [taps] [rout=val] [ENCODE=DW8b10b] [RD_INIT=0|1] [(]
```

Parameter	Description
LFSR	Specifies the voltage or current source as PRBS.
vlow	The minimum voltage or current level.
vhigh	The maximum voltage or current level.
tdelay	Specifies the initial time delay to the first transition.
trise	Specifies the duration of the onset ramp (in seconds) from the initial value to the pulse plateau value (reverse transit time).
tfall	Specifies the duration of the recovery ramp (in seconds) from the pulse plateau back to the initial value (forward transit time).
rate	The bit rate.
seed	The initial value (in integer form) of the LFSR loaded into the shift register. Because the operation of the register is deterministic, its current (or previous) state completely determines the stream of values produced by the register.
taps	The bits used to generate feedback.
rout	The output resistance.

Parameter	Description
ENCODE=DW8b10b	Keyword to specify 8b/10b encoding.
RD_INIT=0 1	Initial value of Running Disparity. The one bit memory that recalls the bias of the last unbalanced code word is the <i>Running Disparity</i> . 1: Specifies that a Running Disparity value of zero is synonymous with negative Running Disparity (?). <ul style="list-style-type: none"> ▪ 0: Specifies that a Running Disparity value of one is synonymous with negative Running Disparity (-) ▪ 1: Specifies that a Running Disparity value of one is synonymous with positive Running Disparity (+).

Example 1

The following example shows the pattern source that connects node in and node gnd.

Example of LFSR, output is 1100011111001101:

```
vin out gnd LFSR (0 1 0 1n 1n 10meg 3 [2, 5] rout=10)
.option POST
.tran 10p 10u
.end
```

Where,

- The output low voltage is 0 , and the output high voltage is 1 v.
- The delay time is 0 ms.
- The rise and fall times are each 1 ns.
- The bit rate is 10meg bits/s.
- The seed is 3 (bits: 00011).
- The taps are at the output of the 2nd and 5th shift registers and are xor'd together as the input to the first shift register.
- The output resistance is 10 ohm.

Example 2

The following example shows the pattern source connected between node 1 and node 0:

```
.PARAM td1=2.5m tr1=2n
vin 1 0 LFSR (2 4 td1 tr1 1n 6meg 2 [10, 5, 3, 2])
```

Where,

Chapter 10: Sources and Stimuli

Independent Source Functions

- The output low voltage is 2 v, and the output high voltage is 4 v.
- The delay is 2.5 ms.
- The rise time is 2 ns, and the fall time is 1 ns.
- The bit rate is 6meg bits/s.
- The seed is 2.
- The taps are [10, 5, 3, 2].
- The output resistance is 0 ohm.

Example 3

This example uses demonstration netlist `prbs.sp`, which is available in directory `$installdir/demo/hspice/sources`:

```
* prbs.sp
.OPTION POST
.TRAN 0.5n 50u
V1 1 0 LFSR (0 1 1u 1n 1n 10meg 1 [5, 2] rout=10)
R1 1 0 1
.END
```

Example 4

To generate a PRBS source that includes jitter, use the following steps:

1. Construct your usual linear feedback shift register (LFSR) generator.
2. Construct a matching (T,tr,tf) PULSE source as a clock, but add jitter to it with the PERJITTER keyword.
3. Use the PULSE source to gate (buffer) the LFSR output (through an ideal AND gate, VCCS, or similar function).

The following sections discuss these topics:

- [Linear Feedback Shift Register](#)
- [Conventions for Feedback Tap Specification](#)
- [Example: Noise Generator Used for a Pulse or DC Level](#)

Linear Feedback Shift Register

A LFSR consists of several simple-shift registers in which a binary-weighted modulo-2 sum of the taps feeds back to the input. The modulo-2 sum of two 1-bit binary numbers yields 0 if the two numbers are identical and 1 if they differ: $0+0=0$, $0+1=1$, or $1+1=0$. See [Figure 30 on page 271](#).

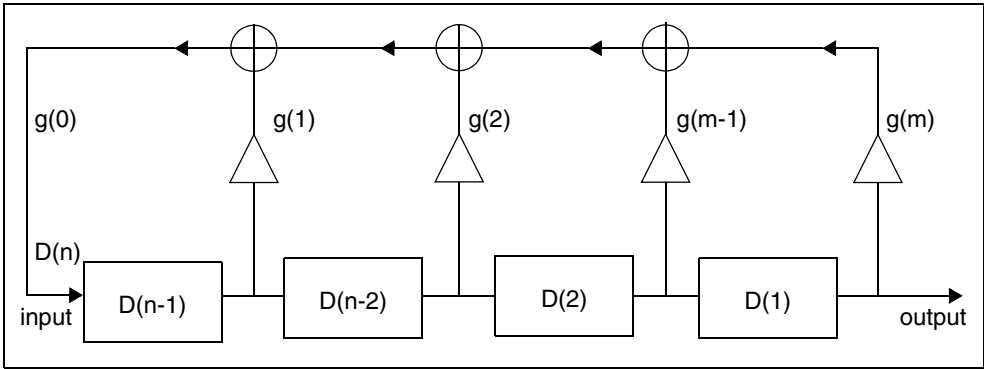


Figure 30 LFSR Diagram

For any given tap, the weight “ g_i ” is either 0, (meaning “no connection”), or 1, (meaning it feeds back). Two exceptions are g_0 and g_m , which are always 1 and therefore always connected. The g_m is not really a feedback connection, but rather an input-assigned feedback weight of the shift register for mathematical purposes.

The first number in your TAPS definition defines the maximum number of bits. For example [23, 22, 21, 20, 19, 7] denotes a 23-stage LFSR. The TAPS definition is a specific feedback tap sequence that generates an M-Sequence PRB. The LFSR stages can range from 2 to 64 bits. You cannot set the seed to zero; HSPICE reports an error and exits the simulation if you set the seed to zero.

Conventions for Feedback Tap Specification

You can express a given set of feedback connections in a convenient and easy-to-use shorthand form with the connection numbers listed within a pair of brackets. The g_0 connection is implied and not listed since it is always connected. Although g_m is also always connected, it is listed to convey the shift register size (number of registers).

The following line is a set of feedback taps where j is the total number of feedback taps (not including g_0), $f(1)=m$ is the highest-order feedback tap (and the size of the LFSR), and $f(j)$ are the remaining feedback taps:

[$f(1)$, $f(2)$, $f(3)$, ..., $f(j)$]

Example

The following line shows that the number of registers is 7 and the total number of feedback taps is 4:

```
[7, 3, 2, 1]
```

The following feedback input applies for this specification:

$$D(n) = [D(n-7) + D(n-3) + D(n-2) + D(n-1)] \text{ mod } 2$$

Example: Noise Generator Used for a Pulse or DC Level

The following example creates a DC source voltage with a random noise of 0.1v max amplitude in a spice stimulus.

```
* Generate transient noise to pulse or DC source
.options post=2 probe
* instantiate the noise generator
x_noise n_noise 0 tr_nsrc
r_load n_noise 0 1k
* Add noise to a clock signal
*v_clk clk 0 pulse (0 1.8 10n 10n 10n 90n 200n)
* or add noise to DC level...
v_clk clk 0 5
* Add the noise. Amplitude is .1v
E_jtr clk_jtr clk VCVS n_noise 0 0.1
.tran 1n 500n
.probe tran v(n_noise) v(clk) v(clk_jtr)
.subckt tr_nsrc np nn $ subcircuit generating transient noise
*****
*** To make the nature of output signal more random ****
*** generate 10 separate PRBS sources and sum them up ****
*****
v1 n1 nn LFSR (0 1 0p 5p 5p 2g 10001 [10,9,6,3,1] )
v2 n2 nn LFSR (0 1 0p 5p 5p 2g 5313 [10,9,6,3,1] )
v3 n3 nn LFSR (0 1 0p 5p 5p 2g 9 [10,9,6,3,1] )
v4 n4 nn LFSR (0 1 0p 5p 5p 2g 213 [10,9,6,3,1] )
v5 n5 nn LFSR (0 1 0p 5p 5p 2g 741 [10,9,6,3,1] )
v6 n6 nn LFSR (0 1 0p 5p 5p 2g 397 [10,9,6,3,1] )
v7 n7 nn LFSR (0 1 0p 5p 5p 2g 4793 [10,9,6,3,1] )
v8 n8 nn LFSR (0 1 0p 5p 5p 2g 7039 [10,9,6,3,1] )
v9 n9 nn LFSR (0 1 0p 5p 5p 2g 12031 [10,9,6,3,1] )
va na nn LFSR (0 1 0p 5p 5p 2g 50071 [10,9,6,3,1] )
*****
*** Sum up the 10 random pulse signals ***
*****
esum ns nn POLY(10) n1 nn n2 nn n3 nn n4 nn n5 nn n6 nn n7 nn n8
nn n9 nn na nn
+ 0 1 1 1 1 1 1 1 1 1 1
```

```
*** Filter out the DC component of the summed voltage ***
cs ns out 1p
rs out nn 1k
*** Insert two sets of RC filter to smooth the ***
*** edges and eliminate high frequency components ***
e_1 nout_f1 nn vcvs out nn 1
rf1 nout_f1 no_f1 2k
cf1 no_f1 nn .35p
e_2 nout_f2 nn vcvs no_f1 nn 1
rf2 nout_f2 np 2k
cf2 np nn .35p
.ends
.end
```

Voltage and Current Controlled Elements

HSPICE provides two voltage-controlled and two current-controlled elements, known as E-, G-, H-, and F-elements. You can use these controlled elements to model:

- MOS transistors
- Bipolar transistors
- Tunnel diodes
- SCRs
- Analog functions, such as:
 - operational amplifiers
 - summers
 - comparators
 - voltage-controlled oscillators
 - modulators
 - switched capacitor circuits

Depending on whether you used the polynomial or piecewise linear functions, the controlled elements can be:

- Linear functions of controlling-node voltages.
- Non-linear functions of controlling-node voltages.

Chapter 10: Sources and Stimuli

Voltage and Current Controlled Elements

- Linear functions of branch currents.
- Non-linear functions of branch currents.

The functions of the E, F, G, and H controlled elements are different.

- The E-element can be:
 - A voltage-controlled voltage source
 - A behavioral voltage source
 - An ideal op-amp.
 - An ideal transformer.
 - An ideal delay element.
 - A piecewise linear, voltage-controlled, multi-input AND, NAND, OR, or NOR gate.
- The F-element can be:
 - A current-controlled current source.
 - An ideal delay element.
 - A piecewise linear, current-controlled, multi-input AND, NAND, OR, or NOR gate.
- The G-element can be:
 - A voltage-controlled current source.
 - A behavioral current source.
 - A voltage-controlled resistor.
 - A piecewise linear, voltage-controlled capacitor.
 - An ideal delay element.
 - A piecewise linear, multi-input AND, NAND, OR, or NOR gate.
- The H-element can be:
 - A current-controlled voltage source.
 - An ideal delay element.
 - A piecewise linear, current-controlled, multi-input AND, NAND, OR, or NOR gate.

The next section describes polynomial and piecewise linear functions. Later sections describe element statements for linear or nonlinear functions. For full PWL examples, see [PWL/DATA/VEC Converter on page 459](#).

The following sections discuss these topics:

- [Polynomial Functions](#)
- [Piecewise Linear Function](#)

Polynomial Functions

You can use the controlled element statement to define the controlled output variable (current, resistance, or voltage), as a polynomial function of one or more voltages or branch currents. You can select several polynomial equations, by using the `POLY (NDIM)` parameter in the E, F, G, or H-element statement. Syntax can be either `POLY=INTEGER_NUMBER` or `POLY (INTEGER_NUMBER)`

For example, either of the following are legitimate statements for an E-element instance with the `POLY` function:

```
E1 e1 0 POLY=2 e11 0 e12 0 1 2 3
```

or

```
E1 e1 0 POLY(2) e11 0 e12 0 1 2 3
```

Polynomial values can be:

Value	Description
POLY(1)	One-dimensional equation (function of one controlling variable).
POLY(2)	Two-dimensional equation (function of two controlling variables).
POLY(3)	Three-dimensional equation (function of three controlling variables).
POLY(n)	Multi-dimensional equation (function of n controlling variables). HSPICE RF has a maximum allowable dimension limit of 3.

Each polynomial equation includes polynomial coefficient parameters (P0, P1 ... Pn), which you can set to explicitly define the equation.

One-Dimensional Function

If the function is one-dimensional (a function of one branch current or node voltage), the following expression determines the `FV` function value:

Chapter 10: Sources and Stimuli

Voltage and Current Controlled Elements

Equation 11

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

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

Note: If you specify one coefficient in a one-dimensional polynomial, HSPICE assumes that the coefficient is P1 (P0=0.0). Use this as input for linear controlled sources.

The following controlled source statement is a one-dimensional function. This voltage-controlled voltage source connects to nodes 5 and 0.

```
E1 5 0 POLY(1) 3 2 1 2.5
```

In the above source statement, the single-dimension polynomial function parameter, `POLY(1)`, informs HSPICE that E1 is a function of the difference of one nodal voltage pair. In this example, the voltage difference is between nodes 3 and 2, so $FA = V(3, 2)$.

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:

Equation 12

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

You can also express $V(5,0)$ as *E1*:

Equation 13

$$E1 = 1 + 2.5 \cdot V(3,2)$$

Two-Dimensional Function

If the function is two-dimensional (that is, a function of two node voltages or two branch currents), the following expression determines *FV*:

Equation 14

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

For a two-dimensional polynomial, the controlled source is a function of two nodal voltages or currents. To specify a two-dimensional polynomial, set POLY(2) in the controlled source statement.

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

Equation 15

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

or

Equation 16

$$E1 = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2$$

To implement this function, use this controlled-source element statement:

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

This example specifies a controlled voltage source, which connects between nodes 1 and 0. Two differential voltages control this voltage source:

- Voltage difference between nodes 3 and 2.
- Voltage difference between nodes 7 and 6.

That is, $FA=V(3,2)$, and $FB=V(7,6)$. The polynomial coefficients are:

- P0=0
- P1=3
- P2=0
- P3=0
- P4=0
- P5=4

Three-Dimensional Function

For a three-dimensional polynomial function, with FA , FB , and FC as its arguments, the following expression determines the FV function value:

Chapter 10: Sources and Stimuli
Voltage and Current Controlled Elements

Equation 17

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

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

Equation 18

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

or

Equation 19

$$E1 = 3 \cdot V(3,2) + 4 \cdot V(7,6)^2 + 5 \cdot V(9,8)^3$$

The resulting three-dimensional polynomial equation is:

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

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

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

$$P1 = 3$$

$$P7 = 4$$

$$P19 = 5$$

Substitute these values into the voltage controlled voltage source statement:

```
E1 1 0 POLY(3) 3 2 7 6 9 8 0 3 0 0 0 0 0 4 0 0 0 0 0 0
+ 0 0 0 0 0 5
```

The preceding example specifies a controlled voltage source, which connects between nodes 1 and 0. Three differential voltages control this voltage source:

- Voltage difference between nodes 3 and 2.
- Voltage difference between nodes 7 and 6.
- Voltage difference between nodes 9 and 8.

That is:

- $FA=V(3, 2)$
- $FB=V(7, 6)$
- $FC=V(9, 8)$

The statement defines the polynomial coefficients as:

- $P1=3$
- $P7=4$
- $P19=5$
- Other coefficients are zero.

N-Dimensional Function

Express an N-dimensional polynomial function as:

Equation 20

$$FV = p_0 + \sum_{j=1}^k (p_{1j}Fx_1 + p_{2j}Fx_2 + \dots + p_{kj}Fx_k)^j$$

where, Fx_1, Fx_2, \dots, Fx_k , represent the k independent controlling branch current, or nodal voltage, and $p_{ij}, i = 1, 2, \dots, k = 1, 2, \dots, n$ are the coefficients.

Piecewise Linear Function

You can use the one-dimensional piecewise linear (PWL) function to model special element characteristics, such as those of:

- Tunnel diodes
- Silicon-controlled rectifiers
- Diode breakdown regions

To describe the piecewise linear function, specify measured data points. Although data points describe the device characteristic, HSPICE automatically smooths the corners, to ensure derivative continuity. This, in turn, results in better convergence.

Chapter 10: Sources and Stimuli

Power Sources

The `DELTA` parameter controls the curvature of the characteristic at the corners. The smaller the `DELTA`, the sharper the corners are. The maximum `DELTA` limits to half of the smallest breakpoint distance. If the breakpoints have sufficient separation, specify the `DELTA` to a proper value.

- You can specify up to 100 point pairs.
- You must specify at least two point pairs (each point consists of an x and a y coefficient).

To model bidirectional switch or transfer gates, G-elements use the `NPWL()` and `PPWL()` functions, which behave the same way as NMOS and PMOS transistors.

You can also use the piecewise linear function to model multi-input AND, NAND, OR, and NOR gates. In this usage, only one input determines the state of the output.

- In AND and NAND gates, the input with the smallest value determines the corresponding output of the gates.
- In OR and NOR gates, the input with the largest value determines the corresponding output of the gates.

Power Sources

These sections describe independent power sources and controlled power sources.

The following sections discuss these topics:

- [Independent Sources](#)
- [Controlled Sources](#)

Independent Sources

A power source is a special kind of voltage or current source that supplies the network with a pre-defined power which varies by time or frequency. The source produces a specific input impedance.

To apply a power source to a network, you can use either:

- A Norton-equivalent circuit (if you specify this circuit and a current source)—the I (current source) element, or
- A Thevenin-equivalent circuit (if you specify this circuit and a voltage source)—the V (voltage source) element.

As with other independent sources, an HSPICE simulation assumes that positive current flows from the positive node through the source to the negative node. A power source is a time-variant or frequency-dependent utility source; therefore, the value/phase can be a function of either time or frequency.

A power source is a subclass of the independent voltage/current source, with some additional keywords or parameters:

You can use I and V elements in DC, AC, and transient analysis. The I and V elements can be data-driven. Supported power source formats include:

- DC
- SIN, a damped sinusoidal function.

The following sections discuss these topics:

- [Using the Keyword POWER](#)
- [Calculation for Total Dissipated Power and for Voltage Source Power](#)
- [Subcircuit Power Calculation](#)

Using the Keyword POWER

If you use the POWER keyword in the netlist, then a simulation recognizes a current/voltage source as a power source:

```
Vxxx n+ n- [[DC=] dcval tranfun [AC=acmag acphase]]
+ power=[0|1|2|w|dbm]
Ixxx n+ n- [[DC=] dcval tranfun [AC=acmag acphase]]
+ power=[0|1|2|w|dbm] [M=val]
```

Parameter	Description
DC=dcval	DC source keyword and value in volts. Used for the operating point calculation for all simulations except transient. Transient analysis calculates an additional operating point with the tranfun value at time zero. Default=0.0.

Parameter	Description
tranfun	Transient source function (When POWER is on, transient source function only supports 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 or in watts or dbms when the power switch is turned on.
acphase	Phase of the AC source, in degrees. Default=0.0.
power=[0 1 2 W dbm]	Power switch <ul style="list-style-type: none"> ▪ When 0 (default), element treated as a voltage or current source. ▪ When 1 or W, the element is treated as a power source, and realized as a voltage source with a series impedance or a current source with a parallel impedance. In this case, the source value is interpreted as RMS available power in units of Watts. ▪ When 2 or dbm, the element is treated as a voltage source with a series impedance or a current source with a parallel impedance. Values are in dbms. You can use this parameter for transient analysis if the power source is either DC or SIN. When POWER is on, a 50 ohm source impedance is added automatically.

Example 1

This example applies a 5 W power source to node 10 and node 20, in a Thevenin-equivalent manner. The impedance of this power source is 50 ohms.

```
V11 10 20 power=1 dc=5
```

Example 2

This example applies a 20 dbm power source to node 1 and to ground, in a Norton-equivalent manner. The source impedance is 50 ohms.

```
Iname 1 0 power=2 dc=20
```

Example 3

This example applies a 1 dbm (available) AC power source to node 1 and ground, in a Thevenin-equivalent manner. The impedance of this power source is 50 ohms.

```
V11 1 0 power=dbm ac=1
```


Example 4

This example applies a 1w (available) sinusoidal power source to node 1 and 0, in a Norton-equivalent manner. The source impedance is 50 ohms

```
Iname 1 0 power=w sin(0 1 1k)
```

Calculation for Total Dissipated Power and for Voltage Source Power

You can use two keywords to calculate power:

- The `POWER` keyword calculates the total dissipated power of the circuit.
- The `SRC_PWR` keyword calculates the total power from the voltage source of the circuit.

For example, if you have a top level circuit as follows:

```
*****
vin n1 pulse(0 5 0 1n 1n 4n 10n)
r0 n1 n2 1
r1 n2 0 1
c0 n2 0 .2u
*****
```

...you can use `.MEAS TRAN POWER` to calculate the total dissipated power in resistors. The total dissipated power is $p(r0) + p(r1)$.

You can use `.MEAS TRAN SRC_PWR` to calculate the total power from the voltage source. The total power from the voltage source is $p(r0) + p(r1) + p(c0)$. HSPICE can verify this with the following statement:

```
.print tran tot_pwr=par('p(r0) + p(r1) + p(c0)')
```

Note that HSPICE does not add the power of each independent source (V & I).

For power measurements, HSPICE and HSPICE RF compute the dissipated or stored power in each passive element (R, L, C), and source (V, I, G, E, F, and H). To compute this power, HSPICE and HSPICE RF multiply the voltage across an element and the corresponding branch current. However, for semiconductor devices, HSPICE and HSPICERF measure only the dissipated power, excluding the power stored in the device junction or parasitic capacitances from the device power computation. The `avg(P(Vdd))` function measures both the dissipated power and the stored power while `avg(P(X))` measures only the dissipated power for semiconductor devices. If the circuit

contains semiconductor devices, the results of the $\text{avg}(P(X))$ function can be erroneous. The recommended method to measure the average power of the entire circuit in this case is to measure the average power of the source by using the $\text{avg}(P(V_{dd}))$ function.

Subcircuit Power Calculation

To print or probe an element or subcircuit power, use the variables, $P(\text{element_name})$ or $P(\text{instance_name_of_subckt})$. For example, if you have the following subcircuit:

```
x1 in out inv
.subckt inv in out
    mn out in 0 0 nch W=10u L=1u
    mp out in vdd vdd pch W=10u L=1u
.ends
```

The total circuit POWER is $-(p(v_{in}) + p(v_{vdd}))$, which is equal to $p(r_{out}) + p(c_{out}) + p(x1)$. You can verify this by using the parameter expression.

Measuring Leakage Power

When you probe $P(\text{Instance_name})$, beginning with HSPICE version A-2007.12, HSPICE includes the gate tunneling current in the power function calculations. In prior releases, the power calculation function in HSPICE did not include gate tunneling induced power dissipation.

Troubleshooting Differences in Rise/Fall Power Input Signals and Power Measurements

You may note a difference between the AVG power calculations for the rise time and fall time for input signals. There should not be a difference, but if you measure the AVG subcircuit power by using $P(\text{Instance_name})$ for an inverter subcircuit, for example, you may note that HSPICE excludes the energy stored in the output and includes the energy discharged from the capacitor for fall time.

If there is any difference in the measured results, then try running the simulation with the most accurate settings by setting `.OPTION RUNLVL=6`. In addition, you can set `.OPTION DELMAX` to the minimum rise or fall time of your circuit.

You may observe a difference in results of two measurements when using the $\text{avg}(P(X))$ function, where X contains semiconductor devices, and using $\text{avg}(P(V_{dd}))$, the power of the voltage source.

For power measurements, HSPICE and HSPICE RF computes the dissipated or stored power in each passive element (R, L, C), and source (V, I, G, E, F, and H). To compute this power, HSPICE and HSPICE RF multiplies the voltage across an element, and the corresponding branch current. However, for semiconductor devices, HSPICE and HSPICE RF measures only the dissipated power and excludes the power stored in the device junction or parasitic capacitances from the device power computation. The `avg(P(Vdd))` measures both the dissipated power and the stored power while `avg(P(X))` measures only the dissipated power for semiconductor devices. If the circuit contains semiconductor devices, the results of the `avg(P(X))` function can be erroneous. The recommended method to measure the average power of the entire circuit in this case is to measure the average power of the source by using the `(avg P(Vdd))` function.

Controlled Sources

In addition to independent power sources, you can also create four types of controlled sources:

- Voltage-controlled voltage source (VCVS), or E-element
- Current-controlled current source (CCCS), or F-element
- Voltage-controlled current source (VCCS), or G-element
- Current-controlled voltage source (CCVS), or H-element

Voltage-Dependent Voltage Sources — E-elements

This section explains E-element syntax statements, and defines their parameters.

```
Exxx n+ n- [VCVS|LEVEL=0] in+ in- ...
```

- `LEVEL=0` is an interchangeable function keyword, such as `VCAP` or `VCCS`, etc.
- `LEVEL=1` is an OpAmp.
- `LEVEL=2` is a Transformer.

HSPICE and HSPICE RF support both support E-elements (and G-elements).

The E-element specifies a voltage noise source with the noise value in $V^{2/Hz}$.

Chapter 10: Sources and Stimuli

Voltage-Dependent Voltage Sources — E-elements

The noise source can be bias and/or frequency dependent. For a white noise source, use a constant value for 'expression'.

For noise sources using expressions, the E-element takes the form:

```
ename n1 n2 noise='expression'
```

See also:

- [Using G- and E-elements](#) in the *HSPICE User Guide: Advanced Analog Simulation and Analysis*.
- [Filters Examples](#) (`bandstop1.sp`) for a demonstration of the E-element as a filter.

Voltage-Controlled Voltage Source (VCVS)

The following sections discuss these topics:

- [E-element Parameters](#)
- [Linear](#)
- [Polynomial \(POLY\)](#)
- [Piecewise Linear \(PWL\)](#)
- [Multi-Input Gates](#)
- [Delay Element](#)
- [Laplace Transform](#)
- [Pole-Zero Function](#)
- [Frequency Response Table](#)
- [Foster Pole-Residue Form](#)
- [Behavioral Voltage Source \(Noise Model\)](#)
- [Ideal Op-Amp](#)
- [Ideal Transformer](#)

E-element Parameters

The following list describes the E-element parameters.

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
DELAY	Keyword for the delay element. Same as for the voltage-controlled voltage source, except it has an associated propagation delay, TD. This element adjusts propagation delay in macro (subcircuit) modeling. DELAY is a reserved word; do not use it as a node name.
DELTA	Controls the curvature of the piecewise linear corners. This parameter defaults to one-fourth of the smallest distance between breakpoints. The maximum is one-half of the smallest distance between breakpoints.
Exxx	Voltage-controlled element name. Must begin with E, followed by up to 1023 alphanumeric and special characters.
gain	Voltage gain.
gatetype(k)	Can be AND, NAND, OR, or NOR. k represents the number of inputs of the gate. x and y represent the piecewise linear variation of output, as a function of input. In multi-input gates, only one input determines the state of the output.
IC	Initial condition: initial estimate of controlling voltage value(s). If you do not specify IC, default=0.0.
in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
k	Ideal transformer turn ratio: $V(\text{in+},\text{in-}) = k \cdot V(\text{n+},\text{n-})$ 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.

Chapter 10: Sources and Stimuli

Voltage-Dependent Voltage Sources — E-elements

Parameter	Description
NDIM	Number of polynomial dimensions. If you do not set POLY(NDIM), HSPICE assumes a one-dimensional polynomial. NDIM must be a positive number.
NPDELAY	Sets the number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is, $NPDELAY_{default} = \max\left[\frac{\min\langle TD, tstop \rangle}{tstep}, 10\right]$ The .TRAN statement specifies tstep and tstop values.
LEVEL=0/1/2/3	Function keyword such as VCCS, VCAP, etc. <ul style="list-style-type: none">▪ LEVEL=0: Voltage-Controlled Current Source (VCCS).▪ LEVEL=1: Voltage-Controlled Resistor (VCR).▪ LEVEL=2: Voltage-Controlled Capacitor (VCCAP), Negative Piece-Wise Linear (NPWL).▪ LEVEL=3: VCCAP, Positive Piece-Wise Linear (PPWL).
OPAMP or Level=1	The keyword for an ideal op-amp element. OPAMP is a HSPICE reserved word; do not use it as a node name.
P0, P1 ...	The polynomial coefficients. If you specify one coefficient, HSPICE assumes that it is P1 (P0=0.0), and that the element is linear. If you specify more than one polynomial coefficient, the element is nonlinear, and P0, P1, P2 ... represent them (see Polynomial Functions on page 275).
POLY	Keyword for the polynomial function. If you do not specify POLY(ndim), HSPICE assumes a one-dimensional polynomial. Ndim must be a positive number.
PWL()	Keyword for the piecewise linear function; must include parenthetical ().
SCALE	Multiplier for the element value.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $SCALE_{eff} = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Keyword for the time (propagation) delay.

Parameter	Description
TRANSFORMER or LEVEL=2	Keyword for an ideal transformer. TRANSFORMER is a reserved word; do not use it as a node name.
VOL	Voltage output that flows from $n+$ to $n-$. The expression that you define can be a function of: <ul style="list-style-type: none"> ▪ node voltages ▪ branch currents ▪ time (time variable) ▪ temperature (temper variable) ▪ frequency (hertz variable)
VCVS	Keyword for a voltage-controlled voltage source. VCVS is a reserved word; do not use it as a node name. The VCVS keyword is optional for E-elements and not required for LAPLACE, FREQ and FOSTER forms.
x1,...	Controlling voltage across the $in+$ and $in-$ nodes. The x values must be in increasing order.
y1,...	Corresponding element values of x .
LAPLACE	Laplace transform resolves a function into its moments (see Laplace Transform on page 290)
FREQ	Response table (see Frequency Response Table on page 292)
FOSTER	Pole residue form (see Foster Pole-Residue Form on page 294)

For a listing of output templates for current-controlled current source E-elements models (HSPICE only), see [Table 37 on page 440](#).

Linear

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

For a description of these parameters, see [E-element Parameters on page 287](#).

Polynomial (POLY)

```
Exxx n+ n- [VCVS] POLY(NDIM) in1+ in1- ...
+ inndim+ inndim-TC1=val [TC2=val] [SCALE=val]
+ [MAX=val] [MIN=val] [ABS=1] p0 p1... [IC=val]
```

In this syntax, *dim (dimensions)* ≤ 3 . HSPICE has no limitation for dimensions while the HSPICE RF maximum dimensions are 3; HSPICE reports an error message, otherwise. For a description of these parameters, see [E-element Parameters on page 287](#). For a description of possible POLY syntaxes, see [Polynomial Functions on page 275](#).

Piecewise Linear (PWL)

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

For a description of these parameters, see [E-element Parameters on page 287](#).

Multi-Input Gates

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

In this syntax, *gatetype(k)* can be AND, NAND, OR, or NOR gates. For a description of these parameters, see [E-element Parameters on page 287](#).

Delay Element

```
Exxx n+ n- [VCVS] DELAY in+ in- TD=val [SCALE=val]
+ [TC1=val] [TC2=val] [NPDELAY=val]
```

For a description of these parameters, see [E-element Parameters on page 287](#).

Laplace Transform

Voltage Gain H(s):

```
Exxx n+ n- LAPLACE in+ in- k0, k1, ..., kn / d0, d1, ..., dm
+ [SCALE=val] [TC1=val] [TC2=val]
```

For a description of these parameters, see [E-element Parameters on page 287](#).

Transconductance H(s):

```
Gxxx n+ n- LAPLACE in+ in- k0, k1, ..., kn / d0, d1, ..., dm
+ [SCALE=val] [TC1=val] [TC2=val] [M=val]
```

H(s) is a rational function, in the following form:

Equation 21

$$H(s) = \frac{k_0 + k_1s + \dots + k_ns^n}{d_0 + d_1s + \dots + d_ms^m}$$

You can use parameters to define the values of all coefficients ($k_0, k_1, \dots, d_0, d_1, \dots$).

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

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:

Equation 22

$$H(s) = \frac{1}{1 + 2s + 2s^2 + s^3}$$

The Ehipass element statement describes a third-order high-pass filter, with the transfer function:

Equation 23

$$H(s) = \frac{s^3}{1 + 2s + 2s^2 + s^3}$$

For full demonstration netlists using E-elements for Laplace transforms use the paths shown as follows in [Filters Examples on page 995](#):

- lowloss.sp (RL line model using Laplace behavioral element)
- rcline.sp (RC line model using Laplace elements)
- ninth.sp (active low pass filter using Laplace elements)

Pole-Zero Function

Voltage Gain H(s):

```
Exxx n+ n- POLE in+ in- a az1, fz1, ..., azn, fzn / b,
+ ap1, fp1, ..., apm, fpm [SCALE=val] [TC1=val]
+ [TC2=val]
```

For a description of these parameters, see [E-element Parameters on page 287](#).

Transconductance H(s):

Gxxx n+ n- POLE in+ in- a az1, fz1, ..., azn, fzn / b,
+ ap1, fp1, ..., apm, fpm [SCALE=val] [TC1=val]
+ [TC2=val] [M=val]

The following equation defines H(s) in terms of poles and zeros:

Equation 24

$$H(s) = \frac{a \cdot (s + \alpha_{z1} - j2\pi f_{z1}) \dots (s + \alpha_{zn} - j2\pi f_{zn})(s + \alpha_{zn} + j2\pi f_{zn})}{b \cdot (s + \alpha_{p1} - j2\pi f_{p1}) \dots (s + \alpha_{pm} - j2\pi f_{pm})(s + \alpha_{pm} + j2\pi f_{pm})}$$

The complex poles or zeros are in conjugate pairs. The element description specifies only one of them, and the program includes the conjugate. You can use parameters to specify the a, b, α , and f values.

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Example

```
Ghigh_pass 0 out POLE in 0 1.0 0.0,0.0 / 1.0 0.001,0.0
Elow_pass out 0 POLE in 0 1.0 / 1.0, 1.0,0.0 0.5,0.1379
```

The Ghigh_pass statement describes a high-pass filter, with the transfer function:

Equation 25

$$H(s) = \frac{1.0 \cdot (s + 0.0 + j \cdot 0.0)}{1.0 \cdot (s + 0.001 + j \cdot 0.0)}$$

The Elow_pass statement describes a low-pass filter, with the transfer function:

Equation 26

$$H(s) = \frac{1.0}{1.0 \cdot (s + 1)(s + 0.5 + j2\pi \cdot 0.1379)(s + 0.5 - (j2\pi \cdot 0.1379))}$$

For a full demonstration netlist of the G- and E-elements used in a behavioral model use the path shown for low_pass.sp in [Filters Examples on page 995](#).

Frequency Response Table

Voltage Gain H(s):

Exxx n+ n- FREQ in+ in- f1, a1, f1, ..., fi, ai, f1

```
+ [DELF=val] [MAXF=val] [SCALE=val] [TC1=val]
+ [TC2=val] [LEVEL=val] [ACCURACY=val]
```

For a description of these parameters, see [E-element Parameters on page 287](#)

Transconductance H(s):

```
Gxxx n+ n- FREQ in+ in- f1, a1, f1, ..., fi, ai, f1
+ [DELF=val] [MAXF=val] [SCALE=val] [TC1=val]
+ [TC2=val] [M=val] [LEVEL=val] [ACCURACY=val]
```

Where,

- Each f_i is a frequency point, in hertz.
- a_i is the magnitude, in dB.
- f_1 is the phase, in degrees.

At each frequency, HSPICE uses interpolation to calculate the network response, magnitude, and phase. HSPICE interpolates the magnitude (in dB) logarithmically, as a function of frequency. It also interpolates the phase (in degrees) linearly, as a function of frequency.

Equation 27

$$|H(j2\pi f)| = \left(\frac{a_i - a_k}{\log f_i - \log f_k} \right) (\log f - \log f_i) + a_i$$

Equation 28

$$\angle H(j2\pi f) = \left(\frac{\phi_i - \phi_k}{f_i - f_k} \right) (f - f_i) + \phi_i$$

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Example

```
Eftable output    0 FREQ input    0
+ 1.0k    -3.97m    293.7
+ 2.0k    -2.00m    211.0
+ 3.0k    17.80m    82.45
+ .....
+ 10.0k   -53.20    -1125.5
```

- The first column is frequency, in hertz.
- The second column is magnitude, in dB.
- The third column is phase, in degrees.

Set the LEVEL to 1 for a high-pass filter.

Set the last frequency point to the highest frequency response value that is a real number, with zero phase.

You can use parameters to set the frequency, magnitude, and phase, in the table.

For a full demonstration file showing a behavioral model using a G- table element see the path to the demo file `phaseshift.sp` in [Filters Examples on page 995](#).

Foster Pole-Residue Form

Gain E(s) form

```
Exxx n+ n- FOSTER in+ in- k0 k1
+ (Re{A1}, Im{A1}) / (Re{p1}, Im{p1})
+ (Re{A2}, Im{A2}) / (Re{p2}, Im{p2})
+ (Re{A3}, Im{A3}) / (Re{p3}, Im{p3})
+ ...
```

For a description of these parameters, see [E-element Parameters on page 287](#).

Transconductance G(s) form

```
Gxxx n+ n- FOSTER in+ in- k0 k1
+ (Re{A1}, Im{A1}) / (Re{p1}, Im{p1})
+ (Re{A2}, Im{A2}) / (Re{p2}, Im{p2})
+ (Re{A3}, Im{A3}) / (Re{p3}, Im{p3})
+ ...
```

In the above syntax, parenthesis, commas, and slashes are separators—they have the same meaning as a space. Four numbers represent a pole-residue pair (real and imaginary part of the residue, then real and imaginary part of the pole).

You must make sure that $\text{Re}[p_i] < 0$; otherwise, the simulations diverge. Also, assure passivity of the model (for an N-port admittance matrix Y, $\text{Re}\{Y\}$ should be positive-definite), or the simulation is likely to diverge.

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Example

To represent a G(s) in the form,

Equation 29

$$G(s) = 0.001 + 1 \times 10^{-12} s + \frac{0.0008}{s + 1 \times 10^{10}} + \frac{(0.001 - j0.006)}{s - (-1 \times 10^8 + j1.8 \times 10^{10})} + \frac{(0.001 + j0.006)}{s - (-1 \times 10^8 - j1.8 \times 10^{10})}$$

You would input:

```
G1 1 0 FOSTER 2 0 0.001 1e-12
+(0.0004, 0)/(-1e10, 0) (0.001, -0.006)/(-1e8, 1.8e10)
```

Note: For real poles, enter half the residue value because the netlist applies it twice. In the above example, the first pole-residue pair is real, you write it as “A1/(s-p1)+A1/(s-p1)”; therefore, enter 0.0004 rather than 0.0008.

Behavioral Voltage Source (Noise Model)

You can implement a behavioral voltage noise source with an E-element. As noise elements, these are two-terminal elements that represent a noise source connected between two specified nodes.

```
Exxx n+ n- noise='expression'
```

Where,

Exxx is the voltage-controlled element name, which must begin with “E”, followed by up to 1023 alphanumeric and special characters.

n+ is the positive node.

n- is the negative node.

noise='expression' can contain the bias, frequency, or other parameters.

Data form

```
Exxx n+ n- noise data=dataname
.DATA dataname
+ pname1 pname2
+ freq1 noise1
+ freq2 noise2
+ ...
.enddata
```

The data form defines a basic frequency-noise table. The .DATA statement contains 2 parameters: frequency and noise to specify the noise value at each frequency point. The unit for frequency is hertz, and the unit for noise is

Chapter 10: Sources and Stimuli

Voltage-Dependent Voltage Sources — E-elements

decibels relative to the carrier (dBc/Hz). Noise files report the same units. See also, [Using Noise Analysis Results as Input Noise Sources on page 338](#).

HSPICE supports the following syntaxes:

- `Exxx n1 n2 noise data=dataname`
- `Exxx n1 n2 noise data=atablock`
- `Exxx n1 n2 noisefile='filename'`
- `Exxx n1 n2 noise='expression'`
- `Exxx n1 n2 noise='Table(arg1,f1,v1,f2,v2...)'`
- `Exxx n1 n2 noise='Table(arg1,dotDataBlockName)'`, where `dotDataBlockName` is the `.data` statement reference

Ideal Op-Amp

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

You can also substitute `LEVEL=1` in place of `OPAMP`:

```
Exxx n+ n- in+ in- level=1
```

For a description of these parameters, see [E-element Parameters](#).

Ideal Transformer

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

You can also substitute `LEVEL=2` in place of `TRANSFORMER`:

```
Exxx n+ n- in+ in- level=2 k
```

For a description of these parameters, see [E-element Parameters](#).

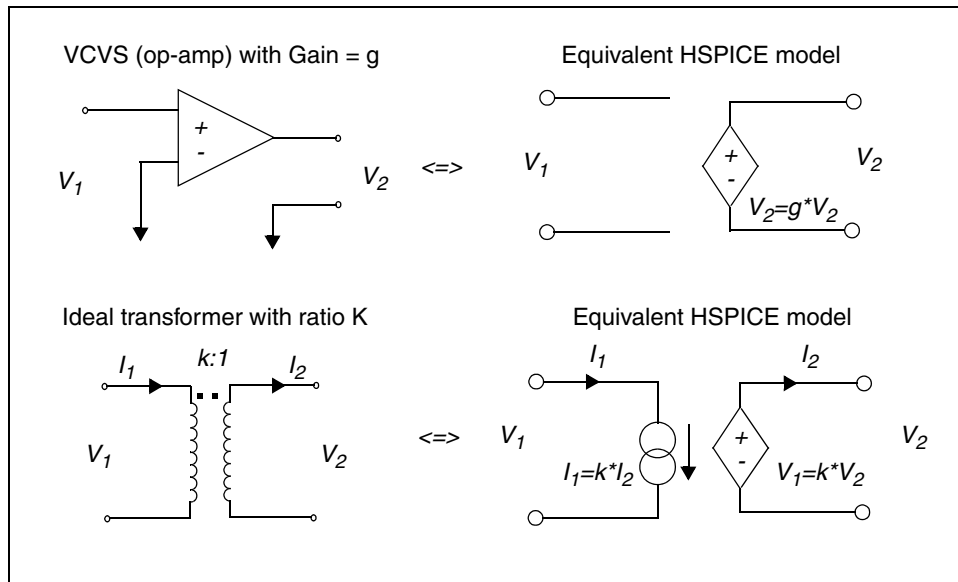


Figure 31 Equivalent VCVS and Ideal Transformer HSPICE Models

E-element Examples

For full demonstration examples of circuits using E-elements see the paths to the following netlist files in the section [Behavioral Application Examples](#):

- `behave.sp` (AND/NAND gates)
- `compar.sp` (behavioral comparator with hysteresis)
- `pwl7.sp` (modeling inverter by using a PWL VCVS)
- `sampling.sp` (Sampling a sine wave using an expression)
- `swcap5.sp` (Fifth-order elliptic switched capacitor filter—OPAMP(LEVEL=1))
- `idealop.sp` (Ninth-order low-pass filter, OPAMP)
- `integ.sp` (Integrator circuit, GAIN)
- `pll_bvp.sp` (PLL build with behavioral source, PWL)

The following sections provide these examples:

- [Ideal OpAmp](#)
- [Voltage Summer](#)

Chapter 10: Sources and Stimuli

Voltage-Dependent Voltage Sources — E-elements

- [Polynomial Function](#)
- [Zero-Delay Inverter Gate](#)
- [Delayed and Inverted Signal](#)
- [Differential Amplifiers and Opamp Signals](#)
- [Ideal Transformer](#)
- [Voltage-Controlled Oscillator \(VCO\)](#)
- [Switching between Two Voltage Sources Connected to the Same Node](#)

Ideal OpAmp

You can use the voltage-controlled voltage source to build a voltage amplifier, with supply limits.

- The output voltage across nodes 2,3 is $v(14,1) * 2$.
- The value of the voltage gain parameter is 2.
- The `MAX` parameter sets a maximum E1 voltage of 5 V.
- The `MIN` parameter sets a minimum E1 voltage output of -5 V.

Example

If $V(14,1)=-4V$, then HSPICE sets E1 to -5V, and not -8V as the equation suggests.

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

To specify a value for polynomial coefficient parameters, use the following format:

```
.PARAM CU=2.0  
E1 2 3 14 1 MAX=+5 MIN=-5 CU
```

For a full demonstration netlist of the E-element used for an active low pass filter using behavioral opamp models, use the path shown for `low_pass9a.sp` in [Filters Examples on page 995](#).

Voltage Summer

An ideal voltage summer specifies the source voltage, as a function of three controlling voltage(s):

- V(13,0)
- V(15,0)
- V(17,0)

To describe a voltage source, the voltage summer uses this value:

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

This example represents an ideal voltage summer. It initializes the three controlling voltages for a DC operating point analysis, to 1.5, 2.0, and 17.25 V.

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

Polynomial Function

A voltage-controlled source can also output a non-linear function, using a one-dimensional polynomial. This example does not specify the `POLY` parameter, so HSPICE assumes it is a one-dimensional polynomial—that is, a function of one controlling voltage. The equation corresponds to the element syntax. Behavioral equations replace this older method.

```
V (3,4)=10.5 + 2.1 *V(21,17) + 1.75 *V(21,17)2"  
E2 3 4 POLY 21 17 10.5 2.1 1.75  
E2 3 4 VOLT="10.5 + 2.1 *V(21,17) + 1.75 *V(21,17)2"  
E2 3 4 POLY 21 17 10.5 2.1 1.75
```

Zero-Delay Inverter Gate

Use a piecewise linear transfer function to build a simple inverter, with no delay.

```
Einv out 0 PWL(1) in 0 .7v,5v 1v,0v
```

Delayed and Inverted Signal

You can use an E-element to invert a signal to generate a delayed signal which is the inverse of another signal. Use the following syntax to generate a delayed signal (`TD`) and is the inverse of the input signal (`in`).

Chapter 10: Sources and Stimuli

Voltage-Dependent Voltage Sources — E-elements

```
.option list node post
Vin in 0 pwl(0 0 10n 0 13n 2v 23n 2v 24n 0v)    $$ input signal
      to be inverted and delayed
Edelay in_delay 0 DELAY in 0 TD=2n              $$ signal
      "in_delay" is the "in" signal delayed by TD
Edelay_inv out_inv 0 PWL(1) in_delay 0 0v 2v 2v 0v  $$ signal
      "out_inv" is the inverted and delayed "in" signal
.tran .1n 30n
.print v(in_delay) v(out_inv) v(in)
.end
```

Differential Amplifiers and Opamp Signals

E-elements define a differential voltage source to use with differential amplifiers or opamps. E-elements can also provide differential signals to drive circuits requiring differential signals.

```
*****spice definition*****
VID 7 0 DC 0 ac=1 SIN (0 0.70 1MEG)
E+ in+ 10 7 0 0.5  $ differential signal 1, level can be varied
by changing gain
E- in- 10 7 0 -0.5  $ differential signal 2
VIC 10 0 DC 0.3V  $VIC is the common mode signal source
```

For a full demonstration example of AND/NAND gates and use of the E-element see the path to the `behave.sp` netlist noted in the section [Behavioral Application Examples](#).

For a full netlist differential block analysis example, see also:
`$installdir/demo/hspice/behave/diff.sp`.

Ideal Transformer

If the turn ratio is 10 to 1, the voltage relationship is $V(\text{out})=V(\text{in})/10$.

```
Etrans out 0 TRANSFORMER in 0 10
```

You can also substitute `LEVEL=2` in place of `TRANSFORMER`:

```
Etrans out 0 in 0 level=2 10
```

Voltage-Controlled Oscillator (VCO)

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

Example 1

In this example, the voltage at the control node controls the frequency of the sinusoidal output voltage at the out node. `v0` is the DC offset voltage, and gain

is the amplitude. The *freq · control* portion of the equation specifies the frequency of the output, which is a sinusoidal voltage.

```
Evco out 0 VOL='v0+gain*SIN(6.28 freq*v(control)*TIME)'
```

Note: This equation is valid only for a steady-state VCO (fixed voltage).
If you sweep the control voltage, this equation does not apply.

Example 2

In this example, a Verilog-A module controls VCO output by tracking phase to ensure continuity.

```
`include "disciplines.vams"

module vco(vin, vout);
  inout vin, vout;
  electrical vin, vout;
  parameter real amp = 1.0;
  parameter real offset = 1.0;
  parameter real center_freq = 1G;
  parameter real vco_gain = 1G;
  real phase;

  analog begin
    phase = idt(center_freq + vco_gain*V(vin), 0.0);
    V(vout) <+ offset+amp*sin(6.2831853*phase);
  end
endmodule
```

Example 3

This example is a controlled-source equivalent of the Verilog-A module shown in the previous example. Like the previous example, it establishes a continuous phase and therefore, a continuous output voltage.

```
.subckt vco in out amp=1 offset=1 center_freq=1 vco_gain=1
.ic v(phase)=0
cphase phase 0 1e-10
g1 0 phase cur='1e-10*(center_freq+vco_gain*v(in))'
eout out 0 vol='offset+amp*sin(6.2831853*v(phase))'
.ends
```

Example 4

In this example, controlled-sources control VCO output.

```
.param pi=3.1415926
.param twopi='2*pi'
.subckt vco in inb out outb f0=100k kf=50k out_off=0.0 out_amp=1.0
```

```
gs 0 s poly(2) c 0 in inb 0 'twopi*1e-9*f0' 0 0 'twopi*1e-9*kf'
gc c 0 poly(2) s 0 in inb 0 'twopi*1e-9*f0' 0 0 'twopi*1e-9*kf'
cs s 0 1e-9 ic=0
cc c 0 1e-9 ic=1
eout out 0 vol='out_off+(out_amp*v(s))'
eoutb outb 0 vol='out_off+(out_amp*v(c))'
.ic v(c)=1 v(s)=0
.ends
```

Switching between Two Voltage Sources Connected to the Same Node

You can use the HSPICE voltage controlled voltage source (E-element) to design a behavioral switch by creating a netlist as follows:

```
.option post
Vin1 in1 0 pwl ...          $ source 1
Vin2 in2 0 pwl ...          $ source 2
Ein in 0 vol='v(in1)*v(ctrl) + v(in2)*(1-v(ctrl))'
                                     $ behavioral switch
vctrl ctrl 0 pwl 0 1 49n 1 50n 0    $ control voltage
.tran 1n 100n
.end
```

In this example, the $v(ctrl)$ value is 1 initially so that $v(in) = v(in1)$. At 50nS, the control voltage changes to 0 and $v(in) = v(in2)$.

Another means to achieve this is to design a Verilog-A module as a behavioral switch.

Using the E-element for AC Analysis

The following equation describes an E-element:

```
E1 ee 0 vol=f(v(1), v(2))
```

In an AC analysis computes voltage as follows:

$$v(ee) = A * \delta_v1 + B * \delta_v2$$

Where,

- A is the derivative of $f(v(1), v(2))$ to $v(1)$ at the operating point
- B is the derivative of $f(v(1), v(2))$ to $v(2)$ at the operating point
- δ_v1 is the AC voltage variation of $v(1)$
- δ_v2 is the AC voltage variation of $v(2)$

Example

This example uses demonstration netlist `eelm.sp`, which is available in directory `$installdir/demo/hspice/sources`:

```
*****
***** E element for AC analysis
.option post
.op
*CASE1-Mixed and zero time unit has zero value(tran)
v_n1 n1 gnd dc=6.0 pw1 0.0 6.0 1.0n 6.0 ac 5.0
v_n2 n2 gnd dc=4.0 pw1 0.0 4.0 1.0n 6.0 ac 2.0
e1 n3 gnd vol='v(n1)+v(n2) '
e2 n4 gnd vol='v(n1)*v(n2) '
r1 n1 gnd 1
r2 n2 gnd 1
r3 n3 gnd 1
r4 n4 gnd 1
.tran 10p 3n
.ac dec 1 1 100meg
.print ac v(n?)
.end
*****
```

The AC voltage of node n3 is:

$$\begin{aligned} v(n3) &= 1.0 * v(n1) (ac) + 1.0 * v(n2) (ac) \\ &= 1.0 * 5.0 + 1.0 * 2.0 \\ &= 7.0 (v) \end{aligned}$$

The AC voltage of node n4 is:

$$\begin{aligned} v(n4) &= v(n2) (op) * v(n1) (ac) + v(n1) (op) * v(n2) (ac) \\ &= 4.0 * 5.0 + 6.0 * 2.0 \\ &= 32.0 (v) \end{aligned}$$

Current-Dependent Current Sources — F-elements

This section explains the F-element syntax and parameters.

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

The following section introduces this topic:

- [Current-Controlled Current Source \(CCCS\) Syntax](#)

Current-Controlled Current Source (CCCS) Syntax

The following sections provide these syntax examples:

- [Linear](#)
- [Polynomial \(POLY\)](#)
- [Piecewise Linear \(PWL\)](#)
- [Multi-Input Gates](#)
- [Delay Element](#)
- [F-element Parameters](#)
- [F-element Examples](#)

Linear

```
Fxxx n+ n- [CCCS] vn1 gain [MAX=val] [MIN=val] [SCALE=val]  
+ [TC1=val] [TC2=val] [M=val] [ABS=1] [IC=val]
```

Polynomial (POLY)

```
Fxxx n+ n- [CCCS] POLY(ndim) vn1 [... vnndim] [MAX=val]  
+ [MIN=val] [TC1=val] [TC2=val] [SCALE=val] [M=val]  
+ [ABS=1] p0 [p1...] [IC=val]
```

In this syntax, *dim (dimensions)* ≤ 3.

Piecewise Linear (PWL)

```
Fxxx n+ n- [CCCS] PWL(1) vn1 [DELTA=val] [SCALE=val]  
+ [TC1=val] [TC2=val] [M=val] x1,y1 ... x100,y100 [IC=val]
```

For a full demo file example, go to [Behavioral Application Examples](#) and take the path to the `pw18.sp` file (smoothing the triangle waveform by using the PWL CCCS).

Multi-Input Gates

```
Fxxx n+ n- [CCCS] f(k) vn1, ... vnk [DELTA=val]
```

```
+ [SCALE=val] [TC1=val] [TC2=val] [M=val] [ABS=1]
+ x1,y1 ... x100,y100 [IC=val]
```

In this syntax, `gatetype(k)` can be AND, NAND, OR, or NOR gates.

Delay Element

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

```
Fxxx n+ n- [CCCS] DELAY vn1 TD=val [SCALE=val]
+ [TC1=val] [TC2=val] NPDELAY=val
```

F-element Parameters

The following list describes The F-element parameters.

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
CCCS	Keyword for current-controlled current source. CCCS is a HSPICE reserved keyword; do not use it as a node name.
DELAY	Keyword for the delay element. Same as for a current-controlled current source, but has an associated propagation delay, TD. Adjusts the propagation delay in the macro model (subcircuit) process. DELAY is a reserved word; do not use it as a node name.
DELTA	Controls the curvature of piecewise linear corners. The default is 1/4 of the smallest distance between breakpoints. The maximum is 1/2 of the smallest distance between breakpoints.
Fxxx	Element name of the current-controlled current source. Must begin with F, followed by up to 1023 alphanumeric and special characters.
gain	Current gain.
gatetype(k)	AND, NAND, OR, or NOR. <i>k</i> is the number of inputs for the gate. <i>x</i> and <i>y</i> are the piecewise linear variation of the output, as a function of input. In multi-input gates, only one input determines the output state. Do not use the above keywords as node names.

Chapter 10: Sources and Stimuli
 Current-Dependent Current Sources — F-elements

Parameter	Description
IC	Initial condition (estimate) of the controlling current(s), in amps. If you do not specify IC, the default=0.0.
M	Number of replications of the element, in parallel.
MAX	Maximum output current. Default=undefined; sets no maximum.
MIN	Minimum output current. Default=undefined; sets no minimum.
n+/-	Connecting nodes for a positive or negative controlled source.
NDIM	Number of polynomial dimensions. If you do not specify POLY(NDIM), HSPICE assumes a one-dimensional polynomial. NDIM must be a positive number.
NPDELAY	Number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep. That is, $NPDELAY_{default} = \max\left[\frac{\min\langle TD, tstop \rangle}{tstep}, 10\right]$ The .TRAN statement specifies the tstep and tstop values.
P0, P1 ...	The polynomial coefficients. If you specify one coefficient, HSPICE assumes it is P1 (P0=0.0), and the source element is linear. If you specify more than one polynomial coefficient, then the source is non-linear, and HSPICE assumes that the polynomials are P0, P1, P2 ... See Polynomial Functions on page 275 .
POLY	Keyword for the polynomial function. If you do not specify POLY(ndim), HSPICE assumes that this is a one-dimensional polynomial. Ndim must be a positive number.
PWL	Keyword for the piecewise linear function.
SCALE	Multiplier for the element value.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $SCALE_{eff} = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$

Parameter	Description
TD	Keyword for the time (propagation) delay.
vn1 ...	Names of voltage sources, through which the controlling current flows. Specify one name for each dimension.
x1,...	Controlling current, through the <i>vn1</i> source. Specify the <i>x</i> values in increasing order.
y1,...	Corresponding output current values of <i>x</i> .

For a listing of output templates for current-controlled current source F-element (HSPICE only) models, see [Table 38 on page 440](#).

F-element Examples

Example 1

This example describes a current-controlled current source, connected between nodes 13 and 5. The current, which controls the value of the controlled source, flows through the voltage source named *VSENS*.

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

Note: To use a current-controlled current source, you can place a dummy independent voltage source into the path of the controlling current.

The defining equation is:

Equation 30

$$I(F1) = 5 \cdot I(VSENS)$$

- Current gain is 5.
- Maximum current flow through F1 is 3 A.
- Minimum current flow is -3 A.

If $I(VSENS) = 2 \text{ A}$, then this examples sets $I(F1)$ to 3 amps, not 10 amps (as the equation suggests). You can define a parameter for the polynomial coefficient(s):

```
.PARAM VU=5
F1 13 5 VSENS MAX=+3 MIN=-3 VU
```

Example 2

This example is a current-controlled current source, with the value:

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

Current flows from the positive node, through the source, to the negative node. The positive controlling-current flows from the positive node, through the source, to the negative node of `vnam` (linear), or to the negative node of each voltage source (nonlinear).

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

Example 3

This example is a delayed, current-controlled current source.

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

Example 4

This example is a piecewise-linear, current-controlled current source.

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

Voltage-Dependent Current Sources — G-elements

This section explains G-element syntax statements, and their parameters.

Both HSPICE and HSPICE RF support G-elements (and E-elements). The G-element specifies a current noise source with the noise value in $A^{2/Hz}$. The noise source can be bias and/or frequency dependent. For a white noise source, use a constant value for 'expression'.

For noise sources using expressions, the G-element takes the form:

```
gname n1 n2 noise='expression'
```

For full demonstration files for voltage-dependent current sources using G-element parameters for NPWL/PPWL/NAND circuits, see the paths to these example netlists in the section [Behavioral Application Examples](#).

- `det_dff.sp` (double edge-triggered flip-flop)
- `diode.sp` (behavioral diode by using a PWL VCCS)
- `dlatch.sp` (CMOS D-latch by using behavioral models)

Refer to other demonstration files in the following sections for specific circuits that use the G-element.

```
Gxxx n+ n- [VCCS|LEVEL=0] in+ in- ...
```

- LEVEL=0 is a Voltage-Controlled Current Source (VCCS).
- LEVEL=1 is a Voltage-Controlled Resistor (VCR).
- LEVEL=2 is a Voltage-Controlled Capacitor (VCCAP), Negative Piece-Wise Linear (NPWL).
- LEVEL=3 is a VCCAP, Positive Piece-Wise Linear (PPWL).

See also:

- [Using G- and E-elements](#) in the *HSPICE User Guide: Advanced Analog Simulation and Analysis*.
- [Filters Examples](#) (`lcline.sp`) for a demonstration of a G-element as a filter
- A listing of output templates for voltage-controlled current source G-element models (HSPICE only), [Table 36 on page 440](#).

The following sections introduce these topic:

- [Voltage-Controlled Current Source \(VCCS\)](#)
- [Behavioral Current Source \(Noise Model\)](#)
- [Voltage-Controlled Resistor \(VCR\)](#)
- [Voltage-Controlled Capacitor \(VCCAP\)](#)
- [G-element Examples](#)

Voltage-Controlled Current Source (VCCS)

The following sections discuss these topics:

- [G-element Parameters](#)
- [Linear](#)
- [Polynomial \(POLY\)](#)
- [Piecewise Linear \(PWL\)](#)
- [Multi-Input Gate](#)
- [Delay Element](#)

Chapter 10: Sources and Stimuli

Voltage-Dependent Current Sources — G-elements

- [Laplace Transform](#)
- [Pole-Zero Function](#)
- [Frequency Response Table](#)
- [Foster Pole-Residue Form](#)

G-element Parameters

The following list describes the G-element parameters.

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
CUR, VALUE, NOISE	Current output that flows from n+ to n-. The expression that you define can be a function of: <ul style="list-style-type: none">▪ node voltages▪ branch currents▪ time (<i>time variable</i>)▪ temperature (<i>temper variable</i>)▪ frequency (<i>hertz variable</i>)
DELAY	Keyword for the delay element. Same as in the voltage-controlled current source, but has an associated propagation delay, TD. Adjusts propagation delay in macro (subcircuit) modeling. DELAY is a keyword; do not use it as a node name.
DELTA	Controls curvature of piecewise linear corners. Defaults to 1/4 of the smallest distance between breakpoints. Maximum is 1/2 of the smallest distance between breakpoints.
Gxxx	Name of the voltage-controlled element. Must begin with G, followed by up to 1023 alphanumeric and special characters.
gatetype(k)	AND, NAND, OR, or NOR. The <i>k</i> parameter is the number of inputs of the gate. <i>x</i> and <i>y</i> represent the piecewise linear variation of the output, as a function of the input. In multi-input gates, only one input determines the state of the output.

Parameter	Description
LEVEL=0/1/2/3	<p>Function keyword such as VCCS, VCAP, etc.</p> <ul style="list-style-type: none"> ▪ LEVEL=0: Voltage-Controlled Current Source (VCCS). ▪ LEVEL=1: Voltage-Controlled Resistor (VCR). ▪ LEVEL=2: Voltage-Controlled Capacitor (VCCAP), Negative Piece-Wise Linear (NPWL). ▪ LEVEL=3: VCCAP, Positive Piece-Wise Linear (PPWL).
IC	Initial condition. Initial estimate of the value(s) of controlling voltage(s). If you do not specify IC, the default=0.0.
in +/-	Positive or negative controlling nodes. Specify one pair for each dimension.
M	Number of replications of the elements in parallel.
MAX	Maximum value of the current or resistance. The default is undefined, and sets no maximum value.
MIN	Minimum value of the current or resistance. The default is undefined, and sets no minimum value.
n+/-	Positive or negative node of the controlled element.
NDIM	Number of polynomial dimensions. If you do not specify POLY(NDIM), HSPICE assumes a one-dimensional polynomial. NDIM must be a positive number.
NPDELAY	<p>Sets the number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of TD/tstep and tstop/tstep.</p> <p>That is, $NPDELAY_{default} = \max\left[\frac{\min\langle TD, tstop \rangle}{tstep}, 10\right]$.</p> <p>The .TRAN statement specifies the tstep and tstop values.</p>
NPWL	Models symmetrical bidirectional switch/transfer gate, NMOS.

Chapter 10: Sources and Stimuli

Voltage-Dependent Current Sources — G-elements

Parameter	Description
P0, P1 ...	The polynomial coefficients. <ul style="list-style-type: none">▪ If you specify one coefficient, HSPICE assumes that it is P1 (P0=0.0), and the element is linear.▪ If you specify more than one polynomial coefficient, the element is non-linear, and the coefficients are P0, P1, P2 ... (see Polynomial Functions on page 275).
POLY	Keyword for the polynomial dimension function. If you do not specify <i>POLY(ndim)</i> , HSPICE assumes that it is a one-dimensional polynomial. <i>Ndim</i> must be a positive number.
PWL()	Keyword for the piecewise linear function; must include parenthetical ().
PPWL	Models symmetrical bidirectional switch/transfer gate, PMOS.
SCALE	Multiplier for the element value.
SMOOTH	For piecewise-linear, dependent-source elements, SMOOTH selects the curve-smoothing method. See Turning off Smoothing . A curve-smoothing method simulates exact data points that you provide. You can use this method to make HSPICE simulate specific data points, which correspond to either measured data or data sheets. Choices for SMOOTH are 1 or 2: <ul style="list-style-type: none">▪ Selects the smoothing method used in HSPICE versions before release H93A. Use this method to maintain compatibility with simulations that you ran, using releases older than H93A.▪ Selects the smoothing method, which uses data points that you provide. This is the default for HSPICE versions since release H93A.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the $SCALE: SCALE_{eff} = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$.
TD	Keyword for the time (propagation) delay.
transconductance	Voltage-to-current conversion factor.
transfactor	Voltage-to-resistance conversion factor.
VCCAP	Keyword for voltage-controlled capacitance element. VCCAP is a reserved HSPICE keyword; do not use it as a node name.

Parameter	Description
VCCS	Keyword for the voltage-controlled current source. VCCS is a reserved HSPICE keyword; do not use it as a node name.
VCR	Keyword for the voltage controlled resistor element. VCR is a reserved HSPICE keyword; do not use it as a node name. The LAPLACE, FREQ and FOSTER forms of G elements do not require the VCR.
x1,...	Controlling voltage, across the <i>in+</i> and <i>in-</i> nodes. Specify the <i>x</i> values in increasing order.
y1,...	Corresponding element values of <i>x</i> .
LAPLACE	Laplace transform resolves a function into its moments (see Laplace Transform on page 290)
FREQ	Response table (see Frequency Response Table on page 292)
FOSTER	Pole residue form (see Foster Pole-Residue Form on page 294)

Linear

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

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Polynomial (POLY)

```
Gxxx n+ n- [VCCS] POLY(NDIM) in1+ in1- ... [inndim+ inndim-]
+ [MAX=val] [MIN=val] [SCALE=val] [M=val] [TC1=val]
+ [TC2=val] [ABS=1] P0 [P1...] [IC=val]
```

For a description of the G-element parameters, see [G-element Parameters on page 310](#). For a description of possible POLY syntaxes, see [Polynomial Functions on page 275](#).

For a full demonstration file for voltage-dependent current sources using G-element parameters for a polynomial example, see the path to the example netlist `p11_bvp.sp` (PLL build with behavioral source) in the section [Behavioral Application Examples](#).

Piecewise Linear (PWL)

```
Gxxx n+ n- [VCCS] PWL(1) in+ in- [DELTA=val]
+ [SCALE=val] [M=val] [TC1=val] [TC2=val]
+ x1,y1 x2,y2 ... x100,y100 [IC=val] [SMOOTH=val]
Gxxx n+ n- [VCCS] NPWL(1) in+ in- [DELTA=val]
+ [SCALE=val] [M=val] [TC1=val] [TC2=val]
+ x1,y1 x2,y2 ... x100,y100 [IC=val] [SMOOTH=val]
Gxxx n+ n- [VCCS] PPWL(1) in+ in- [DELTA=val]
+ [SCALE=val] [M=val] [TC1=val] [TC2=val]
+ x1,y1 x2,y2 ... x100,y100 [IC=val] [SMOOTH=val]
```

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

For a set of full demonstration files for voltage-dependent current sources using G-element parameters for PWL examples, see the paths to the example netlists as follows in the section [Behavioral Application Examples](#):

- `switch.sp` (test for PWL switch element)
- `swrc.sp` (switched capacitor RC circuit)
- `pwl2.sp` (PPW-VCCS with a gain of 1 amp/volt)
- `pwl4.sp` (eight-input NAND gate)
- `ivx.sp` (characteristics of the PMOS and NMOS as a switch)
- `vcob.sp` (voltage-controlled oscillator by using PWL functions)

Multi-Input Gate

```
Gxxx n+ n- [VCCS] gatetype(k) in1+ in1- ...
+ ink+ ink- [DELTA=val] [TC1=val] [TC2=val] [SCALE=val]
+ [M=val] x1,y1 ... x100,y100 [IC=val]
```

In this syntax, `gatetype(k)` can be AND, NAND, OR, or NOR gates. For a description of the G-element parameters, see [G-element Parameters on page 310](#).

For a full demonstration example of AND/NAND gates that uses the G-element see also the path to the `behave.sp` netlist noted in the section [Behavioral Application Examples](#).

Delay Element

```
Gxxx n+ n- [VCCS] DELAY in+ in- TD=val [SCALE=val]
+ [TC1=val] [TC2=val] NPDELAY=val
```


For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Laplace Transform

For details, see [Laplace Transform on page 290](#).

Pole-Zero Function

For details, see [Pole-Zero Function on page 291](#).

Frequency Response Table

For details, see [Frequency Response Table on page 292](#).

Foster Pole-Residue Form

For details, see [Foster Pole-Residue Form on page 294](#).

Behavioral Current Source (Noise Model)

Expression form

```
Gxxx node1 node2 noise='noise_expression'
```

You can set the *xxx* parameter with a value up to 1024 characters. The *node1* and *node2* are the positive and negative nodes that connect to the noise source. The noise expression can contain the bias, frequency, or other parameters, and involve node voltages and currents through voltage sources.

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

This syntax creates a simple two-terminal current noise source; A^2/Hz describes its value. The output noise generated from this noise source is:

$$\text{noise_expression} * H$$

H is the transfer function from the terminal pair (node1,node2) to the circuit output, which measures the output noise.

You can also implement a behavioral noise source with an E-element. As noise elements, they are two-terminal elements that represent a noise source connected between two specified nodes.

```
gname node1 node2 node3 node4 noise='expression'
```

This syntax produces a noise source correlation between the terminal pairs (node1 node2) and (node3 node4). Compute the resulting output noise as:
`noise_expression*sqrt(H1*H2*)`

- H1 is the transfer function from (node1,node2) to the output.
- H2 is the transfer function from (node3,node4) to the output.

The noise expression can involve node voltages and currents through voltage sources.

Data form

```
Gxxx node1 node2 noise data=dataname
.DATA dataname
+ pname1 pname2
+ freq1 noise1
+ freq2 noise2
+ ...
.enddata
```

The data form defines a basic frequency-noise table. The `.DATA` statement contains two parameters: frequency and noise to specify the noise value at each frequency point. The unit for frequency is hertz, and the unit for noise is A^2/Hz .

The G-element with noise sources supports the following syntaxes:

- `Gxxx n1 n2 noise data=dataname`
- `Gxxx n1 n2 noise data=datablock`
- `Gxxx n1 n2 noisefile='filename'`
- `Gxxx n1 n2 noise='expression'`
- `Gxxx n1 n2 noise='Table(arg1,f1,v1,f2,v2...)'`
- `Gxxx n1 n2 noise='Table(arg1,dotDataBlockName)'`, where `dotDataBlockName` is the `.data` statement reference

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Example

The following netlist shows a 1000 ohm resistor (`g1`) using a G-element. The `g1noise` element, placed in parallel with the `g1` resistor, delivers the thermal noise expected from a resistor. The following example includes the `r1` resistor for comparison: The noise due to `r1` should be the same as the noise due to `g1noise`.

```
* Resistor implemented using g-element
v1 1 0 1
r1 1 2 1k
g1 1 2 cur='v(1,2)*0.001'
glnoise 1 2
+ noise='4*1.3806266e-23*(TEMPER+273.15)*0.001'
rout 2 0 1meg
.ac lin 1 100 100
.noise v(2) v1 1
.end
```

Voltage-Controlled Resistor (VCR)

Note: The LAPLACE, FREQ and FOSTER forms of G-elements do not require the VCR.

The following sections discuss these topics:

- [Linear](#)
- [Polynomial \(POLY\)](#)
- [Piecewise Linear \(PWL\)](#)
- [Multi-Input Gates](#)

Linear

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

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Polynomial (POLY)

```
Gxxx n+ n- VCR POLY(NDIM) in1+ in1- ...
+ [inndim+ inndim-] [MAX=val] [MIN=val] [SCALE=val]
+ [M=val] [TC1=val] [TC2=val] P0 [P1...] [IC=vals]
```

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Piecewise Linear (PWL)

```
Gxxx n+ n- VCR PWL(1) in+ in- [DELTA=val] [SCALE=val]
```

```
+ [M=val] [TC1=val] [TC2=val] x1,y1 x2,y2 ... x100,y100
+ [IC=val] [SMOOTH=val]
Gxxx n+ n- VCR NPWL(1) in+ in- [DELTA=val] [SCALE=val]
+ [M=val] [TC1=val] [TC2=val] x1,y1 x2,y2 ... x100,y100
+ [IC=val] [SMOOTH=val]
Gxxx n+ n- VCR PPWL(1) in+ in- [DELTA=val] [SCALE=val]
+ [M=val] [TC1=val] [TC2=val] x1,y1 x2,y2 ... x100,y100
+ [IC=val] [SMOOTH=val]
```

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Multi-Input Gates

```
Gxxx n+ n- VCR gatetype(k) in1+ in1- ... ink+ ink-
+ [DELTA=val] [TC1=val] [TC2=val] [SCALE=val] [M=val]
+ x1,y1 ... x100,y100 [IC=val]
```

For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Voltage-Controlled Capacitor (VCCAP)

```
Gxxx n+ n- VCCAP PWL(1) in+ in- [DELTA=val]
+ [SCALE=val] [M=val] [TC1=val] [TC2=val]
+ x1,y1 x2,y2 ... x100,y100 [IC=val] [SMOOTH=val]
```

HSPICE uses either LEVEL=2 (NPWL) or LEVEL=3 (PPWL), based on the relationship of the (n+, n-) and (in+, in-) nodes. For a description of the G-element parameters, see [G-element Parameters on page 310](#).

Use the NPWL and PPWL functions to interchange the n+ and n- nodes, but use the same transfer function.

The following sections summarize these actions:

- [NPWL Function](#)
- [PPWL Function](#)

NPWL Function

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

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

For the $in-$ node connected to $n-$:

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

PPWL Function

For the $in-$ node, connected to $n+$:

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

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

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

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

G-element Examples

The following section provide illustrative examples for the G-element:

- [Modeling Switches](#)
- [Switch-Level MOSFET](#)
- [Runtime Current Source with Equation Containing Output Variable](#)
- [Voltage-Controlled Capacitor](#)
- [Zero-Delay Gate](#)
- [Delay Element](#)
- [Diode Equation](#)
- [Diode Breakdown](#)
- [Diode Lookup Table \(vs. Model\)](#)
- [Triodes](#)
- [Behavioral Noise Model](#)
- [Turning off Smoothing](#)

- [Using Dependent Sources to Convert I to V and V to I](#)
- [Additional Full Demonstration Netlists Using G-Element](#)

Modeling Switches

You can model a switch to be open or closed based on simulation time or a pair of controlling nodes.

Switch Example 1: Time-varying switch—use the built-in function TIME to change the value of R from 0 (closed) to 100g ohm (open) when the simulator reaches time value T1:

```
R1 n1 n2 '100g*(TIME <= T1)'
```

As long as $TIME \leq T1$, the expression evaluates to zero and so does the resistor (switch) value. You could easily rewrite the resistor to switch from closed to open:

```
R1 n1 n2 '100g*(TIME >= T1)'
```

Switch example 2: Voltage-controlled switch—use a voltage-controlled resistor and the PWL (piece-wise linear) function. The point-value pair represents the controlling input voltage and output resistance respectively

```
G_Switch n1 n2 VCR PWL(1) c1 c2 0v,100g 1v,1p
```

where: n1 and n2 are the poles of the switch, and c1 and c2 are the control nodes. In the following sample netlist, the switch is controlled by the PWL voltage source to switch at 1us:

```
* g-element switch
.option post
V_ctrl c1 0 PWL (0 0v .99u 0v 1u 1v)
G_Switch n1 n2 VCR PWL(1) c1 0 0v,100g 1v,1p
V_ref n1 0 10v
R_load n2 0 100
.tran .1u 2u
.end
```

Switch-Level MOSFET

To model a switch level n-channel MOSFET, use the N-piecewise linear resistance switch. The resistance value does not change when you switch the *d* and *s* node positions.

```
Gnmos d s VCR NPWL(1) g s LEVEL=1 0.4v,150g
+ 1v,10meg 2v,50k 3v,4k 5v,2k
```

For a full demonstration example see the path to the `ac1.sp` netlist noted in the section [Behavioral Application Examples](#).

Runtime Current Source with Equation Containing Output Variable

HSPICE does not support a runtime output variable such as `v(gate)` in the example equation that follows. If the `.DATA` block has a runtime current source (I-element) where an equation contains runtime output variable such as `v(gate)`, as in this example equation,

```
I0 1 0 '(1-a0*v(gate))/b0'
vg gate 0 '(gt_v1)' $ (gt_v1)
```

—then the recommended method is to use the G-element:

```
g0 1 0 cur='((1-(a0*v(gate)))/b0)'
```

For a full demonstration example of runtime current (amplitude modulator with pulse waveform carrier) see the path to the `amp_mod.sp` netlist noted in the section [Behavioral Application Examples](#). See also in [Behavioral Application Examples](#) the path to `galg1.sp` for sampling a sine wave with a current source.

Voltage-Controlled Capacitor

The capacitance value across the (`out,0`) nodes varies linearly (from 1 p to 5 p), when voltage across the `ctrl,0` nodes varies between 2 v and 2.5 v. The capacitance value remains constant at 1 picofarad when below the lower voltage limit, and at 5 picofarads when above the upper voltage limit.

```
Gcap out 0 VCCAP PWL(1) ctrl 0 2v,1p 2.5v,5p
```

Zero-Delay Gate

To implement a two-input AND gate, use an expression and a piecewise linear table.

- The inputs are voltages at the `a` and `b` nodes.
- The output is the current flow from the `out` to `0` node.
- HSPICE multiplies the current by the `SCALE` value—which in this example, is the inverse of the load resistance, connected across the `out,0` nodes.

```
Gand out 0 AND(2) a 0 b 0 SCALE='1/rload' 0v,0a 1v,.5a
+ 4v,4.5a 5v,5a
```

Delay Element

A delay is a low-pass filter type delay, similar to that of an opamp. In contrast, a transmission line has an infinite frequency response. A glitch input to a G delay attenuates in a way that is similar to a buffer circuit. In this example, the output of the delay element is the current flow from the *out* node to the *1* node with a value equal to the voltage across the (*in*, *0*) nodes, multiplied by the `SCALE` value, and delayed by the `TD` value.

```
Gdel out 0 DELAY in 0 TD=5ns SCALE=2 NPDELAY=25
```

For a full demonstration example of delay parameter (five-stage ring oscillator – macromodel CMOS inverter) see the path to the `ring5bm.sp` netlist.

Diode Equation

To model forward-bias diode characteristics from node 5 to ground use a runtime expression. The saturation current is $1e-14$ amp and the thermal voltage is 0.025 v.

```
Gdio 5 0 CUR='1e-14*(EXP(V(5)/0.025)-1.0)'
```

Diode Breakdown

You can model the diode breakdown region to a forward region. When voltage across a diode is above or below the piecewise linear limit values (-2.2v, 2v), the diode current remains at the corresponding limit values (-1a, 1.2a).

```
Gdiode 1 0 PWL(1) 1 0 -2.2v,-1a -2v,-1pa .3v,.15pa  
+.6v,10ua 1v,1a 2v,1.2a
```

Diode Lookup Table (vs. Model)

In HSPICE you can use the diode lookup-table values in a G-element (VCCS) as a PWL table. Here is a simple netlist using the lookup table values in a G-element:

```
.option post  
Gdiode 1 0 PWL(1) 1 0 -2.2v,-1a -2v,-1pa .3v,.15pa  
+ .6v,10ua 1v,1a 2v,1.2a  
V1 1 0 2v  
.op  
.print i(*)  
.tran 1n 10n  
.end
```


The only limitation is that the maximum number of I-V value pairs for a PWL G-element is 100.

Triodes

Both of the following voltage-controlled current sources implement a basic triode.

- The first example uses the `poly(2)` operator, to multiply the anode and grid voltages together, and to scale by `.02`.
- The second example uses the explicit behavioral algebraic description.

```
gt i_anode cathode poly(2) anode,cathode
+ grid,cathode 0 0 0 0 .02
gt i_anode cathode
+ cur='20m*v(anode,cathode)*v(grid,cathode)'
```

Behavioral Noise Model

The following netlist shows a 1000 Ohm resistor (`g1`) implemented using a G-element. The `g1noise` element, placed in parallel with the `g1` resistor, delivers the thermal noise expected from a resistor. The following example includes the `r1` resistor for comparison: the noise due to `r1` should be the same as the noise due to `g1noise`.

```
* Resistor implemented using g-element
v1 1 0 1
r1 1 2 1k
g1 1 2 cur='v(1,2)*0.001'
g1noise 1 2 noise='sqrt(4*1.3806266e-23*(TEMPER+273.15)*0.001) '
rout 2 0 1meg
.ac lin 1 100 100
.noise v(2) v1 1
.end
```

Turning off Smoothing

By default, a PWL controlled source performs smoothing on corners which may introduce some error. To turn off the smoothing, you can use `DELTA=0` in the G-element definition.

```
Gtest BB 0 VCCS PWL(1) B E 0,0 0.1,0 1,1 2,2 DELTA=0
```

You can also set `DELTA` to a small value to get some smoothing which can help convergence but reduce the difference from the ideal PWL.

Using Dependent Sources to Convert I to V and V to I

You can convert voltage to current using the G-element and current to voltage by using the H-element (see [Current-Controlled Voltage Source \(CCVS\)—H-Element](#)). To use the H-element for a I to V conversion, you also need to use a dummy voltage source. For example:

```
* I-V and V-I conversion
.option post list node
*** I-V Conversion ****
I_in      p1 0 1
vdummy    p2 p1 0 $ dummy voltage source
H_in      p2 0 CCVS vdummy 1
R_fc      p2 p3 10
C_fc      p3 0 500fF
*** V-I Conversion ****
G_in      p4 p5 VCCS p3 0 1
Rs        p5 p6 1
C1        p4 p5 500fF
C2        p4 p6 500fF
.tran 10p 1n
.probe tran v(p?) i(g*)
.end
```

Additional Full Demonstration Netlists Using G-Element

For an additional set of full demonstration files for voltage-dependent current sources using G-element parameters examples, see the paths to the example netlists as follows in the section [Behavioral Application Examples](#):

- pdp.sp (phase detector by using behavioral NAND gates Multi-Input NAND)
- ringb.sp (ring oscillator by using behavioral model—NAND)
- vcob.sp (voltage-controlled oscillator by using PWL functions—PWL/NAND)
- rtest.sp (voltage-controlled resistor, inverter chain—Level=1)
- vcob.sp (voltage-controlled oscillator by using PWL functions)

Current-Dependent Voltage Sources — H-elements

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

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

For a listing of output templates for current-controlled voltage source (H-element) models (HSPICE only), see [Table 39 on page 441](#).

The following section introduces the topics for the CCVS source.

- [Current-Controlled Voltage Source \(CCVS\)—H-Element](#)

Current-Controlled Voltage Source (CCVS)—H-Element

The following sections discuss these topics:

- [Linear](#)
- [Polynomial \(POLY\)](#)
- [Piecewise Linear \(PWL\)](#)
- [Multi-Input Gate](#)
- [Delay Element](#)

Linear

```
Hxxx n+ n- [CCVS] vn1 transresistance [MAX=val] [MIN=val]  
+ [SCALE=val] [TC1=val] [TC2=val] [ABS=1] [IC=val]
```

Polynomial (POLY)

```
Hxxx n+ n- [CCVS] POLY(NDIM) vn1 [... vnndim]  
+ [MAX=val] [MIN=val] [TC1=val] [TC2=val] [SCALE=val]  
+ [ABS=1] P0 [P1...] [IC=val]
```

Piecewise Linear (PWL)

```
Hxxx n+ n- [CCVS] PWL(1) vn1 [DELTA=val] [SCALE=val]  
+ [TC1=val] [TC2=val] x1,y1 ... x100,y100 [IC=val]
```

For full demonstration examples of H-element PWL netlist files, go to [Behavioral Application Examples](#) and follow the paths to these two netlists:

- `op_amp.sp` (OPAMP from Chua and Lin)
- `pwl10.sp` (OPAMP used as a voltage follower)

Multi-Input Gate

```
Hxxx n+ n- gatetype(k) vn1, ...vnk [DELTA=val] [SCALE=val]
+ [TC1=val] [TC2=val] x1,y1 ... x100,y100 [IC=val]
```

In this syntax, `gatetype(k)` can be AND, NAND, OR, or NOR gates.

Delay Element

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

```
Hxxx n+ n- [CCVS] DELAY vn1 TD=val [SCALE=val] [TC1=val]
+ [TC2=val] [NPDELAY=val]
```

Parameter	Description
ABS	Output is an absolute value, if ABS=1.
CCVS	Keyword for the current-controlled voltage source. CCVS is a HSPICE reserved keyword; do not use it as a node name.
DELAY	Keyword for the delay element. Same as for a current-controlled voltage source, but has an associated propagation delay, TD. Use this element to adjust the propagation delay in the macro (subcircuit) model process. DELAY is a HSPICE reserved keyword; do not use it as a node name.
DELTA	Controls curvature of piecewise linear corners. The default is 1/4 of the smallest distance between breakpoints. Maximum is 1/2 of the smallest distance between breakpoints.
gatetype(k)	Can be AND, NAND, OR, or NOR. The <i>k</i> value is the number of inputs of the gate. The <i>x</i> and <i>y</i> terms are the piecewise linear variation of the output, as a function of the input. In multi-input gates, one input determines the output state.
Hxxx	Element name of current-controlled voltage source. Must start with H, followed by up to 1023 alphanumeric and special characters.
IC	Initial condition (estimate) of the controlling current(s), in amps. If you do not specify IC, the default=0.0.

Parameter	Description
MAX	Maximum voltage. Default is undefined; sets no maximum.
MIN	Minimum voltage. Default is undefined; sets no minimum.
n+/-	Connecting nodes for positive or negative controlled source.
NDIM	Number of polynomial dimensions. If you do not specify <code>POLY(NDIM)</code> , HSPICE assumes a one-dimensional polynomial. NDIM must be a positive number.
NPDELAY	<p>Number of data points to use in delay simulations. The default value is the larger of either 10, or the smaller of $TD/tstep$ and $tstop/tstep$. That is:</p> $NPDELAY_{default} = \max\left[\frac{\min\langle TD, tstop \rangle}{tstep}, 10\right].$ <p>The <code>.TRAN</code> statement specifies the <code>tstep</code> and <code>tstop</code> values.</p>
P0, P1 . . .	<p>Polynomial coefficients.</p> <ul style="list-style-type: none"> ▪ If you specify one polynomial coefficient, the source is linear, and HSPICE assumes that the polynomial is P1 ($P0=0.0$). ▪ If you specify more than one polynomial coefficient, the source is non-linear. HSPICE assumes the polynomials are P0, P1, P2 . . . See Polynomial Functions on page 275.
POLY	Keyword for polynomial dimension function. If you do not specify <code>POLY(ndim)</code> , HSPICE assumes a one-dimensional polynomial. <i>Ndim</i> must be a positive number.
PWL	Keyword for a piecewise linear function.
SCALE	Multiplier for the element value.
TC1,TC2	First-order and second-order temperature coefficients. Temperature changes update the SCALE: $SCALE_{eff} = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$
TD	Keyword for the time (propagation) delay.
transresistance	Current-to-voltage conversion factor.
vn1 . . .	Names of voltage sources, through which controlling current flows. You must specify one name for each dimension.

Parameter	Description
x1,...	Controlling current, through the <i>vn1</i> source. Specify the <i>x</i> values in increasing order.
y1,...	Corresponding output voltage values of <i>x</i> .

Example 1

```
HX 20 10 VCUR MAX=+10 MIN=-10 1000
```

The example above selects a linear current-controlled voltage source. The controlling current flows through the dependent voltage source, called VCUR.

Example 2

The defining equation of the CCVS is:

Equation 31

$$HX = 1000 \cdot I(VCUR)$$

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

- If the equation produces a value of HX greater than +10 V, then the MAX parameter sets HX to 10 V.
- If the equation produces a value of HX less than -10 V, then the MIN parameter sets HX to -10 V.

VCUR is the name of the independent voltage source, through which the controlling current flows. If the controlling current does not flow through an independent voltage source, you must insert a dummy independent voltage source.

Example 3

```
.PARAM CT=1000
HX 20 10 VCUR MAX=+10 MIN=-10 CT
HXY 13 20 POLY(2) VIN1 VIN2 0 0 0 0 1 IC=0.5, 1.3
```

The example above describes a dependent voltage source, with the value:

Equation 32

$$V = I(VIN1) \cdot I(VIN2)$$

This two-dimensional polynomial equation specifies:

- FA1=VIN1
- FA2=VIN2

- P0=0
- P1=0
- P2=0
- P3=0
- P4=1

The initial controlling current is .5 mA through VIN1, and 1.3 mA for VIN2.

Positive controlling current flows from the positive node, through the source, to the negative node of vnam (linear). The (non-linear) polynomial specifies the source voltage, as a function of the controlling current(s).

HSPICE RF-Specific Sources and Stimuli

The following section discuss use of RF analysis-specific sources.

- [Steady-State Voltage and Current Sources](#)
- [Steady-State HB Sources](#)
- [Phase Differences Between HB and SIN Sources](#)
- [Behavioral Noise Sources](#)
- [Function Approximations for Distributed Devices](#)
- [Complex Signal Sources and Stimuli](#)
- [SWEEPBLOCK in Sweep Analyses](#)
- [Clock Source with Random Jitter](#)

Steady-State Voltage and Current Sources

The I (current source) and V (voltage source) elements include extensions that allow you to use them as sources of steady-state sinusoidal signals for HB and HBAC analyses. When you use a power parameter to specify the available power, you can also use these elements as power sources.

For a general description of the I and V elements, see [Power Sources on page 280](#) in this chapter.

I and V Element Syntax

```
Vxxx p n
+ $ **** Voltage or Power Information ****
+ [[dc] mag] [ac [mag [phase]]] [HBAC [mag [phase]]]
+[SNAC [mag [phase]]]
+ [hb [mag [phase [harm [tone [modharm [modtone]]]]]]]
+ [transient waveform] [TRANFORHB=[1|0]]

+ $ **** Power Switch ****
+ [power=[0 | 1 | W | dbm]] [z0=val] [rdc=val] [rac=val]
+ [RHBAC=val] [rhb=val] [rtran=val]

Ixxx p n
+ $ **** Current or Power Information ****
+ [[dc] mag] [ac [mag [phase]]] [HBAC [mag [phase]]]
+[SNAC [mag [phase]]]
+ [hb [mag [phase [harm [tone [modharm [modtone]]]]]]]
+ [transient waveform] [TRANFORHB=[1|0]]

+ $ **** Power Switch ****
+ [power=[0 | 1 | W | dbm]] [z0=val] [rdc=val] [rac=val]
+ [RHBAC=val] [rhb=val] [rtran=val]
```

Parameter	Description
[[dc] mag]	DC voltage or power source value. You do not need to specify DC explicitly (default=0).
[ac [mag [phase]]]	AC voltage or power source value.
[HBAC [mag [phase]]]	(HSPICE RF) HBAC voltage or power source value.
[SNAC [mag [phase]]]	(HSPICE RF) SNAC voltage or power source value.

Parameter	Description
[hb [mag [phase [harm [tone [modharm [modtone]]]]]]]]]	<p>(HSPICE RF) HB voltage, current, or power source value. Multiple HB specifications with different harm, tone, modharm, and modtone values are legal.</p> <ul style="list-style-type: none"> ▪ <i>phase</i> is in degrees ▪ <i>harm</i> and <i>tone</i> are indices corresponding to the tones specified in the .HB statement. Indexing starts at 1 (corresponding to the first harmonic of a tone). ▪ <i>modtone</i> and <i>modharm</i> specify sources for multi-tone simulation. A source specifies a tone and a harmonic, and up to 1 offset tone and harmonic (modtone for tones and modharm for harmonics). The following then describes the signal as: $V(\text{or } I) = \text{mag} * \cos(2 * \pi * (\text{harm} * \text{tone} + \text{modharm} * \text{modtone}) * t + \text{phase})$
[transient waveform]	<p>(Transient analysis) Any one of waveforms: AM, EXP, PULSE, PWL, SFFM, or SIN. Multiple transient descriptions are illegal.</p>
[power=[0 1 W dbm]]	<p>(HSPICE RF) Power Switch</p> <ul style="list-style-type: none"> ▪ When 0 (default), element treated as a voltage or current source. ▪ When 1 or W, element treated as a power source, realized as a voltage source with a series impedance. In this case, HSPICE interprets the source value as RMS available power in units of Watts. ▪ When dbm, element treated as a power source in series with the port impedance. Values are in dbms. <p>You can use this parameter for Transient analysis if the power source is either DC or SIN.</p>
[z0= <i>val</i>]	<p>(LIN analysis) System impedance used when converting to a power source, inserted in series with the voltage source. Currently, this only supports real impedance.</p> <ul style="list-style-type: none"> ▪ When power=0, z0 defaults to 0. ▪ When power=1, z0 defaults to 50 ohms. <p>You can also enter <i>zo=val</i>.</p>
[rdc= <i>val</i>]	<p>(DC analysis) Series resistance (overrides z0).</p>
[rac= <i>val</i>]	<p>(AC analysis) Series resistance (overrides z0).</p>

Parameter	Description
[RHBAC= <i>val</i>]	(HSPICE RF HBAC analysis) Series resistance (overrides z0).
[rhb= <i>val</i>]	(HSPICE RF HB analysis) Series resistance (overrides z0).
[rtran= <i>val</i>]	(Transient analysis) Series resistance (overrides z0).
[TRANFORHB=[0 1]]	<ul style="list-style-type: none"> ▪ 0 (default): HSPICE RF ignores the transient description when an HB or DC value exists. If no DC or HB value exists and TRANFORHB=0, then HB treats the source as a DC source, and the DC source value is the time=0 value. ▪ 1: HB analysis uses the transient description if its value is VMRF, SIN, PULSE, PWL, or LFSR. If the type is a non-repeating PWL source, then a DC source value uses the time=infinity value. For example, the following statement is treated as a DC source with value=1 for HB: v1 1 0 PWL (0 0 1n 1 1u 1) TRANFORHB=1 In contrast, the following statement is a 0V DC source: v1 1 0 PWL (0 0 1n 1 1u 1) TRANFORHB=0 The following statement is treated as a periodic source with a 1us period that uses PWL values: v1 1 0 PWL (0 0 1n 1 0.999u 1 1u 0) R TRANFORHB=1 To override the global TRANFORHB option, explicitly set TRANFORHB for a V/I source.

Example 1

This example shows an HB source for a single tone analysis:

```
.hb tones=100MHz harms=7
```

```
I1 1 2 dc=1mA hb 3mA 0. 1 1
```

I1 is a current source with a the following time-domain description:

```
I1=1mA + 3mA*cos(2*pi*1.e8*t)
```

Example 2

This example shows HB sources used for a two-tone analysis:

```
.hb tones=1.e9 1.1e9 intmodmax=5  
Vin lo 0 dc=0. hb 1.5 90 1 1
```

```
Vrf rf 0 dc=0. hb 0.2 0 1 2
```

These sources have the following time-domain descriptions:

$V_{in} = 1.5 \cdot \cos(2 \cdot \pi \cdot 1.1 \cdot 10^9 \cdot t - 90 \cdot \pi / 180)$ V

$V_{rf} = 0.2 \cdot \cos(2 \cdot \pi \cdot 1.1 \cdot 10^9 \cdot t)$ V

Example 3

The following HB source uses a `modtone` and `modharms`:

```
.hb tones=2.e9 1.9e9 harms=5 5
```

```
Vm input gnd dc=0.5 hb 0.2 0. 1 1 -1 2
```

V_m has the following time-domain description:

$V_m = 0.5 + \cos(2 \cdot \pi \cdot 1.1 \cdot 10^8 \cdot t)$

Example 4

This example uses an HB source specified with a `SIN` source and `HBTRANINIT`.

```
.hb tone=1.e8 harms=7
Vt 1 2 SIN(0.1 1.0 2.e8 0. 0. 90) tranforhb=1
```

Where V_t converts to the following HB source:

```
Vt 1 2 dc=0.1 hb 1.0 0.0 2 1
```

Example 5

This example shows a power source (the units are Watts).

```
.hb tones=1.1e9 harms=9
```

```
Pt Input Gnd power=1 Z0=50. 1m 0. 1 1
```

P_t delivers 1 mW of power through a 50 ohm impedance.

Steady-State HB Sources

Use the `.HB TONES` command to specify the fundamental frequencies used with harmonic balance analysis. You can then reference these frequencies by their integer indices when specifying steady-state signal sources. For example:

```
.HB TONES=1900MEG,1910MEG INTMODMAX=5
```

This specifies two fundamental frequencies: $f_{[tone = 1]} = 1.9 \text{ GHz}$ and $f_{[tone = 2]} = 1.91 \text{ GHz}$. You can then reference their mixing product at 10 MHz using indices as $|f_{[2]} - f_{[1]}|$, while you reference their 3rd order intermodulation product at 1.89 GHz as $|2f_{[1]} - f_{[2]}|$.

Identify steady-state voltage and current sources with the HB keyword according to

```
[HB [mag [phase [harm [tone [modharm [modtone]]]]]]]]
```

The source is mathematically equivalent to a cosine signal source that follows the equation

$$A \cos(\omega t + \phi)$$

where

$$A = mag$$

$$\omega = 2\pi|harm \cdot f[tone] + modharm \cdot f[modtone]|$$

$$\phi = \frac{\pi}{180} \cdot phase$$

Values for tone and modtone (an optional modulating tone) must be non-negative integers that specify index values for the frequencies specified with the .HB TONES command. Values for harm (harmonic) and modharm (modulating tone harmonic) must be integers (negative values are OK) that specify harmonic indices.

Example 1

The following example is a 1.0 Volt (peak) steady-state cosine voltage source, which is at the fundamental HB frequency with zero phase and with a zero volt DC value:

```
Vsrc in gnd DC 0 HB 1.0 0 1 1
```

Example 2

The following example is a steady-state cosine power source with 1.0mW available power, which is implemented with a Norton equivalent circuit and a 50 ohm input impedance:

```
Isrc in gnd HB 1.0e-3 0 1 1 power=1 z0=50
```

Example 3

Five series voltage sources sum to produce a stimulus of five equally spaced frequencies at and above 2.44 GHz using modharm and modtone parameters. These are commensurate tones (an integer relation exists); therefore, you only need to specify two tones when invoking the HB analysis.

```
.param Vin=1.0
.param f0=2440MEG
.param deltaf=312.5K
.param fcenter='f0 + 2.0*deltaf'
Vrfa    in    ina    HB    'Vin'    0    1    1                $ 2.440625
      GHz
Vrfb    ina    inb    HB    'Vin'    0    1    1    -1    2    $
2.4403125 GHz
Vrfc    inb    inc    HB    'Vin'    0    1    1    -2    2    $
2.440    GHz
Vrfd    inc    ind    HB    'Vin'    0    1    1    +1    2    $
2.4409375 GHz
Vrfe    ind    gnd    HB    'Vin'    0    1    1    +2    2    $ 2.44125
      GHz
.HB tones=fcenter,deltaf intmodmax=5
```

Phase Differences Between HB and SIN Sources

The HB steady-state cosine source has a phase variation compared to the TRAN time-domain SIN source. The SIN source (with no offset, delay or damping) follows the equation:

Equation 33

$$A \sin(\omega t + \phi)$$

while the HB sources follow

Equation 34

$$A \cos(\omega t + \phi)$$

In order for the two sources to yield identical results it is necessary to align them by setting their phase values accordingly using:

Equation 35

$$A \cos(\omega t + \phi) = A \sin(\omega t + \phi + 90^\circ)$$

Equation 36

$$A \sin(\omega t + \phi) = A \cos(\omega t + \phi - 90^\circ)$$

To specify sources with matching phase for HB and TRAN analysis, use a convention similar to:

Chapter 10: Sources and Stimuli

HSPICE RF-Specific Sources and Stimuli

```
** Example #1 with equivalent HB and SIN sources
** SIN source is given +90 phase shift
.param freq1=2400MEG Vin=1.0
Vsrc in gnd DC 0 HB 'Vin' 0 1 1 SIN(0 'Vin' 'freq1' 0 0 90)
.HB tones=freq1 intmodmax=7
** Example #2 with equivalent HB and SIN sources
** HB source is given -90 phase shift to align with SIN
.param freq1=2400MEG Vin=1.0
Vsrc in gnd DC 0 HB 'Vin' -90 1 1 SIN(0 'Vin' 'freq1' 0)
.HB tones=freq1 intmodmax=7
** Example #3 with equivalent .HB and .TRAN sources
** SIN source is activated for HB using "TRANFORHB"
.param freq1=2400MEG Vin=1.0
Vsrc in gnd DC 0 SIN(0 'Vin' 'freq1' 0) TRANFORHB=1
.HB tones=freq1 intmodmax=7
```

Behavioral Noise Sources

In HSPICE RF, you can use the G-element to specify noise sources. Frequency domain noise analyses (.NOISE, .HBNOISE, and .PHASENOISE) take these noise sources into account.

You can attach noise sources to behavioral models. For example, you can use a G-element with the VCCAP parameter to model a varactor, which includes a noise model. You can also simulate effects such as substrate noise, including its effect on oscillator phase noise. You can also use this G element syntax to simulate behavioral descriptions of substrate noise during any frequency domain noise analysis, which includes phase noise analysis. For example,

```
gname node1 node2 noise='noise_equation'
gname node1 node2 node3 node4 noise='noise_equation'
```

The first line creates a simple two-terminal current noise source, whose value is described in $A^2/(\text{Hz})$. The output noise generated from this noise source is:

noise_equation*H

Where H is the transfer function from the terminal pair (node1,node2) to the circuit output, where HSPICE RF measures the output noise.

The second line produces a noise source correlation between the (node1,node2) and (node3,node4) terminal pairs. The resulting output noise is calculated as noise_equation*sqrt(H1*H2*); where,

- H1 is the transfer function from (node1,node2) to the output
- H2 is the transfer function from (node3,node4) to the output.

The `noise_equation` expression can involve node voltages and currents through voltage sources.

For the PAC phasenoise simulation to evaluate the frequency-dependent noise, the frequency-dependent noise factor in the phasenoise must be expressed in between the parentheses. For example:

```
gname node1 node2 noise = '(frequency_dependent_noise)*  
bias_dependent_noise'
```

This is only true when the total noise can be expressed in this form and when the frequency-dependent noise can be evaluated in the PAC phasenoise simulation. You can also input the behavioral noise source as a noise table with the help of predefined `Table()` function. The `Table()` function takes two formats:

- Noise table can be input directly through the `Table()` function. For example:

```
gname node1 node2 noise = 'Table(arg1,f1,v1,f2,v2,.....)'
```

- The `f1,v1,f2,v2,.....` parameters describe the noise table. When `arg1 == f1`, the function returns `v1`. The `arg1` can be an expression of either `HERTZ`, `bias`, or both. For example, `arg1 = 'HERTZ * 1.0E+3'`.

- The noise table can be input through a `.DATA` structure:

```
.DATA d1  
+ x y  
+ f1 v1  
+ f2 v2  
.ENDDATA
```

```
gname node1 node2 noise = 'TABLE(arg1,d1)'
```

The `x, y` parameters in the `DATA` structure are two placeholder strings that can be set to whatever you prefer even if they are in conflict with other parameters in the netlist. The `arg1` parameter can be an expression of `HERTZ` and `bias`. When `arg1 == f2`, the function will return `v2`.

The following sections discuss these topics:

- [Using Noise Analysis Results as Input Noise Sources](#)
- [Syntax Rules for Behavioral Noise Source Files](#)
- [Power Supply Current and Voltage Noise Sources](#)

Using Noise Analysis Results as Input Noise Sources

By extending the E- and G- voltage-controlled source syntax, the phase noise or noise data in ASCII phase noise and noise files can be used as input for specifying behavioral noise sources.

Usage Model

The syntax for the voltage controlled voltage (E) or current (G) source is:

```
Exxx node1 node2 noisefile='filename' [mname='measname']  
Gxxx node1 node2 noisefile='filename' [mname='measname']
```

Where,

`noisefile='filename'` is the name of the ASCII noise data file. Phase noise analysis and noise analyses can output simulation results as ASCII data using `.PRINT` statements. The file name is designated as `'design.printsnpn0'` for `.SNOSC` phase noise and `.SNNOISE` analyses. For `.PHASENOISE`, `.HBNOISE` and `.ACPHASENOISE` analyses, the file name is designated as `'design.printtpn0'`. For `.NOISE` analyses, the file name is designated as `'design.printac0'`.

`mname='measname'` is used to select the appropriate noise measurement name to be taken from the `*.printtpn0`, `*.printsnpn0`, or `*.printac0` file.

`measname` can be one of the following:

- `NLP_L(f)` - selects the `nlp_L(f)` phase noise data in units of dBc/Hz
- `PAC_L(f)` - selects the `pac_l(f)` phase noise data in units of dBc/Hz
- `BPN_L(f)` - selects the `bpn_l(f)` phase noise data in units of dBc/Hz
- `ONoise` - selects the `onoise` data based on `.SNNOISE` or `.NOISE` analysis

The following syntaxes are supported in both HSPICE RF and HSPICE:

- `Exxx n1 n2 noise data=dataname`
- `Exxx n1 n2 noise data=datablock`
- `Exxx n1 n2 noisefile='filename'`
- `Exxx n1 n2 noise='expression'`
- `Exxx n1 n2 noise='Table(arg1,f1,v1,f2,v2...)'`
- `Exxx n1 n2 noise='Table(arg1,dotDataBlockName)'`, where `dotDataBlockName` is the `.data` statement reference

Syntax Rules for Behavioral Noise Source Files

You can use ASCII noise data files not generated by `.PRINT` statements when you follow these syntax rules:

- A noise file always needs a header (do not include frequency or noise points in the header line, as they will be ignored)
- The items in the header must be consistent with the number of data columns
- You can use noise keywords in the header (see below)
- The noise value units depend on your use of the `MNAME` keyword

You can use noise keywords like `ONoise`, and the phase noise keywords, `PAC_L(f)`, `BPN_L(f)`, and, `NLP_L(f)`. This is useful for `*.printac#`, `*.printpn#`, or `*.printsnpn#` files and use of the `MNAME` keyword to specify the noise data column to be used. For example, if you do a phase noise analysis and specify broadband phase noise (`method=2` in the `.PHASENOISE` command), the `*.printpn#` file contains three columns of phase noise data: `PAC_L(f)`, `BPN_L(f)`, and, `NLP_L(f)`.

If `MNAME` is not specified with the behavioral noise source, then HSPICE assumes that the 2nd column in the noise file represents the desired noise values, and the `MNAME` string is read from the noise file header. The first column is always assumed to contain frequency values.

The `MNAME` keyword is used to differentiate units. If you use `MNAME=ONoise` and the file contains `ONoise` in the header, then the noise units are interpreted as $V/\sqrt{\text{Hz}}$. The same is true with `MNAME=ONoise(M)` and `MNAME=ONoise(MAG)`. If `MNAME` is set to `PAC_L(f)`, `BPN_L(f)`, `NLP_L(f)`, or `PHNOISE`, then units are interpreted as dBc/Hz . With `MNAME=ONoise(DB)`, units are taken as dB/Hz ($20 \cdot \text{Log}(\text{onoise})$), as are other `MNAME` keywords that end in `DB` or `(DB)`. Most other `MNAME` keywords are generally interpreted in V^2/Hz units, unless the name ends in `(M)` or `(MAG)`, which are taken to be $V/\sqrt{\text{Hz}}$. Depending on the noise units, different interpolation schemes are used for frequencies needed between noise data points. Noise in dB/Hz or dBc/Hz units use logarithmic interpolation (in x and y), while non-dB units use linear interpolation in V^2/Hz .

Noise File Examples

The following examples provide sample noise results noting the various noise value units:

Chapter 10: Sources and Stimuli

HSPICE RF-Specific Sources and Stimuli

Example 1: Phase noise file generated by using `.PRINT phasenoise phnoise` statement. The noise units are dBc/Hz.

```
HERTZ  NLP_L(f)
10.0000000000k      -130.2612492663
12.5892541179k     -132.2612484153
15.8489319246k     -134.2612470666
19.9526231497k     -136.2612449295
25.1188643151k     -138.2612415429
31.6227766017k     -140.2612361768
39.8107170553k     -142.2612276743
50.1187233627k     -144.2612142034
63.0957344480k     -146.2611928626
79.4328234724k     -148.2611590575
100.0000000000k    -150.2611055150
125.8925411794k    -152.2610207237
158.4893192461k    -154.2608864692
199.5262314969k    -156.2606739394
251.1886431510k    -158.2603375701
316.2277660168k    -160.2598053187
398.1071705535k    -162.2589632779
501.1872336273k    -164.2576312765
630.9573444802k    -166.2555240055
794.3282347243k    -168.2521885144
1.0000000000x      -170.2469020563
```

Example 2: Noise file generated by using `.PRINT snnoise onoise` statement. The noise units are V/sqrt(Hz).

```
HERTZ  ONOISE()
100.0000      1.6699u
526.4105k     46.8034n
1.0527x       45.1538n
1.5790x       44.6096n
2.1053x       44.3405n
2.6317x       44.1804n
3.1580x       44.0744n
3.6843x       43.9990n
4.2106x       43.9425n
4.7369x       43.8986n
5.2632x       43.8634n
5.7895x       43.8345n
6.3158x       43.8102n
6.8421x       43.7894n
7.3684x       43.7714n
7.8948x       43.7556n
8.4211x       43.7415n
8.9474x       43.7288n
9.4737x       43.7172n
```

10.0000x 43.7067n

Example 3: User-generated noise file. Units are in dBc/Hz.

```
HERTZ  NOISE(DB)
10.0    -120
100.0   -125
1.0k    -130
5.0k    -135
10.0k   -140
100.0k  -150
```

Power Supply Current and Voltage Noise Sources

You can implement the power supply noise source with G- and E- elements. The G-element for the current noise source and the E-element for the voltage noise source. As noise elements, they are two-terminal elements that represent a noise source connected between two specified nodes.

Syntax

Expression form

```
Gxxx node1 node2 noise='expression'
Exxx node1 node2 noise='expression'
```

The G noise element represents a noise current source and the E noise element represents a noise voltage source. The *xxx* parameter can be set with a value up to 1024 characters. The *node1* and *node2* are the positive and negative nodes that connect to the noise source. The noise expression can contain the bias, frequency, or other parameters.

Data form

```
Gxxx node1 node2 noise data=dataname
Exxx node1 node2 noise data=dataname
.data dataname
+ pname1 pname2
+ freq1 noise1
+ freq2 noise2
+ ...
.enddata
```

The data form defines a basic frequency-noise table. The `.DATA` statement contains two parameters: frequency and noise to specify the noise value at each frequency point. The unit for frequency is hertz, and the unit for noise is A^2/Hz (for G current noise source) or V^2/Hz (for E voltage noise source).

Example

The following netlist shows a 1000 ohm resistor (g1) using a G-element. The g1noise element, placed in parallel with the g1 resistor, delivers the thermal noise expected from a resistor. The r1 resistor is included for comparison: The noise due to r1 should be the same as the noise due to g1noise.

```
* Resistor implemented using g-element
v1 1 0 1
r1 1 2 1k
g1 1 2 cur='v(1,2)*0.001'
g1noise 1 2
+ noise='4*1.3806266e-23*(TEMPER+273.15)*0.001'
rout 2 0 1meg
.ac lin 1 100 100
.noise v(2) v1 1
.end
```

Function Approximations for Distributed Devices

High-order rational function approximations constructed for distributed devices used at RF frequencies are obtained in the pole-residue form (also known as Foster canonical form). The popular method of recursive convolution also uses this form.

HSPICE supports the pole-residue form for its frequency-dependent controlled sources (G and E-elements). You can enter the pole-residue form directly without first converting to another form.

Foster Pole-Residue Form for Transconductance or Gain

The Foster pole-residue form for transconductance $G(s)$ or gain $E(s)$ has the form:

Equation 37

$$G(s) = k_0 + k_1s + \sum_{i=1}^N \left(\frac{A_i}{s-p_i} + \frac{A_i^*}{s-p_i^*} \right)$$

Where,

- k_0, k_1 are real constants
- residues A_i and poles p_i are complex numbers (or real as a special case of complex)
- asterisk (*) denotes the expression's complex conjugate

Advantages of Foster Form Modeling

The advantages of Foster canonical form modeling are:

- models high-order systems. It can theoretically model systems having infinite poles without numerical problems.
- equivalent to Laplace and Pole-zero models
- popular method of recursive convolution uses this form.

G and E-element Syntax

Transconductance G(s) form

```
Gxxx n+ n- FOSTER in+ in- k0 k1
+ (Re{A1}, Im{A1}) / (Re{p1}, Im{p1})
+ (Re{A2}, Im{A2}) / (Re{p2}, Im{p2})
+ (Re{A3}, Im{A3}) / (Re{p3}, Im{p3})
+ ...
```

Gain E(s) form

```
Exxx n+ n- FOSTER in+ in- k0 k1
+ (Re{A1}, Im{A1}) / (Re{p1}, Im{p1})
+ (Re{A2}, Im{A2}) / (Re{p2}, Im{p2})
+ (Re{A3}, Im{A3}) / (Re{p3}, Im{p3})
+ ...
```

In this syntax, parentheses, commas, and slashes are separators—they have the same meaning as a space. A pole-residue pair is represented by four numbers (real and imaginary part of the residue, then real and imaginary part of the pole).

You must make sure that $\text{Re}[p_i] < 0$; otherwise, the simulations will certainly diverge. Also, it is a good idea to assure passivity of the model (for an N-port admittance matrix Y , $\text{Re}\{Y\}$ should be positive-definite), or the simulation is likely to diverge).

Example

To represent a G(s) in the form,

Equation 38

$$G(s) = 0.001 + 1 \times 10^{-12} s + \frac{0.0008}{s + 1 \times 10^{10}} + \frac{(0.001 - j0.006)}{s - (-1 \times 10^8 + j1.8 \times 10^{10})} + \frac{(0.001 + j0.006)}{s - (-1 \times 10^8 - j1.8 \times 10^{10})}$$

You would input:

```
G1 1 0 FOSTER 2 0 0.001 1e-12
+(0.0004, 0)/(-1e10, 0) (0.001, -0.006)/(-1e8, 1.8e10)
```

Note: In the case of a real poles, half the residue value is entered, because it's essentially applied twice. In the above example, the first pole-residue pair is real, but we still write it as “A1/(s-p1)+A1/(s-p1)”; therefore, 0.0004 is entered rather than 0.0008.

Complex Signal Sources and Stimuli

To predict radio-frequency integrated circuit (RFIC) performance, some analyses require simulations that use representative RF signal sources. Among the representative sources available in HSPICE RF is the complex modulated RF source. Also known as the *Vector Modulated* source, it allows digital modulation of an RF carrier using in-phase and quadrature components created from a binary data stream.

Vector-Modulated RF (VMRF) Source

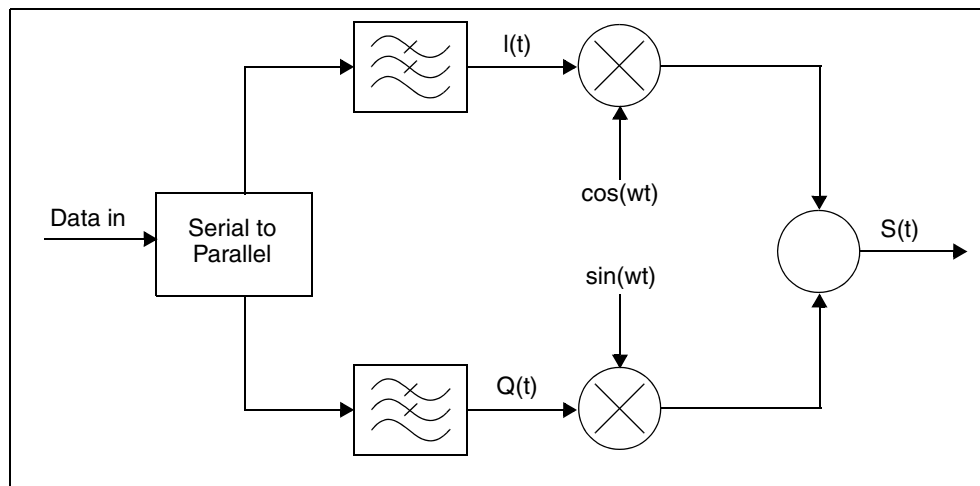
Digital RF waveforms are typically constructed by modulating an RF carrier with in-phase (I) and quadrature (Q) components. In HSPICE RF, this is accomplished using the Vector Modulated RF (VMRF) signal source.

The VMRF signal source function is supported both for independent voltage and current sources (V and I elements), and with controlled sources (E, F, G, and H elements).

- When used with independent sources, a baseband data stream can be input in binary or hexadecimal format, and the scheme used to divide the data into I and Q signals can be specified.
- With controlled VMRF sources, the modulating I and Q signals can be separately specified with other signal sources (such as a PWL source) and then used as control inputs into the VMRF source.

VMRF Implementation

The VMRF source is a mathematical implementation of the following block diagram:



The following equation calculates the time and frequency domain stimuli from the quadrature modulated signal sources:

Equation 39

$$s(t) = I(t) \cos(2\pi f_c t + \phi_0) - Q(t) \sin(2\pi f_c t + \phi_0)$$

The discrete ideal I (in-phase) and Q (quadrature) signal components are digital. Discrete values allow uniform scaling of the overall signal. HSPICE RF generates data streams for the I and Q signals based on interpreting the data string, breaking the data string into a binary representation, and then using the bit pairs to assign values for the I and Q data streams.

For BPSK (binary phase shift keying) modulation, the discrete signals are scaled so that $\sqrt{I^2 + Q^2} = 1$:

Data In	I Data	Q Data
0	$\frac{-1}{\sqrt{2}}$	$\frac{-1}{\sqrt{2}}$
1	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$

For QPSK (quadrature phase shift keying) modulation, the data stream is broken into bit pairs to form the correct I and Q values. This function is represented as the serial to parallel converter:

Data In	I Data	Q Data
00	$\frac{-1}{\sqrt{2}}$	$\frac{-1}{\sqrt{2}}$
01	$\frac{-1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$
10	$\frac{1}{\sqrt{2}}$	$\frac{-1}{\sqrt{2}}$
11	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$

To generate a continuous-time waveform, the VMRF source takes the resulting digital I and Q data streams and passes them through ideal filters. Rectangular and Nyquist (raised-cosine) filter options are available. The output waveforms are therefore band-limited according to the specified data rate.

Voltage and Current Source Elements

The V and I elements can include VMRF signal sources that you can use to generate BPSK and QPSK waveforms.

V and I Element Syntax

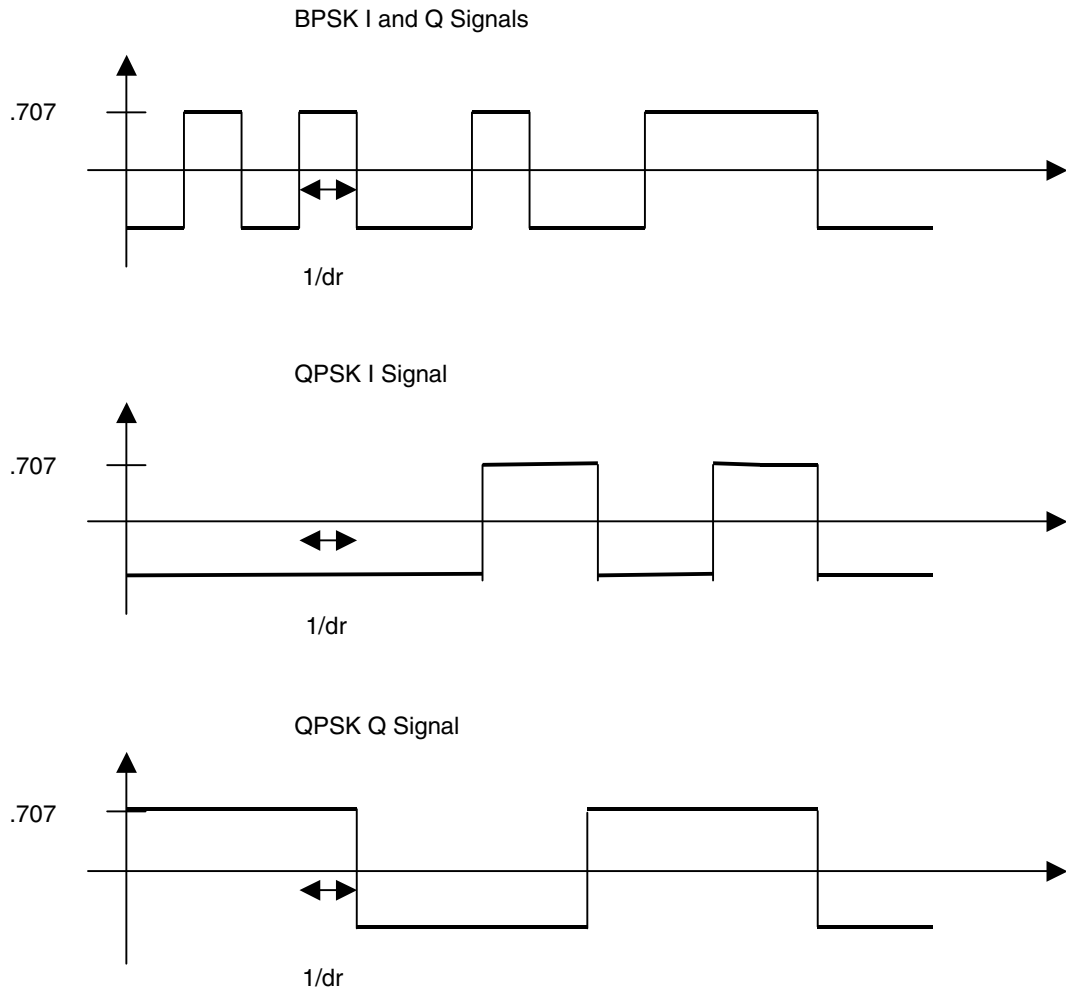
```
Vxxx n+ n- VMRF [ ( ] AMP=sa FREQ=fc PHASE=ph MOD=MOD
```



```
+ FILTER=FIL FILCOEF=filpar RATE=Rb BITSTREAM=data
+ [TRANFORHB=0/1] [ ] ]

Ixxx n+ n- VMRF [ ( ) AMP=sa FREQ=fc PHASE=ph MOD=MOD
+ FILTER=FIL FILCOEF=filpar RATE=Rb BITSTREAM=data
+ [TRANFORHB=0/1] [ ] ]
```

Parameter	Description
Vxxx	Independent voltage source.
Ixxx	Independent current source.
n+ n-	Positive and negative controlled source connecting nodes.
VMRF	Keyword that identifies and activates the Vector Modulated RF signal source.
AMP	Signal amplitude (in volts or amps).
FREQ	Carrier frequency in hertz. Set $fc=0.0$ to generate baseband I/Q signals. For harmonic balance analysis, the frequency spacing must coincide with the .HB TONES settings.
PHASE	Carrier phase (in degrees). If $fc=0.0$, <ul style="list-style-type: none"> ▪ $ph=0$ and baseband $I(t)$ is generated ▪ $ph=-90$ and baseband $q(t)$ is generated ▪ Otherwise, $s(t) = I(t)\cos(\phi_0) - Q(t)\sin(\phi_0)$
MOD	One of the following keywords identifies the modulation method used to convert a digital stream of information to $I(t)$ and $Q(t)$ variations: <ul style="list-style-type: none"> ▪ BPSK (binary phase shift keying) ▪ QPSK (quadrature phase shift keying)
FILTER	One of the following keywords identifies the method used to filter the I and Q signals before modulating the RF carrier signal: <ul style="list-style-type: none"> ▪ COS (raised cosine Nyquist filter) ▪ RECT (rectangular filtering)
FILCOEF	Filter parameter for the COS filter: $0 \leq filpar \leq 1$



The R_b parameter represents the data rate. The associated symbol rate represents how fast the I and Q data streams change. The period for each bit of data is:

Equation 40

$$T_b = \frac{1}{R_b}$$

The symbol rate depends on whether you select BPSK or QPSK modulation:

Chapter 10: Sources and Stimuli
HSPICE RF-Specific Sources and Stimuli

- For BPSK, the symbol rate is the same as the data rate:

$$R_s^{BPSK} = R_b$$

- For QPSK modulation, two bits are used to create each symbol so the symbol rate is half the data rate.

$$R_s^{QPSK} = \frac{R_b}{2}$$

The period for each symbol is computed as:

Equation 41

$$T_s = \frac{1}{R_s}$$

This value is necessary for establishing the characteristics of Nyquist filters.

The following equation calculates the raised cosine (COS) filter response:

Equation 42

$$H_{rc}(f) = \int_0^{T_s} T_s \cos^2 \left[\frac{\pi T_s}{2\alpha} \left(|f| - \frac{1-\alpha}{2T_s} \right) \right] \begin{matrix} |f| \leq \frac{1-\alpha}{2T_s} \\ \frac{1-\alpha}{2T_s} \leq |f| \leq \frac{1+\alpha}{2T_s} \\ |f| > \frac{1+\alpha}{2T_s} \end{matrix}$$

The VMRF signal source is designed primarily for TRAN and HB analyses, and can generate baseband signals. You can also specify DC and AC values as with any other HSPICE signal source:

- In DC analysis, the VMRF source is a constant DC source.
- In AC analysis, the source is a short or an open, unless you specify an AC value.
- In HB analysis, you must specify `.OPTION TRANFORHB` on the source statement line. The `TRANFORHB` option supports the VMRF signal source as well as the SIN, PULSE, and PWL sources.

The VMRF quadrature signal source typically involves an HF carrier signal that is modulated with a baseband signal on a much different time scale. You must set source and simulation control parameters appropriately to avoid time-consuming simulations in both the time and frequency domains.

E, F, G, and H Element Statements

For E, F, G, and H elements, you can use the VMRF function to modulate I(t) and Q(t) signals with a RF carrier signal. The I and Q signal are driven by PWL sources that might be generated by an external tool, such as MATLAB. The PWL source accepts a text file containing time and voltage (or current) pairs.

When the VMRF function is used with controlled sources, it is anticipated that the in-phase (I) and quadrature (Q) signals are not digital, but continuous-time analog signals. The VMRF function therefore includes no filtering, and merely serves to create the complex modulation on the RF carrier.

```
Exxx n+ n- [VCVS] VMRF [(] Iin+ Iin- Qin+ Qin- FREQ=fc
+ PHASE=ph [SCALE=A] [)]
```

```
Fxxx n+ n- [CCCS] VMRF [(] VI VQ FREQ=fc PHASE=ph
+ [SCALE=A] [)]
```

```
Gxxx n+ n- [VCCS] VMRF [(] Iin+ Iin- Qin+ Qin- FREQ=fc
+ PHASE=ph [SCALE=A] [)]
```

```
Hxxx n+ n- [CCVS] VMRF [(] VI VQ FREQ=fc PHASE=ph
+ [SCALE=A] [)]
```

Parameter	Description
Exxx	Voltage-controlled voltage source.
Fxxx	Current-controlled current source.
Gxxx	Voltage-controlled current source.
Hxxx	Current-controlled current source.
VCVS	Keyword for voltage-controlled voltage source.
CCCS	Keyword for current-controlled current source.
VCCS	Keyword for voltage-controlled current source.
CCVS	Keyword for current-controlled current source.
n+ n-	Positive and negative controlled source connecting nodes.

Parameter	Description
VMRF	Keyword that identifies and activates the vector-modulated RF signal source.
lin+ lin-	Node names for input I(t) signal.
Qin+ Qin-	Node names for input Q(t) signal.
VI VQ	
FREQ	Carrier frequency in Hertz. Set fc=0.0 to generate baseband I/Q signals.
PHASE	Carrier phase (in degrees). If fc=0.0, <ul style="list-style-type: none"> ▪ ph=0 and baseband I(t) is generated ▪ ph=-90 and baseband Q(t) is generated
SCALE	Unit-less amplitude scaling parameter.

Example

```
Emod1 inp1 inn1 VMRF It_plus It_neg Qt_plus Qt_neg
+ freq=1g phase=0 scale=1.5
```

SWEEPBLOCK in Sweep Analyses

You can use the `.SWEEPBLOCK` statement to specify complicated sweeps. Sweeps affect:

- DC sweep analysis
- Parameter sweeps around TRAN, AC, or HB analyses
- Frequency values used in AC or HBAC analyses

Currently, HSPICE supports the following types of sweeps:

- Linear sweeps: sweeps a variable over an interval with a constant increment. The syntax is one of the following:
 - *variable start stop increment*
 - *variable lin npoints start stop*

- Logarithmic sweeps: sweeps a variable over an interval. To obtain each point, this sweep multiplies the previous point by a constant factor. You can specify the factor as a number of points per decade or octave as in:
 - `variable dec npoints start stop`
 - `variable oct npoints start stop`
- Point sweeps: a variable takes on specific values that you specify as a list. The syntax is:

```
variable poi npoints p1 p2 ...
```

- Data sweeps: a `.DATA` statement identifies the swept variables and their values. The syntax is:

```
data=dataname
```

You can use the `SWEEPBLOCK` feature to combine linear, logarithmic, and point sweeps, which creates more complicated sets of values over which a variable is swept.

The `.TRAN`, `.AC`, `.DC`, and `.HB` commands can specify `SWEEPBLOCK=blockname` as a sweep instead of `LIN`, `DEC`, `OCT`, and so forth. Also, you can use `SWEEPBLOCK` for frequency sweeps with the `.AC`, `.HBAC`, `.PHASENOISE`, and `.HBNOISE` commands.

All commands that can use `SWEEPBLOCK` must refer to the `SWEEPBLOCK` sweep type. In addition, you must specify `SWEEPBLOCK` as one of the syntax types allowed for frequency sweeps with the `.HBAC`, `.PHASENOISE`, and `.HBNOISE` commands.

The following sections discuss these topics:

- [Input Syntax](#)
- [Using SWEEPBLOCK in a DC Parameter Sweep](#)
- [Using in Parameter Sweeps in TRAN, AC, and HB Analyses](#)
- [Limitations](#)

Input Syntax

The `SWEEPBLOCK` feature creates a sweep whose set of values is the union of a set of linear, logarithmic, and point sweeps. To specify the set of values in the `SWEEPBLOCK`, use the `.SWEEPBLOCK` command. This command also assigns a name to the `SWEEPBLOCK`. For example,

```
.SWEEPBLOCK swblockname sweepspec [sweepspec]
```

```
+ [sweepspec [...]]]
```

You can use SWEEPBLOCK to specify DC sweeps, parameter sweeps, AC and HBAC frequency sweeps, or wherever HSPICE accepts sweeps.

You can specify an unlimited number of *sweepspec* parameters. Each *sweepspec* can specify a linear, logarithmic, or point sweep by using one of the following forms:

```
start stop increment  
lin npoints start stop  
dec npoints start stop  
oct npoints start stop  
poi npoints p1 p2 ...
```

Example

The following example specifies a logarithmic sweep from 1 to 1e9 with more resolution from 1e6 to 1e7:

```
.sweepblock freqsweep dec 10 1 1g dec 1000 1meg 10meg
```

Using SWEEPBLOCK in a DC Parameter Sweep

To use the sweepblock in a DC parameter sweep, use the following syntax:

```
.DC sweepspec [sweepspec [sweepspec]]
```

Each *sweepspec* can be a linear, logarithmic, point, or data sweep, or it can be in the form:

```
variable SWEEPBLOCK=swblockname
```

The SWEEPBLOCK syntax sweeps the specified variable over the values contained in the SWEEPBLOCK.

Example

```
.dc vin1 0 5 0.1 vin2 sweepblock=vin2vals
```

Using in Parameter Sweeps in TRAN, AC, and HB Analyses

To use the sweepblock in parameter sweeps on .TRAN, .AC, and .HB commands, and any other commands that allow parameter sweeps, use the following syntax:

```
variable sweepblock=swblockname
```

Example 1

```
.tran 1n 100n sweep rout sweepblock=rvals
```


AC and HBAC analysis frequency sweeps can use `sweepblock=swblockname` to specify the frequency values.

Example 2

```
.ac sweepblock=freqsweep
```

Limitations

- You cannot use recursive `SWEEPBLOCK` specifications. That is, a `.SWEEPBLOCK` command cannot refer to another `SWEEPBLOCK` to build its list of values.
- You cannot include data sweeps in a `.SWEEPBLOCK` statement.

Clock Source with Random Jitter

In many applications involving signal integrity, RF, analog, and mixed-signal design, it is desirable to have an ideal signal source, such as a sine wave or square wave, that also includes a non-ideal random drift in phase (jitter). Such a source is useful for representing non-ideal clock sources during time-domain transient simulation. Modeling jitter in this way can be used to examine eye-diagram behavior or study how jitter may propagate through a circuit or system. A source with jitter is useful for representing non-ideal clock sources during time-domain transient simulation.

The `PERJITTER` option allows you to add periodic jitter to `SIN`, `COS` and `PULSE` time domain sources.

Syntax of SIN, COS, and Pulse Sources

The syntax of `SIN` source is:

```
Vxxx n+ n- SIN [(] vo va [freq [td [q [j ]]]] [)]  
+ [PERJITTER=val SEED=val]  
Ixxx n+ n- SIN [(] vo va [freq [td [q [j ]]]] [)]  
+ [PERJITTER=val SEED=val]
```

Parameter	Description
Vxxx	Independent voltage source.
Ixxx	Independent current source.

Parameter	Description
PERJITTER	Period jitter
PWL	Keyword for piecewise linear.
PWLFILE	Text file containing the PWL data consisting of time and voltage (or current) pairs. This file should not contain a header row, unless it is a comment. The PWL source data is obtained by extracting col1 and col2 from the file.
col1, [col2]	Time values are in col1 and voltage (or current) values are in col2. By default, col1=1 and col2=2.
R	Repeat function. When an argument is not specified, the source repeats from the beginning of the function. The argument repeated is the time, in seconds, which specifies the start point of the waveform being repeat. The repeat time must be less than the greatest time point in the file.
TD	Time delay, in seconds, of the PWL function.
options	Any standard V or I source options.

The sine wave behavior following the td time delay now becomes

Equation 43

$$V(t) + e^{-(t-t_d) \cdot \theta} = V_0 + V_a \cdot \left[\sin 2\pi f_0(t-t_d) + \frac{\pi}{180} \phi + \phi t - d_d \right]$$

The Syntax of COS source is:

```
Vxxx n+ n- COS [(] vo va [freq [td [q] [j]] []]
+ [PERJITTER=val SEED=val]
Ixxx n+ n- COS [(] vo va [freq [td [q] [j]] []]
+ [PERJITTER=val SEED=val]
```

The new cosine wave becomes

Equation 44

$$V(t) + e^{-(t-t_d) \cdot \theta} = V_0 + V_a \cdot \left[\cos 2\pi f_0(t-t_d + x(t)) + \frac{\pi}{180} \phi \right]$$

The syntax for the PULSE source is:

```
Vxxx n+ n- PU[LSE] [(] v1 v2 [td [tr] [tf] [pw] [per] []]
```

```
+ [PERJITTER=val SEED=val]
Ixxx n+ n- PU[LSE] [(]v1 v2 [td [tr] [tf] [pw] [per] [)]
+ [PERJITTER=val SEED=val]
```

The effect of jitter on the PULSE source results in random shifts of the rise and fall transitions that takes place at

RISE edge: $td + n \cdot T_0 \leq d \leq d + tr + n \cdot T_0$

FALL edge: $td + pw + n \cdot T_0 \leq d + pw + tf + n \cdot T_0$

The jitter effect is equivalent to introducing random shifts in the period T_0 consistent with the 1st order jitter model based on Period Jitter.

A Gaussian random number generator computes the time deviation $x(t)$ after each leading edge of the clock sources. For flexibility, the SEED parameter (integer) is supported for generating different random number sequences when different SEED integers are used for initialization. SEED does not set a fixed time deviation. It only changes the sequence of random samples. By HSPICE (Monte Carlo) convention, the default value for SEED is 1.

An interpretation of PERJITTER is to view it as causing each period of the PULSE/SIN/COS to be a random variable T_j , where period T_j will have a Gaussian distribution about the (mean) given period value of T_0 . The standard deviation of this Gaussian is the PERJITTER value (it is considered RMS period jitter), which results in a bell curve distribution centered about period T_0 .

Apply the following considerations when using PERJITTER:

- T_j should be forced to be between: $0 < T_j < 2 \cdot T_0$, since period cannot go negative, and the curve should be symmetrical.
- It is reasonable to require that $2 \cdot \text{PERJITTER} < T_0$. Otherwise, the jitter would result in very large period changes, and many would be $T_j < 0$.
- To establish a waveform reference, the first period should be T_0 (i.e., no jitter in the first period). This helps to establish good eye diagrams.

Example

As an alternative to using a Verilog-A module, you can generate a pseudo-random binary sequence (PRBS) using the following steps:

1. Construct your usual linear feedback shift register (LFSR) generator.
2. Construct a matching (T,tr,tf) PULSE source as a clock, but add jitter to it with the PERJITTER keyword.

Chapter 10: Sources and Stimuli

Specifying a Digital Vector File and Mixed Mode Stimuli

3. Use the PULSE source to gate (buffer) the LFSR output (through an ideal AND gate, VCCS, and so forth).

Specifying a Digital Vector File and Mixed Mode Stimuli

HSPICE and HSPICE RF input netlists support digital vector files. A VEC file consists of three parts:

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

To incorporate this information into your simulation, include the `.VEC` command in your netlist.

For paths to full demonstration files (`digstim.vec`—2 bit adder with PWL input, `m2bit.sp`, and `m2bit_v.sp`—same as `m2bit.sp`, except uses vector stimulus file, see [Benchmark Examples](#) in this user guide.

The following sections discuss these topics:

- [Commands in a Digital Vector File](#)
- [Vector Patterns](#)
- [Defining Tabular Data](#)
- [Waveform Characteristics](#)
- [Modifying Waveform Characteristics](#)
- [Using the Context-Based Control Option \(CBC\)](#)
- [Comment Lines and Line Continuations](#)
- [Parameter Usage](#)
- [Digital Vector File Example](#)

Commands in a Digital Vector File

For commands you can use in a VEC file, see [Digital Vector File Commands](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Vector Patterns

The *Vector Pattern Definition* section defines the vectors, their names, sizes, signal direction, sequence or order for each vector stimulus, and so on. A RADIX line must occur first and the other lines can appear in any order in this section. All keywords are case-insensitive.

Here is an example Vector Pattern Definition section:

```
; start of Vector Pattern Definition section
RADIX 1111 1111
VNAME A B C D E F G H
IO IIII IIII
TUNIT ns
```

These four lines are required and appear in the first lines of a VEC file:

- RADIX defines eight single-bit vectors.
- VNAME gives each vector a name.
- IO determines which vectors are inputs, outputs, or bidirectional signals. In this example, all eight are input signals.
- TUNIT indicates that the time unit for the tabular data to follow is in units of nanoseconds.

For additional information about these keywords, see [Defining Tabular Data on page 359](#).

Defining Tabular Data

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

The *Tabular Data* section defines (in tabular format) the values of the signals at specified times. Rows in the Tabular Data section must appear in chronological order because row placement carries sequential timing information. Its general format is:

```
time1 signal1_value1 signal2_value1 signal3_value1...
time2 signal1_value2 signal2_value2 signal3_value2...
time3 signal1_value3 signal2_value3 signal3_value3...
.
.
```

Chapter 10: Sources and Stimuli

Specifying a Digital Vector File and Mixed Mode Stimuli

Where *timex* is the specified time, and *signal_n_valuen* is the values of specific signals at specific points in time. The set of values for a particular signal (over all times) is a vector, which appears as a vertical column in the tabular data and vector table. The set of all *signal₁_valuen* constitutes one vector.

For example,

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

This example shows that:

- At 11.0 time units, the value for the first and fifth vectors is 1.
- At 20.0 time units, the first, second, fifth, and sixth vectors are 1.
- At 33.0 time units, the first, third, fifth, and eighth vectors are 1.

The following sections discuss these topics:

- [Input Stimuli](#)
- [Expected Output](#)
- [Verilog Value Format](#)
- [Periodic Tabular Data](#)

Input Stimuli

HSPICE converts each input signal into a PWL (piecewise linear) voltage source, and a series resistance. [Table 22](#) shows the legal states for an input signal. Signal values can have any of these legal states.

Table 22 Legal States for an Input Signal

State	Description
0	Drive to ZERO (gnd). Resistance set to 0.
1	Drive to ONE (vdd). Resistance set to 0.
Z, z	Floating to HIGH IMPEDANCE. A TRIZ statement defines resistance value.
X, x	Drive to ZERO (gnd). Resistance set to 0.
L	Resistive drive to ZERO (gnd). An OUT or OUTZ statement defines resistance value.

Table 22 Legal States for an Input Signal (Continued)

H	Resistive drive to ONE (vdd). An <code>OUT</code> or <code>OUTZ</code> statement defines resistance value.
U, u	Drive to ZERO (gnd). Resistance set to 0.

Expected Output

HSPICE converts each output signal into a `.DOUT` statement in the netlist. During simulation, HSPICE compares the actual results with the expected output vector(s). If the states are different, an error message appears. The legal states for expected outputs include the values listed in [Table 23](#).

Table 23 Legal States for an Output Signal

State	Description
0	Expect ZERO.
1	Expect ONE.
X, x	Don't care.
U, u	Don't care.
Z, z	Expect HIGH IMPEDANCE (don't care). Simulation evaluates Z, z as "don't care" because HSPICE cannot detect a high impedance state.

For example,

```
...
IO 0000
; start of tabular section data
11.0 1001
20.0 1100
30.0 1000
35.0 xx00
```

Where,

- The first line is a comment line because of the semicolon character.
- The second line expects the output to be 1 for the first and fourth vectors, while all others are expected to be low.

Chapter 10: Sources and Stimuli

Specifying a Digital Vector File and Mixed Mode Stimuli

- At 20 time units, HSPICE expects the first and second vectors to be high, and the third and fourth to be low.
- At 30 time units, HSPICE expects only the first vector to be high, and all others low.
- At 35 time units, HSPICE expects the output of the first two vectors to be “don’t care”; it expects vectors 3 and 4 to be low.

Verilog Value Format

HSPICE accepts Verilog-sized format to specify numbers; for example,

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

Where:

- *<size>* specifies the number of bits, in decimal format.
- *<base format>* indicates:
 - binary ('b or 'B)
 - octal ('o or 'O)
 - hexadecimal ('h or 'H).
- *<number>* values are combinations of the 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F characters. Depending on what base format you choose, only a subset of these characters might be legal.

You can also use unknown values (X) and high-impedance (Z) in the *<number>* field. An X or Z sets four bits in the hexadecimal base, three bits in the octal base, or one bit in the binary base.

If the most significant bit of a number is 0, X, or Z, HSPICE automatically extends the number (if necessary), to fill the remaining bits with 0, X, or Z, respectively. If the most significant bit is 1, HSPICE uses 0 to extend it.

For example,

```
4'b1111  
12'hABx  
32'bZ  
8'h1
```

This example specifies values for:

- 4-bit signal in binary
- 12-bit signal in hexadecimal

- 32-bit signal in binary
- 8-bit signal in hexadecimal

Equivalents of these lines in non-Verilog format, are:

```
1111
AB xxxx
ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ
0000 0001
```

Periodic Tabular Data

Tabular data is often periodic, so you do not need to specify the absolute time at every time point. When you specify the `PERIOD` statement, the Tabular Data section omits the absolute times. For more information, see [Defining Tabular Data on page 359](#).

For example, the `PERIOD` statement in the following sets the time interval to 10ns between successive lines in the tabular data. This is a shortcut when you use vectors in regular intervals throughout the entire simulation.

```
RADIX 1111 1111
VNAME A B C D E F G H
IO I III I III
TUNIT ns
PERIOD 10
; start of vector data section
1000 1000
1100 1100
1010 1001
```

Waveform Characteristics

The *Waveform Characteristics* section defines various attributes for signals, such as the rise or fall time, the thresholds for logic high or low, and so on. For example,

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

The waveform characteristics are based on a bit-mask. Where:

Chapter 10: Sources and Stimuli

Specifying a Digital Vector File and Mixed Mode Stimuli

- The `TRISE` (signal rise time) setting of 0.3ns applies to the first four vectors, but not to the last four.
- The example does not show how many bits are in each of the first four vectors, although the first vector is at least one bit.
- The fourth vector is four bits because F is hexadecimal for binary 1111.
- All bits of the fourth vector have a rise time of 0.3ns for the constant you defined in `TUNIT`. This also applies to `TFALL` (fall time), `VIH` (voltage for logic-high inputs), and `VIL` (voltage for logic-low inputs).

Modifying Waveform Characteristics

The `TDELAY`, `IDELAY`, and `ODELAY` statements define the delay time of the signal, relative to the absolute time of each row in the Tabular Data section.

- `TDELAY` applies to the input and output delay time of input, output, and bidirectional signals.
- `IDELAY` applies to the input delay time of input and bidirectional signals.
- `ODELAY` applies to the output delay time of output and bidirectional signals.

The `SLOPE` statement specifies the rise and fall times for the input signal. To specify the signals to which the *slope* applies, use a mask.

The `TFALL` statement sets an input fall time for specific vectors.

The `TRISE` statement sets an input rise time for specific vectors.

The `TUNIT` statement defines the time unit.

The `OUT` and `OUTZ` keywords are equivalent, and specify output resistance for each signal (for which the mask applies); `OUT` (or `OUTZ`) applies only to input signals.

The `TRIZ` statement specifies the output impedance, when the signal (for which the mask applies) is in tristate; `TRIZ` applies only to the input signals.

The `VIH` statement specifies the logic-high voltage for each input signal to which the mask applies.

The `VIL` statement specifies the logic-low voltage for each input signal to which the mask applies.

Similar to the `TDELAY` statement, the `VREF` statement specifies the name of the reference voltage for each input vector to which the mask applies. `VREF` applies only to input signals.

Similar to the `TDELAY` statement, the `VTH` statement specifies the logic threshold voltage for each output signal to which the mask applies. The threshold voltage determines the logic state of output signals for comparison with the expected output signals.

The `VOH` statement specifies the logic-high voltage for each output signal to which the mask applies.

The `VOL` statement specifies the logic-low voltage for each output signal to which the mask applies.

Using the Context-Based Control Option (CBC)

The `OPTION CBC` (Context-Based Control) specifies the direction of bidirectional signals. A bidirectional signal is an input if its value is 0, 1, or Z; conversely, a bidirectional signal is an output if its value is H, L, U, or X.

For example:

```
RADIX 1 1 1
IO I O B
VNAME A Z B
OPTION CBC
10.0 0 X L
20.0 1 1 H
30.0 1 0 Z
```

This example sets up three vectors, named `A`, `Z`, and `B`. Vector `A` is an input, vector `Z` is an output, and vector `B` is a bidirectional signal (defined in the `IO` statement).

The `OPTION CBC` line turns on context-based control. The next line sets vector `A` to a logic-low at 10.0 ns, and vector `Z` is “do not care.” Because the `L` value is under vector `B`, HSPICE expects a logic-low output.

At 20 ns, vector `A` transitions high, and the expected outputs at vectors `Z` and `B` are high. Finally, at 30 ns, HSPICE expects vector `Z` to be low, vector `B` changes from an output to a high-impedance input, and vector the `A` signal does not change.

Comment Lines and Line Continuations

Any line in a VEC file that begins with a semicolon (;) is a comment line. Comments can also start at any point along a line. HSPICE ignores characters after a semicolon. For example,

```
; This is a comment line  
radix 1 1 4 1234 ; This is a radix line
```

As in netlists, any line in a VEC file that starts with a plus sign (+) is a continuation from the previous line.

Parameter Usage

You can use `.PARAM` statements with some `VEC` statements when you run HSPICE. These `VEC` statements fall into the three groups, which are described in the following sections. No other `VEC` statements but those identified here support `.PARAM` statements.

First Group

- PERIOD
- TDELAY
- IDELAY
- ODELAY
- SLOPE
- TRISE
- TFALL

For these statements, the `TUNIT` statement defines the time unit. If you do not include a `TUNIT` statement, the default time unit value is ns.

Do not specify absolute unit values in a `.PARAM` statement. For example, if in your netlist:

```
.param myperiod=10ns           $ 'ns' makes this incorrect
```

And in your VEC file:

```
tunit ns  
period myperiod
```

What you wanted for the time period is 10ns; however, because you specified absolute units, 1e-8ns is the value used. In this example, the correct form is:

```
.param myperiod=10
```

Second Group

- OUT or OUTZ
- TRIZ

In these statements, the unit is ohms.

- If you do not include an OUT (or OUTZ) statement, the default is 0.
- If you do not include a TRIZ statement, the default is 1000M.

The .PARAM definition for this group follows the HSPICE syntax.

For example, if in your netlist:

```
.param myout=10                $ means 10 ohm
.param mytriz= 10Meg           $ means 10,000,000 ohm, don't
$ confuse Meg with M, M means 0.001
```

And in your VEC file:

```
out myout
triz mytriz
```

Then, HSPICE returns 10 ohm for OUT and 10,000,000 ohm for TRIZ.

Third Group

- VIH
- VIL
- VOH
- VOL
- VTH

In these statements, the unit is volts.

- If you do not include an VIH statement, the default is 3.3.
- If you do not include a VIL statement, the default is 0.0.
- If you do not include a VOH statement, the default is 2.64.

Chapter 10: Sources and Stimuli

Specifying a Digital Vector File and Mixed Mode Stimuli

- If you do not include an VOL statement, the default is 0.66.
- If you do not include an VTH statement, the default is 1.65.

Digital Vector File Example

```
; specifies # of bits associated with each vector
radix 1 2 444
;*****
; defines name for each vector. For multi-bit vectors,
; innermost [] provide the bit index range, MSB:LSB
vname v1 va[[1:0]] vb[12:1]
;actual signal names: v1, va[0], va[1], vb1, vb2, ... vb12
;*****
; defines vector as input, output, or bi-directional
io i o bbb
; defines time unit
tunit ns
;*****
; vb12-vb5 are output when 'v1' is 'high'
enable v1 0 0 FF0
; vb4-vb1 are output when 'v1' is 'low'
enable ~v1 0 0 00F
;*****
; all signals have a delay of 1 ns
; Note: do not put the unit (such as ns) here again.
; HSPICE multiplies this value by the specified 'tunit'.
tdelay 1.0
; va[1] and va[0] signals have 1.5ns delays
tdelay 1.5 0 3 000
;*****
; specify input rise/fall times (if you want different
; rise/fall times, use the trise/tfall statement.)
; Note: do not put the unit (such as ns) here again.
; HSPICE multiplies this value by the specified 'tunit'.
slope 1.2
;*****
; specify the logic 'high' voltage for input signals
vih 3.3 1 0 000
vih 5.0 0 0 FFF
; to specify logic low, use 'vil'
;*****
```

Chapter 10: Sources and Stimuli
Specifying a Digital Vector File and Mixed Mode Stimuli

```
; 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
```

Chapter 10: Sources and Stimuli

Specifying a Digital Vector File and Mixed Mode Stimuli

Parameters and Functions

Describes how to use parameters within HSPICE and HSPICE RF netlists.

Parameters are similar to the variables used in most programming languages. Parameters hold a value that you assign when you create your circuit design or that the simulation calculates based on circuit solution values. Parameters can store static values for a variety of quantities (resistance, source voltage, rise time, and so on). You can also use them in sweep or statistical analysis.

HSPICE ships numerous examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

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

These topics are covered in the following sections:

- [Using Parameters in Simulation \(.PARAM\)](#)
- [Using Algebraic Expressions](#)
- [Built-In Functions and Variables](#)
- [Parameter Scoping and Passing](#)

Using Parameters in Simulation (.PARAM)

Defining Parameters

Parameters in HSPICE are names that you associate with numeric values. (See [Assigning Parameters on page 373](#).) You can use any of the methods described in [Table 24](#) to define parameters.

Chapter 11: Parameters and Functions
Using Parameters in Simulation (.PARAM)

Note: A .PARAM statement with no definition is illegal.

Table 24 .PARAM Statement Syntax

Parameter	Description
Simple assignment	.PARAM <i>SimpleParam</i> =1e-12
Algebraic definition	<ul style="list-style-type: none"> ▪ .PARAM <i>AlgebraicParam</i>='SimpleParam*8.2' ▪ SimpleParam excludes the output variable. ▪ You can also use algebraic parameters in .PRINT and .PROBE statements. For example: ▪ .PRINT <i>AlgebraicParam</i>=par('algebraic expression') ▪ You can use the same syntax for .PROBE statements. See Using Algebraic Expressions on page 379.
User-defined function	.PARAM <i>MyFunc</i> (<i>x</i> , <i>y</i>)='Sqrt((<i>x</i> * <i>x</i>)+(<i>y</i> * <i>y</i>))'
Character string definition	.PARAM <i>paramname</i> =str('string')
Subcircuit default	.SUBCKT <i>SubName ParamDefName= Value</i> str('string') .MACRO <i>SubName ParamDefName= Value</i> str('string')
Subcircuit call instance	Xxxx <i>nodename1</i> ... <i>nodenamen</i> + <i>SubName</i> + <i>ParamDefName=Value</i> str('string')
Predefined analysis function	.PARAM <i>mcVar</i> =AGAUSS(1.0,0.1)
.MEASURE statement	.MEASURE [DC AC TRAN] <i>result</i> TRIG ... + TARG ... [GOAL= <i>val</i>] [MINVAL= <i>val</i>] + [WEIGHT= <i>val</i>] [MeasType] [MeasParam] (See Specifying User-Defined Analysis (.MEASURE) on page 424 .)
.PRINT .PROBE	.PRINT .PROBE + <i>outParam=Par_Expression</i>

A parameter definition in HSPICE always uses the last value found in the input netlist (subject to local versus global parameter rules). These definitions assign a value of 3 to the *DupParam* parameter.

```
.PARAM DupParam=1
...
.PARAM DupParam=3
```

HSPICE assigns 3 as the value for all instances of DupParam, including instances that are earlier in the input than the .PARAM DupParam=3 statement.

All parameter values in HSPICE are IEEE double floating point numbers. The parameter resolution order is:

1. Resolve all literal assignments.
2. Resolve all expressions.
3. Resolve all function calls.

Table 25 shows the parameter passing order.

Table 25 Parameter Passing Order

.OPTION PARHIER=GLOBAL	.OPTION PARHIER=LOCAL
Analysis sweep parameters	Analysis sweep parameters
.PARAM statement (library)	.SUBCKT call (instance)
.SUBCKT call (instance)	.SUBCKT definition (symbol)
.SUBCKT definition (symbol)	.PARAM statement (library)

Assigning Parameters

You can assign the following types of values to parameters:

- Constant real number
- Algebraic expression of real values
- Predefined function
- Function that you define
- Circuit value
- Model value

To invoke the algebraic processor, enclose a complex expression in single quotes. A simple expression consists of one parameter name.

The parameter keeps the assigned value, unless:

- A later definition changes its value, or
- An algebraic expression assigns a new value during simulation. HSPICE does not warn you, if it reassigns a parameter.

Note: Parameters you define using `.PARAM` statements are constants; they cannot vary as a function of simulation time. However, you can use elements such as resistors and capacitors whose values can vary as a function of time by using the keyword `TIME`, i.e.,

```
Ctime 1 2 q='1p*time'
```

Example: Modeling an eFuse

You can model an electrically programmable eFUSE device as follows. Instantiate an eFUSE as a subcircuit and pass a parameter that determines whether the eFUSE is “blown” or intact:

```
.subckt efuse in out blown=0
Rfuse in out r='2*(1-blown)+100e6*blown'
.ends efuse
```

If `blown=0`, then the fuse is intact (2 ohms). If `blown=1` then the fuse is blown and you get the much higher resistance of 100 meg. To use the eFUSE, instantiate it with a subcircuit call:

```
xefuse1 in out efuse blown=0
```

Alternately, you can control the eFUSE with a parameter setting:

```
.param blown=1
x1 in out efuse
```

Inline Parameter Assignments

To define circuit values, use a direct algebraic evaluation:

```
r1 n1 0 R='1k/sqrt(HERTZ)' $ Resistance for frequency
```

Parameters in Output

To use an algebraic expression as an output variable in a `.PRINT`, `.PROBE` or `.MEASURE` statement, use the `PAR` keyword. In addition to using quotations, you must define the parameter inside the `PAR('...')` statement for output. HSPICE issues an error message if the parentheses do not surround the

quotes or if you use the keyword *incorrectly* as in `PAR=` instead of `PAR('...')`.

Example

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

Reusing the PAR(...) Output as Input to Other Elements

You can use the `par(...)` output as the input voltage to another elements.

For example:

```
.print tran v(5) par('5*cos(6.28*v(10)*v(5)*k/360)')
```

You can use the signal either as the input to another element in the same simulation (see Example 1), or you can save the signal and use it as input in another simulation (see Examples 2 and 3). You can use the definitions in `print` or `probe` output statements only in output statements. They cannot be referred by any other definitions. For example, you can use the E-element:

Example 1

```
e1 in 0 vol='5*cos(6.28*v(10)*v(5)*k/360)'
```

Example 2: Then, you can uses the node 'in' as the input to the other element in the same netlist, as shown in this example.

```
M1 dr in src subr pch w=2u l=1u
.subckt inv vdd 0 A B
  M1 A B vdd vdd pch w=6u l=1u
  M2 A B 0 0 nch w=3u l=1u
.ends inv
x1 vdd 0 in out inv
```

Example 3: You can use `.STIM` statements to save the signal from the first simulation in order to create a PWL source.

```
.stim tran pwl filename=test1 vsrc[0]=v(in) node1=A node2=B
from=0.0ns to=10ns
+ npoints=100
.stim tran pwl filename=test2 vsrc[1]=v(in) node1=C node2=D
from=0.0ns to=10ns
+ npoints=50
```

In this example, the `.STIM` command creates two stimulus files named `test1.pwl0_tr0` and `test2.pwl2_tr0`. Each has a voltage source: one named `vsrc[0]`, applied between nodes A and B, and one named `vsrc[1]`, applied between nodes C and D. The stimulus files have a PWL source function

based on the voltage of node 'in' during the time 0.0ns to 10ns with 50 points (for vsrc[0]) and 100 points (for vsrc[1]).

Contents of the test1.pwl0_tr0 file:

```
vsrc[0] A B PWL
+ 0.          5.0000
+ 200.00000p  2.2114
+ 400.00000p  2.4666
+ 600.00000p -362.1421m
.....
```

Contents of the test2.pwl1_tr0 file:

```
vsrc[1] C D PWL
+ 0.          5.0000
+ 100.00000p -1.8008
+ 200.00000p -3.2748
+ 300.00000p -1.3264
.....
```

You can use the PWL files generated from the .STIM commands as inputs to another simulation.

Load the signal in WaveView and then export the (x,y) data of the signal in text format by selecting **File > Save > Plotfiles**. You need to edit the data so it looks like a PWL source by adding a source definition and line continuation characters.

User-Defined Function Parameters

You can define a function that is similar to the parameter assignment, but you cannot nest the functions more than two deep.

- An expression can contain parameters that you did not define.
- A function must have at least one argument, and can have up to 20 (and in many cases, more than 20) arguments.
- You can redefine functions.

The format of a function is:

```
funcname1(arg1[,arg2...])=expression1
+ [funcname2(arg1[,arg2...])=expression2] off
```

Parameter	Description
funcname	Specifies the function name. This parameter must be distinct from array names and built-in functions. In subsequently defined functions, you must previously define all embedded functions.
arg1, arg2	Specifies variables used in the expression.
off	voids all user-defined functions.

Example

```
.PARAM f(a,b)=POW(a,2)+a*b g(d)=SQRT(d)
+ h(e)=e*f(1,2)-g(3)
```

Using Parameter Functions to Evaluate Expressions Containing Dynamic Signals

When you use a parameter function to evaluate expression that contain dynamic signals, you must provide a user-defined function. Otherwise, you get unexpected results and generate a warning message.

For example, in the following netlist, a node-voltage (dynamic signal) is once at $t=0$ and keeps this value for the entire simulation. Because the value is only evaluated at the beginning of the simulation and not at each time step, the results are not as expected.

```
* Test Case
VIN D 0 pw1 0 1 10n 5
.param aaa='v(D)-v(0)'
.tran 0.1n 20n
vdd vdd 0 5
R1 vdd 0 'v(D) - v(0)'
R2 vdd 0 'aaa'
.options post
.probe i(*)
.tran 0.1n 20n
.end
```

If you replace the constant parameter assignment with user-defined parameter functions, the function `.param aaa(x)='x'` causes HSPICE to evaluate the

parameter at each time point during the simulation to give you the expected results.

```
* Test Case
VIN D 0 pw1 0 1 10n 5
.param aaa(x)='x'
.tran 0.1n 20n
vdd vdd 0 5
R1 vdd 0 'V(D) - V(0)'
R2 vdd 0 'aaa(V(D,0))'
.options post
.probe i(*)
.end
```

Predefined Analysis Function

HSPICE includes specialized analysis types, such as Optimization and Monte Carlo, that require a way to control the analysis.

Measurement Parameters

.MEASURE statements produce a *measurement* parameter. The rules for measurement parameters are the same as for standard parameters, except that a .MEASURE statement defines measurement parameters, but a .PARAM statement does not define them. For a description of the .MEASURE statement, see [Specifying User-Defined Analysis \(.MEASURE\) on page 424](#).

.PRINT and .PROBE Parameters

.PRINT, and .PROBE statements in HSPICE produce a *print* parameter. The rules for print parameters are the same as the rules for standard parameters, except that you define the parameter directly in a .PRINT or .PROBE statement, not in a .PARAM statement

For more information about the .PRINT or .PROBE statements, see [Displaying Simulation Results on page 398](#).

Multiply Parameter

The most basic subcircuit parameter in HSPICE is the M (multiply) parameter. For a description of this parameter, see [M \(Multiply\) Parameter on page 110](#).

Using Algebraic Expressions

Note: Synopsys HSPICE uses double-precision numbers (15 digits) for expressions, user-defined parameters, and sweep variables.

In HSPICE, an algebraic expression, with quoted strings, can replace any parameter in the netlist.

In HSPICE, you can then use these expressions as output variables in `.PRINT` statements. Algebraic expressions can expand your options in an input netlist file.

Some uses of algebraic expressions are:

- Parameters:

```
.PARAM x='y+3'
```

- Functions:

```
.PARAM rho(leff,weff)='2+*leff*weff-2u'
```

- Algebra in elements:

```
R1 1 0 r='ABS(v(1)/i(m1))+10'
```

- Algebra in `.MEASURE` statements:

```
.MEAS vmax MAX V(1)
.MEAS imax MAX I(q2)
.MEAS ivmax PARAM='vmax*imax'
```

- Algebra in output statements:

```
.PRINT conductance=PAR('i(m1)/v(22)')
```

Output requires uses the following basic syntax for algebraic expressions:

```
PAR('algebraic expression')
```

In addition to using quotations, you must define the expression inside the `PAR('...')` statement for output. HSPICE issues an error message if the

parentheses do not surround the quotes. The continuation character for quoted parameter strings is a double backslash (\\).

Built-In Functions and Variables

In addition to simple arithmetic operations (+, -, *, /), use the built-in functions listed in [Table 26](#) and the variables listed in [Table 25 on page 373](#) in HSPICE expressions.

Table 26 Synopsys HSPICE Built-in Functions

HSPICE Form	Function	Class	Description
sin(x)	sine	trig	Returns the sine of x (radians)
cos(x)	cosine	trig	Returns the cosine of x (radians)
tan(x)	tangent	trig	Returns the tangent of x (radians)
asin(x)	arc sine	trig	Returns the inverse sine of x (radians)
acos(x)	arc cosine	trig	Returns the inverse cosine of x (radians)
atan(x)	arc tangent	trig	Returns the inverse tangent of x (radians)
sinh(x)	hyperbolic sine	trig	Returns the hyperbolic sine of x (radians)
cosh(x)	hyperbolic cosine	trig	Returns the hyperbolic cosine of x (radians)
tanh(x)	hyperbolic tangent	trig	Returns the hyperbolic tangent of x (radians)
abs(x)	absolute value	math	Returns the absolute value of x: x
sqrt(x)	square root	math	Returns the square root of the absolute value of x: sqrt(-x)=-sqrt(x)
pow(x,y)	absolute power	math	Returns the value of x raised to the integer part of y: $x^{(\text{integer part of } y)}$

Table 26 Synopsys HSPICE Built-in Functions (Continued)

HSPICE Form	Function	Class	Description
pwr(x,y)	signed power	math	Returns the absolute value of x, raised to the y power, with the sign of x: (sign of x) x ^y
x**y	power		If x<0, returns the value of x raised to the integer part of y. If x=0, returns 0. If x>0, returns the value of x raised to the y power.
log(x)	natural logarithm	math	Returns the natural logarithm of the absolute value of x, with the sign of x: (sign of x)log(x)
log10(x)	base 10 logarithm	math	Returns the base 10 logarithm of the absolute value of x, with the sign of x: (sign of x)log ₁₀ (x)
exp(x)	exponential	math	Returns e, raised to the power x: e ^x
db(x)	decibels	math	Returns the base 10 logarithm of the absolute value of x, multiplied by 20, with the sign of x: (sign of x)20log ₁₀ (x)
int(x)	integer	math	Returns the integer portion of x (which ignores the fractional portion of the number).
nint(x)	integer	math	Rounds x up or down, to the nearest integer.
sgn(x)	return sign	math	<ul style="list-style-type: none"> ▪ Returns -1 if x is less than 0. ▪ Returns 0 if x is equal to 0. ▪ Returns 1 if x is greater than 0.
sign(x,y)	transfer sign	math	Returns the absolute value of x, with the sign of y: (sign of y) x
def(x)	parameter defined	control	<ul style="list-style-type: none"> ▪ Returns 1 if parameter x is defined. ▪ Returns 0 if parameter x is not defined.
min(x,y)	smaller of two args	control	Returns the numeric minimum of x and y

Table 26 Synopsys HSPICE Built-in Functions (Continued)

HSPICE Form	Function	Class	Description
max(x,y)	larger of two args	control	Returns the numeric maximum of x and y
val(<i>element</i>)	get value	various	Returns a parameter value for a specified element. For example, val(r1) returns the resistance value of the <i>r1</i> resistor.
val(<i>element.parameter</i>)	get value	various	Returns a value for a specified parameter of a specified element. For example, val(rlload.temp) returns the value of the <i>temp</i> (temperature) parameter for the <i>rlload</i> element.
val(<i>model_type: model_name.model_param</i>)	get value	various	Returns a value for a specified parameter of a specified model of a specific type. For example, val(nmos:mos1.rs) returns the value of the <i>rs</i> parameter for the <i>mos1</i> model, which is an nmos model type. CMI models (Level 54 and greater) <i>do not</i> support this form. See Measuring the Value of MOSFET Model Card Parameters for an example and details.
valm(<i>elem_name.model_param</i>)	get value		Returns a value for a specified model parameter of a specified element. For example, valm(m1.vth0) returns the value of vth0 parameter of the model card that uses m1. valm() supports only vth0, lmin, lmax, wmin, wmax, lref, wref, xl, dl, dell, xw, dw, delw, scalm, lmlt, wmlt and level54, level57 and level70. See Measuring the Value of MOSFET Model Card Parameters for an example and details.
valp(<i>parameter</i>)	get value		Returns a value for a specified parameter. The parameter can only be a named parameter as defined in a subcircuit. For example: <pre>.meas tran asdf param='valp(x1/zzz.p1)'</pre> An expression is not permitted.
lv(<i>Element</i>) or lx(<i>Element</i>)	element templates	various	Returns various element values during simulation. See Element Template Output (HSPICE Only) on page 423 for more information.

Table 26 Synopsys HSPICE Built-in Functions (Continued)

HSPICE Form	Function	Class	Description
v(Node), i(Element)...	circuit output variables	various	Returns various circuit values during simulation. See DC and Transient Output Variables on page 407 for more information.
cond ?x : y	ternary operator		Returns x if <i>cond</i> is not zero. Otherwise, returns y. .param z= 'condition ? x:y'
<	relational operator (less than)		Returns 1 if the left operand is less than the right operand. Otherwise, returns 0. .para x=y<z (y less than z)
<=	relational operator (less than or equal)		Returns 1 if the left operand is less than or equal to the right operand. Otherwise, returns 0. .para x=y<=z (y less than or equal to z)
>	relational operator (greater than)		Returns 1 if the left operand is greater than the right operand. Otherwise, returns 0. .para x=y>z (y greater than z)
>=	relational operator (greater than or equal)		Returns 1 if the left operand is greater than or equal to the right operand. Otherwise, returns 0. .para x=y>=z (y greater than or equal to z)
==	equality		Returns 1 if the operands are equal. Otherwise, returns 0. .para x=y==z (y equal to z)
!=	inequality		Returns 1 if the operands are not equal. Otherwise, returns 0. .para x=y!=z (y not equal to z)
&&	Logical AND		Returns 1 if neither operand is zero. Otherwise, returns 0. .para x=y&&z (y AND z)

Table 26 Synopsys HSPICE Built-in Functions (Continued)

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.
			<code>.para x=y z (y OR z)</code>

Example

```
.parameters p1=4 p2=5 p3=6
r1 1 0 value='p1 ? p2+1 : p3'
```

HSPICE reserves the variable names listed in [Table 27 on page 384](#) for use in elements, such as E, G, R, C, and L. You can use them in expressions, but you cannot redefine them; for example, this statement would be illegal:

```
.param temper=100
```

Table 27 Synopsys HSPICE Special Variables

HSPICE Form	Function	Class	Description
time	current simulation time	control	Uses parameters to define the current simulation time, during transient analysis.
temper	current circuit temperature	control	Uses parameters to define the current simulation temperature, during transient/temperature analysis. You can use the HSPICE simulation temperature in an equation by using the temper variable parameter. For example: <code>.temp 20 50 100</code> <code>.par x="temper/2"</code> <code>v0 1 0 1</code> <code>r0 1 0 r=x</code>
hertz	current simulation frequency	control	Uses parameters to define the frequency, during AC analysis.

Parameter Scoping and Passing

If you use parameters to define values in subcircuits, you need to create fewer similar cells, to provide enough functionality in your library. You can pass circuit parameters into hierarchical designs, and assign different values to the same parameter within individual cells, when you run simulation.

For example, if you use parameters to set the initial state of a latch in its subcircuit definition, then you can override this initial default in the instance call. You need to create only one cell, to handle both initial state versions of the latch.

You can also use parameters to define the cell layout. For example, you can use parameters in a MOS inverter, to simulate a range of inverter sizes, with only one cell definition. Local instances of the cell can assign different values to the size parameter for the inverter.

In HSPICE, you can also perform Monte Carlo analysis or optimization on a cell that uses parameters.

How you handle hierarchical parameters depends on how you construct and analyze your cells. You can construct a design in which information flows from the top of the design, down into the lowest hierarchical levels.

- To centralize the control at the top of the design hierarchy, set *global* parameters.
- To construct a library of small cells that are individually controlled from within, set *local* parameters and build up to the block level.

This section describes the scope of parameter names, and how HSPICE resolves naming conflicts between levels of hierarchy.

Library Integrity

Integrity is a fundamental requirement for any symbol library. Library integrity can be as simple as a consistent, intuitive name scheme, or as complex as libraries with built-in range checking.

Library integrity might be poor if you use libraries from different vendors in a circuit design. Because vendors do not standardize names of circuit parameters, two components can include the same parameter name for different functions. For example, one vendor might build a library that uses the name `Tau` as a parameter to control one or more subcircuits in their library.

Another vendor might use τ to control a different aspect of their library. If you set a global parameter named τ to control one library, you also modify the behavior of the second library, which might not be the intent.

If the scope of a higher-level parameter is global to all subcircuits at lower levels of the design hierarchy, higher-level definitions override lower-level parameter values with the same names. The scope of a lower-level parameter is local to the subcircuit where you define the parameter (but global to all subcircuits that are even lower in the design hierarchy). Local scoping rules in HSPICE prevent higher-level parameters from overriding lower-level parameters of the same name.

Reusing Cells

Parameter name problems also occur if different groups collaborate on a design. Global parameters prevail over local parameters, so all circuit designers must learn the names of all parameters, even those used in sections of the design for which they are not responsible. This can lead to a large investment in standard libraries. To avoid this situation, use local parameter scoping, to encapsulate all information about a section of a design, within that section.

Creating Parameters in a Library

To ensure that the input netlist includes critical, user-supplied parameters when you run simulation, you can use “illegal defaults”—that is, defaults that cause the simulator to abort if you do not supply overrides for the defaults.

If a library cell includes illegal defaults, you must provide a value for each instance of those cells. If you do not, the simulation aborts.

For example, you might define a default MOSFET width of 0.0. HSPICE aborts because MOSFET models require this parameter.

Example 1

```
* Subcircuit default definition
.SUBCKT Inv A Y Wid=0 $ Inherit illegal values by default
mp1 NodeList Model L=1u W='Wid*2'
mn1 NodeList Model L=1u W=Wid
.ENDS
```

```
* Invoke symbols in a design
x1 A Y1 Inv          $ Bad! No widths specified
x2 A Y2 Inv Wid=1u  $ Overrides illegal value for Width
```

This simulation aborts on the `x1` subcircuit instance because you never set the required `Wid` parameter on the subcircuit instance line. The `x2` subcircuit simulates correctly. Additionally, the instances of the `Inv` cell are subject to accidental interference because of exposure of the `Wid` global parameter is expose outside the domain of the library. Anyone can specify an alternative value for the parameter, in another section of the library or the circuit design. This might prevent the simulation from catching the condition on `x1`.

Example 2

In this example, the name of a global parameter conflicts with the internal library parameter named `wid`. Another user might specify such a global parameter, in a different library. In this example, the user of the library has specified a different meaning for the `Wid` parameter, to define an independent source.

```
.Param Wid=5u          $ Default Pulse Width for source
v1 Pulsed 0 Pulse ( 0v 5v 0u 0.1u 0.1u Wid 10u )
...
* Subcircuit default definition
.SUBCKT Inv A Y Wid=0    $ Inherit illegals by default
mp1 NodeList Model L=1u W='Wid*2'
mn1 NodeList Model L=1u W=Wid
.Ends
* Invoke symbols in a design
x1 A Y1 Inv             $ Incorrect width!
x2 A Y2 Inv Wid=1u     $ Incorrect! Both x1 and x2
$ simulate with mp1=10u and
$ mn1=5u instead of 2u and 1u.
```

Under global parameter scoping rules, simulation succeeds, but incorrectly. HSPICE does not warn that the `x1` inverter has no assigned width because the global parameter definition for `Wid` overrides the subcircuit default.

Note: Similarly, sweeping with different values of *Wid* dynamically changes both the *Wid* library internal parameter value, and the pulse width value to the *Wid* value of the current sweep.

In global scoping, the highest-level name prevails, when resolving name conflicts. Local scoping uses the lowest-level name.

When you use the parameter inheritance method, you can specify to use local scoping rules.

When you use local scoping rules, the Example 2 netlist correctly aborts in x1 for W=0 (default Wid=0, in the .SUBCKT definition, has higher precedence, than the .PARAM statement). This results in the correct device sizes for x2. This change can affect your simulation results, if you intentionally or accidentally create a circuit such as the second one shown above.

You can use an alternative to the test of the width in the Example 2 netlist: Use .OPTION DEFW to achieve a limited version of library integrity. This option sets the default width for all MOS devices during a simulation. Part of the definition is still in the top-level circuit, so this method can still make unwanted changes to library values, without notification from the HSPICE simulator.

[Table 28](#) compares the three primary methods to configure libraries, to achieve required parameter checks for default MOS transistor widths.

Table 28 Methods for Configuring Libraries

Method	Parameter Location	Pros	Cons
Local	On a .SUBCKT definition line	Protects library from global circuit parameter definitions, unless you override it. Single location for default values.	
Global	At the global level and on .SUBCKT definition lines	Works with all HSPICE versions.	An indiscreet user, another vendor assignment, or the intervening hierarchy can change the library. Cannot override a global value at a lower level.
Special	.OPTION DEFW statement	Simple to do.	Third-party libraries, or other sections of the design, might depend on .OPTION DEFW.

String Parameter (HSPICE Only)

HSPICE uses a special delimiter to identify string and double parameter types. The single quotes ('), double quotes ("), or curly brackets ({}) do not work for these kinds of delimiters. Instead, use the `sp1=str('string')` keyword for an `sp1` parameter definition and use the `str(sp1)` keyword for a string parameter instance.

Example

The following sample netlist shows an example of how you can use these definitions for various commands, keywords, parameters, and elements:

```
xibis1 vccq vss out in IBIS
+ IBIS_FILE=str('file1.ibs') IBIS_MODEL=str('model1')
xibis2 vccq vss out in IBIS
+ IBIS_FILE=str('file2.ibs') IBIS_MODEL=str('model2')

.subckt IBIS vccq vss out in
+ IBIS_FILE=str('file.ibs')
+ IBIS_MODEL=str('ibis_model')
ven en 0 vcc
BMCH vccq vss out in en v0dq0 vccq vss buffer=3
+ file= str(IBIS_FILE) model=str(IBIS_MODEL)
+ typ=typ ramp_rwf=2 ramp_fwf=2 power=on
.ends
```

HSPICE supports these kinds of definitions and instances with the following netlist components:

- .PARAM statements
- .SUBCKT statements
- S-parameter FQMODEL in both the S-parameter instance and S-parameter model and the TSTONEFILE keyword in the S-element
- FILE and MODEL keywords
- B-elements
- W-element keywords RLGCFILE, RLGCMODEL, UMODEL, FSMODEL, TABLEMODEL, and SMODEL

String Parameters in Passive and Active Component Keywords

You can include string parameters in all HSPICE passive and active component model name keywords. When you define a parameter that is a character string, use the keyword `str('string')`. When you use an instance of the parameter, the parameter name is `str(parameter_name)`.

Syntax

For passive elements:

```
Rxxx n1 n2 [mname [str(mname)]] Rval [TC1 [TC2][TC]] [SCALE=val]
+ [M=val] [AC=val] [DTEMP=val] [L=val] [W=val] [C=val]
+ [NOISE = val]
Cxxx n1 n2 [mname [str(mname)]] [C = ]capacitance [[TC1 = ]val]
+ [[TC2 = ]val] [SCALE = val] [IC = val] [M = val]
+ [W = val] [L = val] [DTEMP = val]
Lxxx n1 n2 [L = ]inductance [mname [str(mname)]] [[TC1 = ]val]
+ [[TC2 = ]val] [SCALE = val] [IC = val] [M = val]
+ [DTEMP = val] [R = val]
```

For active elements, define the model name by using the original syntax, or string parameter model-name syntax. (See the [HSPICE Reference Manual: MOSMOD Models](#) for full listing of transistor parameters that can be overridden in transistor definitions.)

```
Dxxx nplus nminus str(mname) [[AREA = ]area] [[PJ = ]val]
+ [WP = val] [LP = val] [WM = val] [LM = val] [OFF]
+ [IC = vd] [M = val] [DTEMP = val]
Qxxx nc nb ne [ns] str(mname) [area] [OFF]
+ [IC = vbeval,vceval] [M = val] [DTEMP = val]
Jxxx nd ng ns [nb] str(mname) [[[AREA] = area | [W = val]
+ [L = val]]] [OFF] [IC = vdsval,vgsval] [M = val]
+ [DTEMP = val]
Mxxx nd ng ns [nb] str(mname) [[L = ]length] [[W = ]width]
+ [AD = val] AS = val] [PD = val] [PS = val]
+ [NRD = val] [NRS = val] [RDC = val] [RSC = val] [OFF]
+ [IC = vds,vgs,vbs] [M = val] [DTEMP = val]
+ [GEO = val] [DELVTO = val]
```

Example

```
.param mypmos=str('p')
.param mynmos=str('n')
.lib 'ltst.lib' TT

.subckt circuit vout vin vdd nmod=str('nch') pmod=str('pch')
m1 vout vin vdd vdd str(pmod) w=4u l=5u
m2 vout vin 0 0 str(nmod) w=2u l=5u
.ends circuit

x1 vout vin vdd circuit dtemp=11 nmod=str(mynmos) pmod=str(mypmos)
```

Parameter Defaults and Inheritance

Use the `.OPTION PARHIER` parameter to specify scoping rules.

Syntax:

```
.OPTION PARHIER=[GLOBAL | LOCAL]
```

The default setting is GLOBAL.

Example

This example explicitly shows the difference between local and global scoping for using parameters in subcircuits.

The input netlist includes the following:

```
.OPTION parhier=[global | local]
.PARAM DefPwid=1u
.SUBCKT Inv a y DefPwid=2u DefNwid=1u
Mpl MosPinList pMosMod L=1.2u W=DefPwid
Mn1 MosPinList nMosMod L=1.2u W=DefNwid
.ENDS
```

Set the `.OPTION PARHIER=parameter scoping option` to GLOBAL. The netlist also includes the following input statements:

Chapter 11: Parameters and Functions

Parameter Scoping and Passing

```
xInv0 a y0 Inv          $ override DefPwid default,
$ xInv0.Mp1 width=1u
xInv1 a y1 Inv DefPwid=5u $ override DefPwid=5u,
$ xInv1.Mp1 width=1u

.measure tran Wid0 param='lv2(xInv0.Mp1)' $ lv2 is the
           $ template for
.measure tran Wid1 param='lv2(xInv1.Mp1)' $ the channel
           $ width
           $ 'lv2(xInv1.Mp1)'
.ENDS
```

Simulating this netlist produces the following results in the listing file:

```
wid0=1.0000E-06
wid1=1.0000E-06
```

If you change the `.OPTION PARHIER=parameter scoping option` to `LOCAL`:

```
xInv0 a y0 Inv          $ not override .param
           $ DefPwid=2u,
           $ xInv0.Mp1 width=2u
xInv1 a y1 Inv DefPwid=5u          $ override .param
           $ DefPwid=2u,
           $ xInv1.Mp1 width=5u:
.measure tran Wid0 param='lv2(xInv0.Mp1)'$ override the
.measure tran Wid1 param='lv2(xInv1.Mp1)'$ global .PARAM
...
```

Simulation produces the following results in the listing file:

```
wid0=2.0000E-06
wid1=5.0000E-06
```

Parameter Passing

[Figure 32 on page 393](#) shows a flat representation of a hierarchical circuit, which contains three resistors.

Each of the three resistors obtains its simulation time resistance from the *Val* parameter. The netlist defines the *Val* parameter in four places, with three different values.

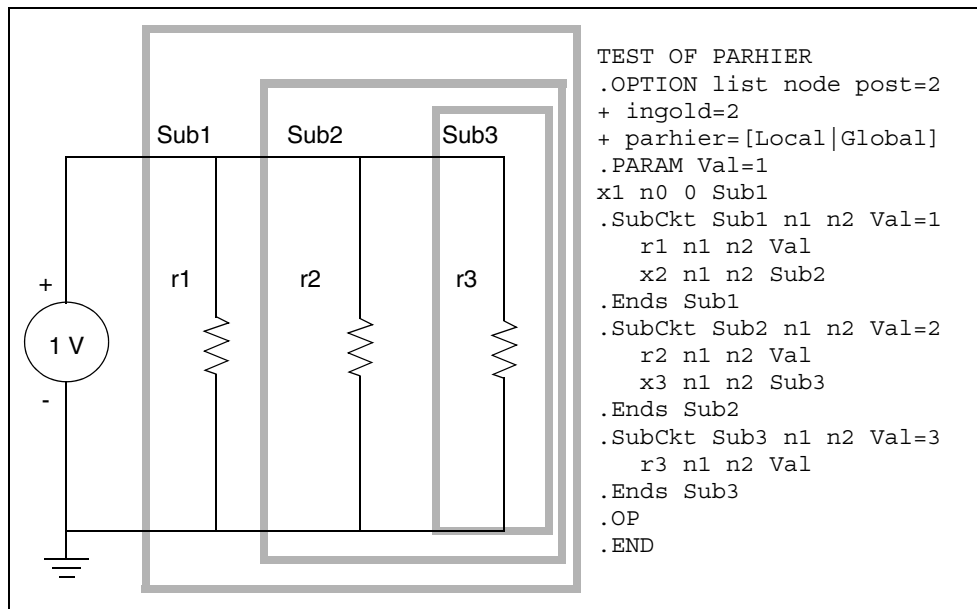


Figure 32 Hierarchical Parameter Passing Problem

The total resistance of the chain has two possible solutions: 0.3333Ω and 0.5455Ω .

You can use `.OPTION PARHIER` to specify which parameter value prevails, when you define parameters with the same name at different levels of the design hierarchy.

Under global scoping rules, if names conflict, the top-level assignment `.PARAM Val=1` overrides the subcircuit defaults, and the total is 0.3333Ω . Under local scoping rules, the lower level assignments prevail, and the total is 0.5455Ω (one, two, and three ohms in parallel).

The example in Figure 32 produces the results in Table 29, based on how you set `.OPTION PARHIER` to local/global:

Table 29 PARHIER=LOCAL vs. PARHIER=GLOBAL Results

Element	PARHIER=Local	PARHIER=Global
r1	1.0	1.0
r2	2.0	1.0

Chapter 11: Parameters and Functions

Parameter Scoping and Passing

Table 29 *PARHIER=LOCAL vs. PARHIER=GLOBAL Results (Continued)*

Element	PARHIER=Local	PARHIER=Global
r3	3.0	1.0

Parameter Passing Solutions

The following checklist determines whether you see simulation differences when you use the default scoping rules. These checks are especially important if your netlists contain devices from multiple vendor libraries.

- Check your subcircuits for parameter defaults, on the `.SUBCKT` or `.MACRO` line.
- Check your subcircuits for a `.PARAM` statement, within a `.SUBCKT` definition.
- To check your circuits for global parameter definitions, use the `.PARAM` statement.
- If any of the names from the first three checks are identical, set up two HSPICE simulation jobs: one with `.OPTION PARHIER=GLOBAL`, and one with `.OPTION PARHIER=LOCAL`. Then look for differences in the output.

Simulation Output

Describes how to use output format statements and variables to display steady state, frequency, and time domain simulation results.

You can also use output variables in behavioral circuit analysis, modeling, and simulation techniques. To display electrical specifications such as rise time, slew rate, amplifier gain, and current density, use the output format features.

HSPICE ships hundreds of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

For descriptions of individual HSPICE commands referenced in this chapter, see the [HSPICE Reference Manual: Commands and Control Options](#).

Note: Parameter Storage Format (PSF) output supports all HSPICE analyses in the HSPICE integration to the Cadence® Virtuoso® Analog Design Environment.

Platform limitation: PC Windows does not support PSF format for HSPICE.

The following sections discuss these topics:

- [Overview of Output Statements](#)
- [Displaying Simulation Results](#)
- [Selecting Simulation Output Parameters](#)
- [Specifying User-Defined Analysis \(.MEASURE\)](#)
- [Expected State of Digital Output Signal \(.DOUT\)](#)
- [Reusing Simulation Output as Input Stimuli \(HSPICE Only\)](#)
- [Element Template Listings \(HSPICE Only\)](#)
- [Vdmargin Output](#)

Chapter 12: Simulation Output

Overview of Output Statements

- [Output Listing \(*.lis\) File with .OPTION LIS_NEW Set](#)
- [For MOSFET Information Use .OPTION LIST](#)
- [HPP Status Updates in *.lis File](#)
- [Verilog-A Simulation Output](#)
- [Field Solver *.str File](#)
- [Redirecting the Simulation Output Results Files to a Different Directory](#)
- [Getting Data Out of HSPICE Plot Files](#)
- [Using the HSPICE Output Converter Utility](#)
- [Troubleshooting Issues](#)

Overview of Output Statements

The following sections discuss these topics:

- [Output Commands](#)
- [Output Variables](#)

Output Commands

The input netlist file contains output statements, including `.PRINT`, `PROBE`, `.MEASURE`, `.DOUT`, and `.STIM`. Each statement specifies the output variables, and the type of simulation result, to display—such as `.DC`, `.AC`, or `.TRAN`. When you specify `.OPTION POST`, HSPICE puts all output variables referenced in `.PRINT`, `.PROBE`, `.MEASURE`, `.DOUT`, and `.STIM` statements into HSPICE output files.

Use the `.STIM` command if you want to reuse the simulation results in another simulation. For example:

```
.stim tran pw1 filename=new v(2)
```

In the example, the `.STIM` command creates a file with an extension `.pw10_tr#` and the results are in the form of a PWL source.

Standalone HSPICE RF supports only `.OPTION POST`, `.OPTION PROBE`, `.PRINT`, `.PROBE`, and `.MEASURE` statements. It does not support `.DOUT` or `.STIM` statements. Refer to the *HSPICE Reference Manual*:

Commands and Control Options for information on all listed statements. See [Table 31 on page 419](#) for output table variable types.

Table 30 Output Statements

Output Statement	Description
.PRINT	Prints numeric analysis results in the output listing file (and post-processor data, if you specify .OPTION POST). See .PRINT .
.PROBE	Outputs data to post-processor output files, but not to the output listing (used with .OPTION PROBE, to limit output). See .PROBE .
.MEASURE	Prints the results of specific user-defined analyses (and post-processor data, if you specify .OPTION POST), to the output listing file. See .MEASURE (or) .MEAS .
.DOUT (HSPICE only)	Specifies the expected final state of an output signal. See .DOUT or Expected State of Digital Output Signal (.DOUT) .
.STIM (HSPICE only)	Specifies simulation results to transform to PWL, Data Card, or Digital Vector File format. See .STIM .

Output Variables

The output format statements require special output variables, to print or plot analysis results for nodal voltages and branch currents. HSPICE uses the following output variables:

- DC and transient analysis
- AC analysis
- element template (HSPICE only)
- .MEASURE statement
- parametric analysis

For HSPICE, DC and transient analysis displays:

Chapter 12: Simulation Output

Displaying Simulation Results

- individual nodal voltages: $V(n1 [,n2])$
- branch currents: $I(Vxx)$
- element power dissipation: $In(element)$

AC analysis displays imaginary and real components of a nodal voltage or branch current, and the magnitude and phase of a nodal voltage or branch current. AC analysis results also print impedance parameters, and input and output noise.

Element template analysis displays element-specific nodal voltages, branch currents, element parameters, and the derivatives of the element's node voltage, current, or charge.

The `.MEASURE` statement variables define the electrical characteristics to measure in a `.MEASURE` statement analysis.

Parametric analysis variables are mathematical expressions, which operate on nodal voltages, branch currents, element template variables (HSPICE only), or other parameters that you specify. Use these variables when you run behavioral analysis of simulation results. See [Using Algebraic Expressions on page 379](#).

Displaying Simulation Results

The following sections describes the statements that you can use to display simulation results for your specific requirements.

- [.PRINT Statement](#)
- [.PROBE Statement](#)
- [Using Wildcards in PRINT and PROBE Statements](#)
- [Print Control Options](#)
- [Printing the Subcircuit Output](#)
- [Using .MODEL_INFO to Print Model Parameters](#)
- [Progress Message Time Estimates](#)

.PRINT Statement

The `.PRINT` statement specifies output variables for which HSPICE prints values.

To simplify parsing of the output listings, HSPICE prints a single `x` in the first column, to indicate the beginning of the `.PRINT` output data. A single `y` in the first column indicates the end of the `.PRINT` output data.

Standalone HSPICE RF prints the `.PRINT` output data to a separate file.

You can include wildcards in `.PRINT` statements.

You can also use the `ia11` keyword in a `.PRINT` statement, to print all branch currents of all diode, BJT, JFET, or MOSFET elements in your circuit design. HSPICE outputs complex data for `i(*)`, `i1(*)`, `i2(*)`, `i3(*)`, `i4(*)` of an AC analysis into `*.ac#` files when specifying the POST, PSF, CSDF or WDF formats for both the `.PRINT` and `.PROBE` commands.

Example

If your circuit contains four MOSFET elements (named `m1`, `m2`, `m3`, `m4`), then `.PRINT ia11 (m*)` is equivalent to `.PRINT i(m1) i(m2) i(m3) i(m4)`. It prints the output currents of all four MOSFET elements. A resultant PSF file has only one signal, `i(c0)`, that contains two real and imaginary points (Re, Im) for each AC frequency point. In other words, if you open the PSF file in a waveform viewer, you see one signal name that contains both the real and Imaginary data, per frequency point.

Statement Order

HSPICE creates different `.sw0` and `.tr0` files, based on the order of the `.PRINT` and `.DC` statements. If you do not specify an analysis type for a `.PRINT` command, the type matches the last analysis command in the netlist, before the `.PRINT` statement.

.PROBE Statement

HSPICE usually saves all voltages, supply currents, and output variables. Set `.OPTION PROBE`, to save output variables only. Use the `.PROBE` statement to specify the quantities to print in the output listing.

If your interest is only in the output data file, and you do not want tabular or plot data in your listing file, set `.OPTION PROBE` and use `.PROBE` to select the values to save in the output listing.

You can include wildcards in `.PROBE` statements.

Using Wildcards in PRINT and PROBE Statements

You can include wildcards in `.PRINT` and `.PROBE` statements. Refer to this example netlist in the discussion that follows:

```
* test wildcard
.option post
v1 1 0 10
r1 1 n20 10
r20 n20 n21 10
r21 n21 0 10
.dc v1 1 10 1
***Wildcard equivalent for:
*.print i(r1) i(r20) i(r21) i(v1)
.print i(*)
***Wildcard equivalent for:
*.probe v(0) v(1)
.probe v(?)
***Wildcard equivalent for:
*.print v(n20) v(n21)
.print v(n2?)
***Wildcard equivalent for:
*.probe v(n20, 1) v(n21, 1)
.probe v(n2*, 1)
.end
```

The following sections discuss these topics.

- [Supported Wildcard Templates](#)
- [Using filter in .PRINT and .PROBE Statements](#)
- [Using level in .PRINT and .PROBE Statements](#)
- [Switching to .PROBE to Output Subcircuit Port Names](#)

Supported Wildcard Templates

```
v vm vr vi vp vdb vt
i im ir ii ip idb it
p pm pr pi pp pdb pt
lxn<n> lvn<n> (n is a number 0~9)
i1 im1 ir1 ii1 ip1 idb1 it1
i2 im2 ir2 ii2 ip2 idb2 it2
i3 im3 ir3 ii3 ip3 idb3 it3
i4 im4 ir4 ii4 ip4 idb4 it4
iall isub
```

For details on wildcard templates, see `.PRINT` statement (see [Selecting Simulation Output Parameters on page 407](#)).

When you use the wildcard `i(*)` in a `.print` or `.probe` statement, HSPICE outputs all branch currents.

For `.AC` analysis, to plot all currents for each valid AC output-variable type, you can also use the following in statements:

```
im(*) ir(*) ip(*) idb(*) it(*)
```

In the preceding test case (named `test wildcard`), if you use an `.AC` statement instead of a `.DC` statement, you can use any valid AC output-variable types with the wildcards `v(n2?)` and `v(n2*,1)`. For example:

```
vm(n2?) vr(n2?) vi(n2?) vp(n2?) vdb(n2?) vt(n2?)
vm(n2*,1) vr(n2*,1) vi(n2*,1) vp(n2*,1) vdb(n2*,1) vt(n2*,1)
```

To output the branch current at all terminals of a diode, BJT, JFET or MOSFET, use the output template `iall`. For example, `iall(m*)` is equivalent to:

```
i1(m*) i2(m*) i3(m*) i4(m*)
```

Using filter in .PRINT and .PROBE Statements

You can include the `filter` clause in `.PRINT` and `.PROBE` statements:

```
filter=pattern
```

HSPICE does not print nodes/elements that match the pattern specified in the `filter` clause when it prints node voltage(s) and/or element current(s) that you specify by wildcard patterns such as: `.print v(x1.x2.*)`. Each `filter` applies to all wildcard voltages/currents that the current `.print` or `.probe` statement prints.

For example:

```
.print v(x1.x2.*) i(x1.x2.*) filter='x1.x2.n*' filter='x1.x2.a*'
```

This syntax example prints the voltages of all nodes in subckt `x1 .x2` that do not start with `n` or `a`, and the current of all elements in subckt `x1 .x2` that do not start with either `n` or `a`.

Using level in .PRINT and .PROBE Statements

You can include the `level` clause in `.PRINT` and `.PROBE` statements:

```
level=val2
```

This setting is effective only when you specify the wildcard character in the output variable. The level value `val2` specifies the number of hierarchical depth levels when the wildcard node/element name matches.

- When you set `val2` to 1, the wildcard match applies to the same depth level where the `.print` statement is.
- When the `val2` is 2, it applies to the same level and to one level below the current level where `.print` is.
- When `val2` is -1, the wildcard match applies to all the depth levels below and including the current level of `.print` statement.
- The default value of `val2` is -1.

Switching to .PROBE to Output Subcircuit Port Names

In cases where you have a subcircuit whose port names are different than instance node names, you can replace the `.PRINT` command with `.PROBE` to print voltage values of a subcircuit.

For example, assume you have a netlist as follows:

```
x3 1 2 inv
  .subckt inv in out
    mn out in 0 0 nch w=1u l=90n
    mp out in 1 1 pch w=1u l=90n
  .ends
.print tran v(*)
```

By default HSPICE prints only the instance node names and not the subcircuit ports. If you include `.print v(*)` in the netlist, HSPICE outputs the top level instance nodes 1 and 2. However, if you want to print the subcircuit ports `in` and `out` then you need to use `.OPTION PROBE` along with an explicit `.PROBE` command. In this case: `.PROBE tran v(x1.in) v(x1.out)`

The following netlist outputs the subcircuit port waveforms:

```
.OPTION POST PROBE
x3 1 2 inv
  .subckt inv in out
    mn out in 0 0 nch w=1u l=90n
    mp out in 1 1 pch w=1u l=90n
  .ends
.PROBE tran v(x1.in) v(x1.out)
```

Print Control Options

The codes that you can use to specify the element templates for output in HSPICE are:

- `.OPTION INGOLD` for output in exponential form.
- `.OPTION POST` where an interactive waveform viewer displays plots.

HSPICE supports the following plot file formats: `*.tr#`, `*.ac#`, and `*.sw#`. If a plot fails to open, it is due to one of the following reasons:

- Does not support the waveform file format.
- Does not read the file format.
- Does not find the file.
- File exceeds the max size of (x) .

Changing the File Descriptor Limit (HSPICE Only)

A simulation that uses a large number of `.ALTER` statements might fail because of the limit on the number of file descriptors. For example, for a Sun workstation, the default number of file descriptors is 64, so a design with more than 50 `.ALTER` statements probably fails, with the following error message:

```
error could not open output spool file /tmp/tmp.nnn
a critical system resource is inaccessible or exhausted
```

To prevent this error on a Sun workstation, enter the following operating system command, before you start the simulation:

```
limit descriptors 128
```

For platforms other than Sun workstations, ask your system administrator to help you increase the number of files that you can open concurrently.

Printing the Subcircuit Output

The following examples demonstrate how to print or plot voltages of nodes that are in subcircuit definitions, using `.PRINT`.

Note: In the following example, you can substitute `.PROBE`, instead of `.PRINT`.

Example 1

```
.GLOBAL vdd vss
X1 1 2 3 nor2
X2 3 4 5 nor2
.SUBCKT nor2 A B Y
.PRINT v(B) v(N1) $ Print statement 1
M1 N1 A vdd vdd pch w=6u l=0.8u
M2 Y B N1 vdd pch w=6u l=0.8u
M3 Y A vss vss vss nch w=3u l=0.8u
M4 Y B vss vss nch w=3u l=0.8u
.ENDS
```

Print statement 1 prints out the voltage on the B input node, and on the N1 internal node for every instance of the nor2 subcircuit.

```
.PRINT v(1) v(X1.A) $ Print statement 2
```

The preceding `.PRINT` statement specifies two ways to print the voltage on the A input of the X1 instance.

```
.PRINT v(3) v(X1.Y) v(X2.A) $ Print statement 3
```

The preceding `.PRINT` statement specifies three different ways to print the voltage at the Y output of the X1 instance (or the A input of the X2 instance).

```
.PRINT v(X2.N1) $ Print statement 4
```

The preceding `.PRINT` statement prints the voltage on the N1 internal node of the X2 instance.

```
.PRINT i(X1.M1) $ Print statement 5
```

The preceding `.PRINT` statement prints out the drain-to-source current, through the M1 MOSFET in the X1 instance.

Example 2

```
X1 5 6 YYY
.SUBCKT YYY 15 16
  X2 16 36 ZZZ
  R1 15 25 1
  R2 25 16 1
.ENDS
.SUBCKT ZZZ 16 36
  C1 16 0 10P
  R3 36 56 10K
  C2 56 0 1P
.ENDS
.PRINT V(X1.25) V(X1.X2.56) V(6)
```

Value	Description
V(X1.25)	Local node to the YYY subcircuit definition, which the X1 subcircuit calls.
V(X1.X2.56)	Local node to the ZZZ subcircuit. The X2 subcircuit calls this node; X1 calls X2.
V(6)	Voltage of node 16, in the X1 instance of the YYY subcircuit.

This example prints voltage analysis results at node 56, within the X2 and X1 subcircuits. The full path, X1.X2.56, specifies that node 56 is within the X2 subcircuit, which in turn is within the X1 subcircuit.

Using .MODEL_INFO to Print Model Parameters

Use the command `.MODEL_INFO ALL | Instance1,...Instance2, ...` to generate a text output file with the suffix `*.model_info#`. (ALL overrides Instance1,...Instance2.) See [.MODEL_INFO](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Different MOSFET instances may use the same model card, to avoid printing of duplicate model information and reduce the file size, so the unique model card prints only once.

The output file contains two parts, instance information and model information. The instance information section contains the instance name and its corresponding model name used. The model information section contains all

Chapter 12: Simulation Output

Displaying Simulation Results

model parameters of each unique model card. Each instance and .model statement is in one line. For example:

```
Output file: *.model_info#

<sweep param =...>

*** mosfets ***

Element Name          Model Name
Instance_name1       model_name
Instance_name2       model_name
...

*** models ***

.model      Model_name1      model_par1 =val1 model_par2=val2 ...
.model      Model_name2      model_par1 =val1 model_par2=val2 ...
```

Sample *.model_info# Output File

```
Input file: test.sp...

X1 c d test1
X2 a b test2

.model_info  x1.m1 x2.m2
.Subckt test1 d g
m1 d g d1 pw nch w=0.2u l=0.03u
m2 d1 g s nw pch w=0.5u l=0.03u
.model pch.1 pmos level=54 version=4.6
+ binunit=1 paramchk=1 mobmod=0
+ ...
.model pch.2 pmos level=54 version=4.6
+ binunit=1 paramchk=1 mobmod=0
+ ...
.model pch.26 pmos level=54 version=4.6
```

Progress Message Time Estimates

During the simulation, simulation progress messages are displayed to the screen. In the simulation progress messages, `etc` is the estimated time to completion and `ett` is the estimated total time. See the sample below.

```
99.6% time = 199.200374 ns ( etc = 1:05, ett = 4:26:22 )
( wall = 4:25:16 cpu = 1:11:06:58 s=7.9424 )
99.7% time = 199.404835 ns ( etc = 49.4 sec, ett = 4:26:20 )
( wall = 4:25:30 cpu = 1:11:08:49 s=7.94239 )
99.8% time = 199.600000 ns ( etc = 32.9 sec, ett = 4:26:20 )
( wall = 4:25:47 cpu = 1:11:11:04 s=7.94241 )
99.9% time = 199.800806 ns ( etc = 16.5 sec, ett = 4:26:22 )
( wall = 4:26:05 cpu = 1:11:13:28 s=7.9424 )
100.0% time = 200.000000 ns ( etc = 0.0 sec, ett = 4:26:19 )
( wall = 4:26:19 cpu = 1:11:15:12 s=7.94241 )
100.0% time = 200.000000 ns ( etc = 0.0 sec, ett = 4:26:19 )
( wall = 4:26:19 cpu = 1:11:15:12 s=7.94241 )
```

Selecting Simulation Output Parameters

Parameters provide the appropriate simulation output. To define simulation parameters, use the `.OPTION` and `.MEASURE` statements, and define specific variable elements.

The following sections discuss these topics:

- [DC and Transient Output Variables](#)
- [Operating Point Variables](#)
- [AC Analysis Output Variables](#)
- [Element Template Output \(HSPICE Only\)](#)

DC and Transient Output Variables

- Voltage differences between specified nodes (or between one specified node and ground).
- Current output for an independent voltage source.
- Current output for any element.

Chapter 12: Simulation Output

Selecting Simulation Output Parameters

- Current output for a subcircuit pin.
- Element templates (HSPICE only). For each device type, the templates contain:
 - values of variables that you set
 - state variables
 - element charges
 - capacitance currents
 - capacitances
 - derivatives

[Print Control Options on page 403](#) summarizes the codes that you can use, to specify the element templates for output in HSPICE.

The following sections discuss these topics.

- [Nodal Capacitance Output](#)
- [Nodal Voltage](#)
- [Current: Independent Voltage Sources](#)
- [Terminal Voltage: MOS Instance](#)
- [Current: Element Branches](#)
- [Current: Subcircuit Pin](#)
- [Independent Source Power Output](#)
- [Wildcard Support](#)
- [Print Power](#)
- [Diode Power Dissipation](#)
- [BJT Power Dissipation](#)
- [JFET Power Dissipation](#)
- [MOSFET Power Dissipation](#)

Nodal Capacitance Output

Syntax

Cap (*nxxx*)

For nodal capacitance output, HSPICE prints or plots the capacitance of the specified node *nxxxx*.

Example

```
.print dc Cap(5) Cap(6)
```

Nodal Voltage

Syntax

```
V(n1 [, n2 ] )
```

Parameter	Description
<i>n1, n2</i>	HSPICE prints or plots the voltage difference (<i>n1-n2</i>) between the specified nodes. If you omit <i>n2</i> , HSPICE prints or plots the voltage difference between <i>n1</i> and ground (node 0).

Current: Independent Voltage Sources

Syntax

```
I(Vxxx)
```

Parameter	Description
<i>Vxxx</i>	Voltage-source element name. If an independent power supply is within a subcircuit, then to access its current output, append a dot and the subcircuit name to the element name. For example, I(X1.Vxxx).

Example

```
.PRINT TRAN I(VIN)
.PRINT DC I(X1.VSRC)
.PRINT DC I(XSUB.XSUBSUB.VY)
```

Terminal Voltage: MOS Instance

Syntax

$Vn(MOSFET_name)$

Parameter	Description
n	Node position number in the element statement. n is 1, 2, 3 or 4. For example: $v1$ is the first node (drain) voltage.
$MOSFET_name$	MOSFET instance name. To access current output for an element in a subcircuit, append a dot and the subcircuit name to the element name. For example, $v3(X1.Wwww)$.

Example

The following example outputs the gate node voltage for the MOSFET.

```
.probe tran v2(XINST1.MN0)
```

Current: Element Branches

Note: With the F-2011.09 release, the direction of the current can follow the HSPICE-specific mode or the generic current polarity mode, based on use of the `.OPTION GEN_CUR_POL=ON|OFF`. The default is `OFF`. See [.OPTION GEN_CUR_POL](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Syntax

$In(Wwww)$
 $Iall(Wwww)$

Parameter	Description
n	Node position number, in the element statement. For example, if the element contains four nodes, $I3$ is the branch-current output for the third node. If you do not specify n , HSPICE assumes the first node.
$Wwww$	Element name. To access current output for an element in a subcircuit, append a dot and the subcircuit name to the element name. For example, $I3(X1.Wwww)$.

Parameter	Description
Iall (Wwww)	<p>An alias just for diode, BJT, JFET, and MOSFET devices.</p> <ul style="list-style-type: none"> ▪ If <i>Wwww</i> is a diode, it is equivalent to: I1(Wwww) I2(Wwww). ▪ If <i>Wwww</i> is one of the other device types, it is equivalent to: I1(Wwww) I2(Wwww) I3(Wwww) I4(Wwww)

Example 1

I1 (R1)

This example specifies the current through the first R1 resistor node.

Example 2

I4 (X1 .M1)

This example specifies the current, through the fourth node (the substrate node) of the M1 MOSFET, defined in the X1 subcircuit.

Example 3

I2 (Q1)

The last example specifies the current, through the second node (the base node) of the Q1 bipolar transistor.

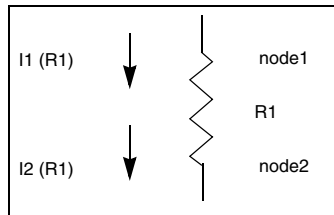
To define each branch circuit, use a single element statement. When HSPICE evaluates branch currents, by default, it inserts a zero-volt power supply, in series with branch elements to enable the HSPICE current direction rule to be more device-aware. The `.OPTION GEN_CUR_POL=ON|OFF` is available to allow use of Custom Designer + HSPICE. When `.OPTION GEN_CUR_POL=ON`, the `i2()` ... `iN()` direction uses a generic direction rule, that is: the current *in* is positive, and the current *out* is negative (see [Figure 33 on page 412](#) through [Figure 38 on page 413](#)).

If HSPICE cannot interpret a `.PRINT` statement that contains a branch current, it generates a warning.

Branch current direction for the elements in `s` defined in terms of arrow-notation (current direction) and node-position number (terminal type). (See [Figure 33](#) through [Figure 38](#).)

Chapter 12: Simulation Output
 Selecting Simulation Output Parameters

.OPTION GEN_CUR_POL=OFF (default)



.OPTION GEN_CUR_POL=ON

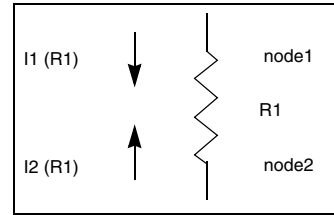
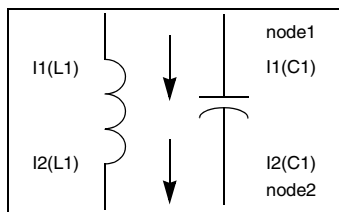


Figure 33 Resistor (node1, node2)

.OPTION GEN_CUR_POL=OFF (default)



.OPTION GEN_CUR_POL=ON

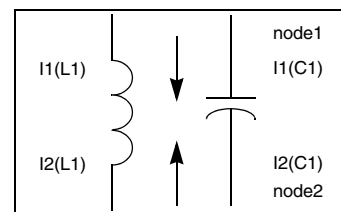
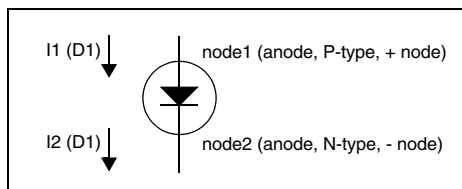


Figure 34 Inductor (node1, node2); capacitor (node 1, node2)

.OPTION GEN_CUR_POL=OFF (default)



.OPTION GEN_CUR_POL=ON

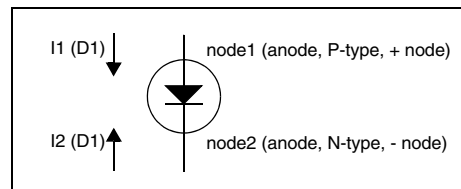
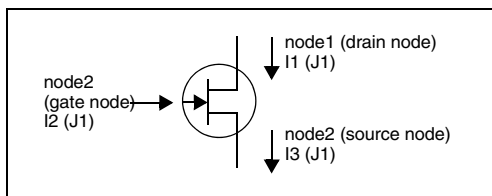


Figure 35 Diode (node1, node2)

.OPTION GEN_CUR_POL=OFF (default)



.OPTION GEN_CUR_POL=ON

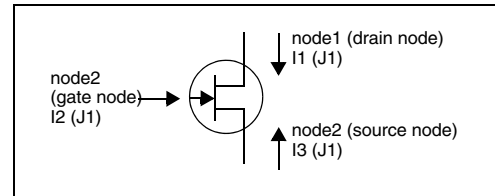


Figure 36 JFET (node1, node2, node3) - n-channel

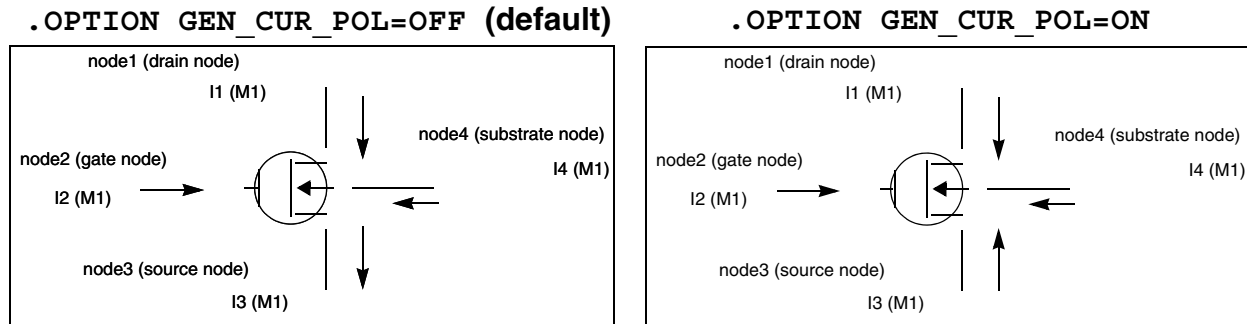


Figure 37 MOSFET (node1, node2, node3, node4) - n-channel

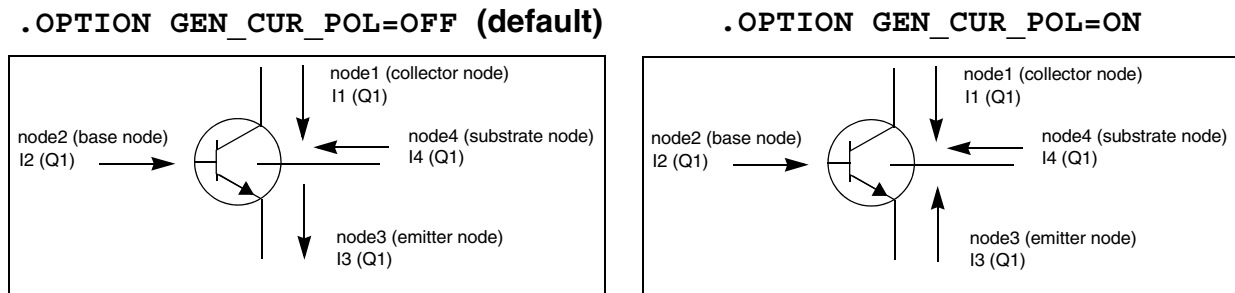


Figure 38 BJT (node1, node2, node3, node4) - npn

Current: Subcircuit Pin

Syntax

```
ISUB (X**** . ****)
```

Example

```
.PROBE ISUB (X1.PIN1)
```

Independent Source Power Output

Syntax

```
SRC_PWR
```

Example

```
.print [dc|tran] src_pwr
```

For power calculations, HSPICE computes dissipated or stored power in each passive element (R, L, C), and source (V, I, G, E, F, and H). To compute this

power, HSPICE multiplies the voltage across an element, and its corresponding branch current.

However, for semiconductor devices, HSPICE calculates only the dissipated power. It excludes the power stored in the device junction or parasitic capacitances from the device power computation. The following sections show equations for calculating the power that different types of devices dissipate.

HSPICE also computes the total power of a circuit, which is the dissipated power + stored power. Total power is the negative value of sum of independent sources.

Note: Since HSPICE and HSPICE RF compute only the dissipated power for semiconductor devices, total power is *not* the sum of power of elements that exclude independent sources.

Wildcard Support

Wildcard support is available for current subcircuit pins in single and multiple hierarchies using asterisk (*) and question mark (?) characters. (Exception: (?) is disallowed.) For example:

Single Hierarchy

```
.print isub(x1.*) isub(x1.a?)
```

Multi-level Hierarchy

```
.print isub(x1.x2.*) isub(x1.x?.a?)
```

Print Power

```
.PRINT [DC | TRAN] P(element_or_subcircuit_name) POWER
```

HSPICE calculates power only for transient and DC sweep analyses. Use the `.MEASURE` statement to compute the average, RMS, minimum, maximum, and peak-to-peak value of the power. The `POWER` keyword invokes the total power dissipation output.

HSPICE RF supports `p(instance)` but not the `POWER` variable in DC/transient analysis.

Example

```
.PRINT TRAN P(M1) P(VIN) P(CLOAD) POWER
.PRINT TRAN P(Q1) P(DIO) P(J10) POWER
.PRINT TRAN POWER $ Total transient analysis
* power dissipation
.PRINT DC POWER P(IIN) P(RLOAD) P(R1)
.PRINT DC POWER P(V1) P(RLOAD) P(VS)
.PRINT TRAN P(Xf1) P(Xf1.Xh1)
```

Diode Power Dissipation

$$P_d = V_{pp}' \cdot (I_{do} + I_{cap}) + V_{p'n} \cdot I_{dc}$$

Parameter	Description
Pd	Power dissipated in the diode.
I _{do}	DC component of the diode current.
I _{cap}	Capacitive component of the diode current.
V _{p'n}	Voltage across the junction.
V _{pp'}	Voltage across the series resistance, R _S .

BJT Power Dissipation

- Vertical

$$P_d = V_{c'e'} \cdot I_{co} + V_{b'e'} \cdot I_{bo} + V_{cc'} \cdot I_{ctot} + V_{ee'} \cdot I_{etot} + V_{sc'} \cdot I_{so} - V_{cc'} \cdot I_{st}$$

- Lateral

$$P_d = V_{c'e'} \cdot I_{co} + V_{b'e'} \cdot I_{bo} + V_{cc'} \cdot I_{ctot} + V_{bb'} \cdot I_{btot} + V_{ee'} \cdot I_{etot} + V_{sb'} \cdot I_{so} - V_{bb'} \cdot I_{stot}$$

Parameter	Description
I _{bo}	DC component of the base current.
I _{co}	DC component of the collector current.
I _{so}	DC component of the substrate current.

Chapter 12: Simulation Output
 Selecting Simulation Output Parameters

Parameter	Description
Pd	Power dissipated in a BJT.
Ibtot	Total base current (excluding the substrate current).
Ictot	Total collector current (excluding the substrate current).
Ietot	Total emitter current.
Istot	Total substrate current.
Vb'e'	Voltage across the base-emitter junction.
Vbb'	Voltage across the series base resistance, RB.
Vc'e'	Voltage across the collector-emitter terminals.
Vcc'	Voltage across the series collector resistance, RC.
Vee'	Voltage across the series emitter resistance, RE.
Vsb'	Voltage across the substrate-base junction.
Vsc'	Voltage across the substrate-collector junction.

JFET Power Dissipation

$$Pd = Vd's' \cdot Ido + Vgd' \cdot Igdo + Vgs' \cdot Igso + Vs's' \cdot (Ido + Igso + Icgs) + Vdd' \cdot (Ido - Igdo - Icgd)$$

Parameter	Description
Icgd	Capacitive component of the gate-drain junction current.
Icgs	Capacitive component of the gate-source junction current.
Ido	DC component of the drain current.
Igdo	DC component of the gate-drain junction current.
Igso	DC component of the gate-source junction current.
Pd	Power dissipated in a JFET.

Parameter	Description
Vd's'	Voltage across the internal drain-source terminals.
Vdd'	Voltage across the series drain resistance, RD.
Vgd'	Voltage across the gate-drain junction.
Vgs'	Voltage across the gate-source junction.
Vs's	Voltage across the series source resistance, RS.

MOSFET Power Dissipation

$$Pd = Vd's' \cdot Ido + Vbd' \cdot Ibdo + Vbs' \cdot Ibso + Vs's' \cdot (Ido + Ibso + Icbs + Icgs) + Vdd' \cdot (Ido - Ibdo - Icbd - Icgd)$$

Parameter	Description
Ibdo	DC component of the bulk-drain junction current.
Ibso	DC component of the bulk-source junction current.
Icbd	Capacitive component of the bulk-drain junction current.
Icbs	Capacitive component of the bulk-source junction current.
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.

Operating Point Variables

When you set `PRINT/.PROBE OP`, only the OP information specified in the statement is output. These commands have a higher priority than the `.OP` command.

Nodal Voltage Syntax

```
V(n1 [, n2])
```

Example 1: This example only outputs the operating point information of variables `I(VIN)` and `I(X1.VSRC)`.

```
.PRINT OP v(1) v(2)
```

Current: Independent Voltage Sources Syntax

```
I(Vxxx)
```

Example 2: This example only outputs the operating point information of the variables `I(VIN)` and `I(X1.VSRC)`.

```
.PRINT OP I(VIN) I(X1.VSRC)
```

For detailed information on controlling the files to which operating point information is written see: [.OP](#), [.OPTION OPFILE](#) and [.OPTION SPLIT_DP](#).

AC Analysis Output Variables

Output variables for AC analysis include:

- Voltage differences between specified nodes (or between one specified node and ground).
- Current output for an independent voltage source.
- Current output for a subcircuit pin.
- Element branch current.
- Impedance (Z), admittance (Y), hybrid (H), and scattering (S) parameters.
- Input and output impedance, and admittance.

Table 31 lists AC output-variable types. In this table, the type symbol appends 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 31 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.

The following sections topics discuss these topics:

- [Nodal Capacitance Output](#)
- [Nodal Voltage](#)
- [Current: Independent Voltage Sources](#)
- [Current: Element Branches](#)
- [Current: Subcircuit Pin](#)
- [Group Time Delay](#)
- [Network](#)
- [Noise and Distortion](#)

Nodal Capacitance Output

Syntax

Cap (nxxx)

For nodal capacitance output, HSPICE prints the capacitance of the specified node nxxxx.

Example

```
.print ac Cap(5) Cap(6)
```

Nodal Voltage

Syntax

```
Vz (n1<, n2>)
```

Parameter	Description
<i>z</i>	Specifies the voltage output type (see Table 31 on page 419)
<i>n1, n2</i>	Specifies node names. If you omit <i>n2</i> , HSPICE assumes ground (node 0).

Example

This example applies to HSPICE, but not HSPICE RF. It prints the magnitude of the AC voltage of node 5, using the VM output variable. HSPICE uses the VDB output variable to print the voltage at node 5, and uses the VP output variable to print the phase of the nodal voltage at node 5.

```
.PRINT AC VM(5) VDB(5) VP(5)
```

Current: Independent Voltage Sources

Syntax

```
Iz (Vxxx)
```

Parameter	Description
<i>z</i>	Current output type (see Table 31 on page 419).
<i>Vxxx</i>	Voltage-source element name. If an independent power supply is within a subcircuit, then to access its current output, append a dot and the subcircuit name to the element name. For example, IM(X1.Vxxx).

Example

```
.PRINT AC IR(V1) IM(VN2B) IP(X1.X2.VSRC)
```

Current: Element Branches

Syntax

`Izn (Wwww)`

Parameter	Description
<code>z</code>	Current output type (see Table 31 on page 419).
<code>n</code>	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.
<code>Wwww</code>	Element name. If the element is within a subcircuit, then to access its current output, append a dot and the subcircuit name to the element name. For example, IM3(X1.Wwww).

```
.PRINT AC IP1(Q5) IM1(Q5) IDB4(X1.M1)
```

If you use the form `In(Xxxx)` for AC analysis output, then HSPICE prints the magnitude value, `IMn(Xxxx)`.

Current: Subcircuit Pin

Syntax

`ISUB (X**** . ****)`

Example

```
.PROBE ISUB(X1.PIN1)
```

Group Time Delay

AC analysis associates the TD group time delay. TD is the negative derivative of the phase in radians, with respect to radian frequency. HSPICE uses the difference method to compute TD:

$$TD = -\frac{1}{360} \cdot \frac{(phase2 - phase1)}{(f2 - f1)}$$

phase1 and *phase2* are the phases (in degrees) of the specified signal, at the *f1* and *f2* frequencies (in hertz).

Syntax

```
.PRINT AC VT(10) VT(2,25) IT(RL)
.PRINT AC IT1(Q1) IT3(M15) IT(D1)
```

Chapter 12: Simulation Output

Selecting Simulation Output Parameters

Note: Because the phase has a discontinuity every 360°, TD shows the same discontinuity, even though TD is continuous.

Example

```
INTEG.SP ACTIVE INTEGRATOR
***** INPUT LISTING
*****
V1  1  0  .5  AC  1
R1  1  2      2K
C1  2  3      5NF
E3  3  0      2 0 -1000.0
.AC DEC      15  1K  100K
.PRINT AC      VT(3)  (0,4U)  VP(3)
.END
```

Network

Syntax

$X_{ij}(z)$, $ZIN(z)$, $ZOUT(z)$, $YIN(z)$, $YOUT(z)$

Parameter	Description
X	Specifies Z (impedance), Y (admittance), H (hybrid), or S (scattering).
ij	i and j can be 1 or 2. They identify the matrix parameter to print.
z	Output type (see Table 31 on page 419). If you omit z, HSPICE prints the magnitude of the output variable.
ZIN	Input impedance. For a one-port network, ZIN, Z11, and H11 are the same.
ZOUT	Output impedance.
YIN	Input admittance. For a one-port network, YIN and Y11 are the same.
YOUT	Output admittance.

Example

```
.PRINT AC  Z11(R)  Z12(R)  Y21(I)  Y22  S11  S11(DB)
.PRINT AC  ZIN(R)  ZIN(I)  YOUT(M)  YOUT(P)  H11(M)
.PRINT AC  S22(M)  S22(P)  S21(R)  H21(P)  H12(R)
```

Noise and Distortion

This section describes the variables used for noise and distortion analysis.

Syntax

```
ovar <(z)>
```

Parameter	Description
ovar	Noise and distortion analysis parameter. It can be ONOISE (output noise), INOISE (equivalent input noise), or any of the distortion analysis parameters (HD2, HD3, SIM2, DIM2, DIM3).
z	Output type (only for distortion). If you omit z, HSPICE outputs the magnitude of the output variable.

Example

```
.PRINT DISTO HD2 (M) HD2 (DB)
```

Prints the magnitude and decibel values of the second harmonic distortion component, through the load resistor that you specified in the `.DISTO` statement (not shown). You cannot use the `.DISTO` statement in HSPICE RF.

```
.PRINT NOISE INOISE ONOISE
```

Note: You can specify the noise and distortion output variable, and other AC output variables, in the `.PRINT AC` statements.

Element Template Output (HSPICE Only)

The `.PRINT`, and `.PROBE` statements use element templates to output user-input parameters, state variables, stored charges, capacitor currents, capacitances, and derivatives of variables. See [Element Template Listings \(HSPICE Only\) on page 438](#).

Syntax

Elname:Property

Parameter	Description
Elname	Name of the element.

Chapter 12: Simulation Output

Specifying User-Defined Analysis (.MEASURE)

Parameter	Description
Property	Property name of an element, such as a user-input parameter, state variable, stored charge, capacitance current, capacitance, or derivative of a variable.

The alias is:

LVnn (Elname)
LXnn (Elname)

Parameter	Description
LV	Form to obtain output of user-input parameters, and state variables.
LX	Form to obtain output of stored charges, capacitor currents, capacitances, and derivatives of variables.
nn	Code number for the desired parameter (See Element Template Listings (HSPICE Only) on page 438 and MOSFET Output Templates, Table 4, Parameters in MOSFET Output Templates).
Elname	Name of the element.

Example

```
.PRINT TRAN V(1,12) I(X2.VSIN) I2(Q3) DI01:GD  
.PRINT TRAN X2.M1:CGGBO M1:CGDBO X2.M1:CGSBO
```

Specifying User-Defined Analysis (.MEASURE)

Use the `.MEASURE` statement to modify information, and to define the results of successive HSPICE simulations.

Computing the measurement results uses postprocessing output. If you use the `INTERP` option to reduce the size of the postprocessing output, then the measurement results can contain interpolation errors. For more information, see [.OPTION INTERP](#) in the *HSPICE Reference Manual: Commands and Control Options*.

This section describes the fundamental measurement modes and includes the following topics:

- [.MEASURE Statement Order](#)
- [.MEASURE Parameter Types](#)
- [FIND and WHEN Functions](#)
- [Continuous Measurement](#)
- [Equation Evaluation](#)
- [Average, RMS, MIN, MAX, INTEG, PP, and EM_AVG](#)
- [INTEGRAL Function](#)
- [DERIVATIVE Function](#)
- [ERROR Function](#)
- [Generating a Measure File \(*.mt0, *.ac0, etc.\) with All Values in a Single Row](#)
- [Outputting Pass/Fail Measure Data](#)
- [Measurements in MOSRA Analysis](#)

.MEASURE Statement Order

The `.MEASURE` statement matches the last analysis command in the netlist before the `.MEASURE` statement.

Example

```
.tran 20p 1.0n sweep sigma -3 3 0.5  
.tran 20p 1.0n sweep monte=20  
.meas mover max v(2,1)
```

In this example, `.meas` matches the second `.tran` statement and generates only one measure output file.

Users need to be aware that there are certain differences on how HSPICE handles `.measures` are handle in the case of complex measurements. Complex `.MEASURE` statements are measure statements dependent on other measure statements.

“Undefined variables” is a common reported issue in `.measure` cases for HSPICE that lead to failed measurements. You may not find the same case in

the FASTSPICE tools due to the differences in how other tools process `.measures`.

Note: HSPICE has a dependency requirement which may not exist in other formats. Fastspice simulators XA, Nanosim, and HSIM support various languages, including HSPICE and other major EDA simulators. These Fastspice simulators process `.measure` results in very flexible manner. These simulators read and reread a netlist for variable identification. Then, XA, NS, and HSIM perform simulation, write results, and then post-process the results to determine the `.measure` results.

If a `.MEASURE` statement does not execute, then HSPICE writes 0.0e0 in the `.mt#` file as the `.MEASURE` result, and writes FAILED in the output listing file. Use `.OPTION MEASFAIL` to write results to the `.mt#`, `.ms#`, or `.ma#` files. For more information, see [.OPTION MEASFAIL](#) in the *HSPICE Reference Manual: Commands and Control Options*.

To control the output variables, listed in `.MEASURE` statements, use the `.PUTMEAS` option. For more information, see the [.OPTION PUTMEAS](#) option in the *HSPICE Reference Manual: Commands and Control Options*.

Note: If a `.measure` statement uses the result of previous `.meas` statement, then the calculation starts when the previous calculation concludes. HSPICE outputs zero until the previous result concludes.

For information on measuring MOSFET parameters see [Measuring the Value of MOSFET Model Card Parameters](#).

.MEASURE Parameter Types

You cannot use measurement parameter results that the `.PARAM` statements in `.SUBCKT` blocks produce, outside of the subcircuit. That is, you cannot pass any measurement parameters defined in `.SUBCKT` statements, as bottom-up parameters in hierarchical designs.

Measurement parameter names must not conflict with standard parameter names. HSPICE issues an error message if it encounters a measurement parameter with the same name as a standard parameter definition.

To prevent `.MEASURE` statement parameters from overwriting parameter values in other statements, HSPICE keeps track of parameter types. If you use the

same parameter name in both a .MEASURE statement and a .PARAM statement at the same hierarchical level, the simulation terminates and reports an error.

No error occurs if parameter assignments are at different hierarchical levels. .PRINT statements that occur at different levels do not print hierarchical information for parameter name headings.

Example

In HSPICE RF simulation output, you cannot apply .MEASURE to waveforms generated from another .MEASURE statement in a parameter sweep.

The following example illustrates how HSPICE handles .MEASURE statement parameters.

```
...
.MEASURE tran length TRIG v(clk) VAL=1.4
+ TD=11ns RISE=1 TARGv(neq) VAL=1.4 TD=11ns
+ RISE=1
.SUBCKT path out in width=0.9u length=600u
+ rm1 in m1 m2mg w='width' l='length/6'
...
.ENDS
```

In the above listing, the *length* in the resistor statement:

```
rm1 in m1 m2mg w='width' l='length/6'
```

does not inherit its value from *length* in the .MEASURE statement:

```
.MEASURE tran length ...
```

because they are of different types.

The correct value of *l* in *rm1* should be:

```
l=length/6=100u
```

In transient analysis, you should not derive the value from a measured value.

FIND and WHEN Functions

The FIND and WHEN functions of the .MEASURE statement specify to measure:

- Any independent variables (time, frequency, parameter).
- Any dependent variables (voltage or current for example).
- Derivative of a dependent variable, if a specific event occurs.

Chapter 12: Simulation Output

Specifying User-Defined Analysis (.MEASURE)

You can use these measure statements in unity gain frequency or phase measurements. You can also use these statements to measure the time, frequency, or any parameter value:

- When two signals cross each other.
- When a signal crosses a constant value.

The measurement starts after a specified time delay, *TD*. To find a specific event, set *RISE*, *FALL*, or *CROSS* to a value (or parameter), or specify *LAST* for the last event.

LAST is a reserved word; you cannot use it as a parameter name in the above measure statements. For definitions of parameters of the measure statement, see [Displaying Simulation Results on page 398](#).

For a full demonstration file for *FIND* and *WHEN* functions follow the path to *ampgain.sp*, which sets unity gain frequency of a BJT diff pair, in [Circuit Optimization Examples on page 993](#) in this user guide.

Continuous Measurement

The continuous measurement feature allows you to specify the continuous measurement of a result derived from a DC, AC, or transient analysis. This feature only applies to *TRIG-TARG* and *Find-When* functions. For example:

```
.measure tran_cont vout1 find v(out1) when v(a1)=2.5 fall=1
```

The *.measure* statements continuously finds the voltage *out1* when the voltage value of node *a1* reaches to 2.5 starting from the first falling edge.

See [.MEASURE \(Continuous Results\)](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Continuous Measure Output Files

HSPICE outputs the continuous measure output into a separate files of the following types:

```
output_prefix_measure_result.mt#  
output_prefix_measure_result.ms#  
output_prefix_measure_result.ma#
```

The file is in text format and readable directly. For example:

```
.measure tran_cont vout1_cont find v(out1) when v(a1)=2.5 fall=1
```

The additional output file name is `t1_vout1_cont.mt`, if the output file prefix is `t1`.

Note: Continuous measurement output is *not* written to the `*.lis` file.

The following is an example of an output file for the measure statement:

```
.measure tran_cont crossing when v(1) = v(2) t1_crossing.mt
1.      crossing, result=1.000000000000e-09
2.      crossing, result=1.000000000000e-07
3.      crossing, result=2.000000000000e-07
4.      crossing, result=3.000000000000e-07
5.      crossing, result=4.000000000000e-07
6.      crossing, result=5.000000000000e-07
7.      crossing, result=6.000000000000e-07
8.      crossing, result=7.000000000000e-07
9.      crossing, result=8.000000000000e-07
10.     crossing, result=9.000000000000e-07
11.     crossing, result=1.000000000000e-06
```

Equation Evaluation

Use the Equation Evaluation form of the `.MEASURE` statement to evaluate an equation that is a function of the results of previous `.MEASURE` statements. The equation must not be a function of node voltages or branch currents.

The `expression` option is an arithmetic expression that uses results from other prior `.MEASURE` statements. If `equation` or `expression` includes node voltages or branch currents, Unexpected results may incur.

Average, RMS, MIN, MAX, INTEG, PP, and EM_AVG

Average (AVG), RMS, MIN, MAX, and peak-to-peak (PP) measurement modes report statistical functions of the output variable, rather than analysis values.

- AVG calculates the area under an output variable divided by the periods of interest.
- RMS divides the square root of the area under the output variable square by the period of interest.
- MIN reports the minimum value of the output function over the specified interval.

Chapter 12: Simulation Output

Specifying User-Defined Analysis (.MEASURE)

- **MAX** reports the maximum value of the output function over the specified interval.
- **PP** (peak-to-peak) reports the maximum value minus the minimum value over the specified interval.
Note: **AVG**, **RMS**, and **INTEG** have no meaning in a DC data sweep, so if you use them, HSPICE issues a warning message.
- **EM_AVG** Calculates the average electromigration current. For a symmetric bipolar waveform, the current is:
 $I_{avg}(0, T/2) - R * I_{avg}(T/2, T)$, where **R** is the recovery factor you specify by using `.option em_recovery`. This measurement also supports wildcards.

Measuring Recovered Electromigration

The `.MEAS` keyword, `EM_AVG`, enables you to calculate “recovered” average current due to electromigration. Recovered average current is specially meaningful for bipolar currents (such as output of the inverter), as the mathematical average for such a waveform is zero. The keyword uses the From-To measurement function to provide a range to measure. For example:

```
.measure tran em em_avg I(out) from=5n to=50n
```

where `out` is the node which specifies the measurement point.

The example does the following operations:

1. Measure the average of positive part of the waveform (`Ipos_avg`) from 5ns to 50ns.
2. Measure the average of negative part (absolute value) of the waveform (`Ineg_avg`) from 5ns to 50ns.
3. Does the operation “ $\max(Ipos_avg, Ineg_avg) - R * \min(Ipos_avg, Ineg_avg)$ ”,

Where **R** is a user-provided coefficient using `.option em_recovery=value`. The default value of `em_recovery` is 1.

See [.OPTION EM_RECOVERY](#) in the *HSPICE Reference Manual: Commands and Control Options*.

4. The polarity of `em_avg` current is same as the polarity of the $\max(Ipos_avg, Ineg_avg)$. Positive, if `abs(pos)` is more than `abs(neg)` and otherwise.

For this feature HSPICE also supports wildcards (*) during `em_avg` measurement. For example:

```
.meas tran em em_avg I(m*) from=10n to=100n
```

INTEGRAL Function

The `INTEGRAL` function reports the integral of an output variable, over a specified period.

DERIVATIVE Function

The `DERIVATIVE` function provides the derivative of:

- An output variable, at a specified time or frequency.
- Any sweep variable, depending on the type of analysis.
- A specified output variable, when some specific event occurs.

In the following HSPICE RF example, the `SLEW` measurement provides the slope of `V(OUT)` during the first time, when `V(1)` is 90% of `VDD`.

```
.MEAS TRAN SLEW DERIV V(OUT) WHEN V(1)='0.90*VDD'
```

ERROR Function

The relative error function reports the relative difference between two output variables. You can use this format in optimization and curve-fitting of measured data. The relative error format specifies the variable to measure and calculate from the `.PARAM` variable. To calculate the relative error between the two, HSPICE uses the `ERR`, `ERR1`, `ERR2`, or `ERR3` function. With this format, you can specify a group of parameters to vary, to match the calculated value and the measured data. The following discusses the `ERR` equations:

Error Equations

ERR

1. `ERR` sums the squares of $(M-C)/\max(M, \text{MINVAL})$ for each point.
2. It then divides by the number of points.
3. Finally, it calculates the square root of the result.
 - `M` (`meas_var`) is the measured value of the device or circuit response.
 - `C` (`calc_var`) is the calculated value of the device or circuit response.

Chapter 12: Simulation Output
 Specifying User-Defined Analysis (.MEASURE)

- NPTS is the number of data points.

$$ERR = \left[\frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} \left(\frac{M_i - C_i}{\max(MINVAL, M_i)} \right)^2 \right]^{1/2}$$

ERR1

ERR1 computes the relative error at each point. For NPTS points, HSPICE calculates NPTS ERR1 error functions. For device characterization, the ERR1 approach is more efficient than the other error functions (ERR, ERR2, ERR3).

$$ERR1_i = \frac{M_i - C_i}{\max(MINVAL, M_i)}, i=1, NPTS$$

HSPICE does not print out each calculated ERR1 value. When you set the ERR1 option, HSPICE calculates an ERR value, as follows:

$$ERR = \left[\frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} ERR1_i^2 \right]^{1/2}$$

ERR2

This option computes the absolute relative error, at each point. For NPTS points, HSPICE calls NPTS error functions.

$$ERR2_i = \left| \frac{M_i - C_i}{\max(MINVAL, M_i)} \right|, i=1, NPTS$$

The returned value printed for ERR2 is:

$$ERR = \frac{1}{NPTS} \cdot \sum_{i=1}^{NPTS} ERR2_i$$

ERR3

$$ERR3_i = \frac{\pm \log \left| \frac{M_i}{C_i} \right|}{\left| \log [\max(MINVAL, |M_i|)] \right|}, i=1, NPTS$$

The + and - signs correspond to a positive and negative M/C ratio.

Note: If the M measured value is less than MINVAL, HSPICE uses MINVAL instead. If the absolute value of M is less than the IGNOR or YMIN value, or greater than the YMAX value, the error calculation does not consider this point.

Generating a Measure File (*.mt0, *.ac0, etc.) with All Values in a Single Row

The MEASFORM option allows you to change how the measure values are written to the measure file. If you set MEASFORM=1, then HSPICE writes the measure values in a single space-delimited row.

You can also use MEASFORM=3 to output values in a single row and generate the file in CSV file format (*.csv) that can be opened in Excel by double-clicking on the file name.

By default, a transient measure file (*.mt0) looks like:

tdelay	vmax	vmin	trise
tfall	temper	alter#	
1.781e-09	5.0072	-4.504e-03	4.778e-09
3.386e-09	25.0000	1	

By using either .OPTION MEASFORM=1 or .OPTION MEASFORM=3, the measure file appears as:

tdelay	vmax	vmin	trise	tfall	temper	alter#
1.781e-09	5.0072	-4.504e-03	4.778e-09	3.386e-09	25.0000	1

Outputting Pass/Fail Measure Data

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

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

Measurements in MOSRA Analysis

When working with measurements in a MOSFET model reliability analysis (MOSRA), you need to use a workaround for measurements beyond the first `mt0` file. For example, in the following netlist, four measures fail:

```
.param t_step=600u
.tran 0.1u '2*t_step'
vBGR BGR 0 pulse ( 0 3 0ns 10us 10us 50u 100u)
.meas tran vbgr_out find V(BGR) at 't_step/20'
.meas tran trise1 when V(BGR)='0.9*vbgr_out'
rise=last
.mosra reltotaltime=3.15e+8 relstep=7.88e+7
.option post=1 probe
.end
```

The file returns the following:

```
result of mt0
vbgr_out trise1 temper alter#
3.0000 1.109e-03 25.0000 1.0000
results of mt1,2,3,4
vbgr_out trise1 temper alter#
3.0000 failed 25.0000 1.0000
```

The first `.measure` statement is independent of `.MOSRA` analysis, so the first measure file (`*.mt0`) succeeds. The second `.measure` statement depends on `.MOSRA` analysis, therefore next four measure files (`mt1`, `mt2`, `mt3`, `mt4`) fail.

Workaround

The workaround to this issue is:

1. Run the simulation without the `.measure` statement. This generates `*.tr0`, `*.tr1` ... files.
2. Copy the `.measure` statement to another file named (for example) `measure_file`.

```
.param t_step=600u
.meas tran vbgr_out find V(BGR) at 't_step/20'
.meas tran trise1 when V(BGR)='0.9*vbgr_out' rise=last
.end
```

3. Run the post process measure utility as follows to successfully generate all measurement files:


```
hspice -meas measure_file -i *.tr1 -o tr1.lis
hspice -meas measure_file -i *.tr2 -o tr2.lis
hspice -meas measure_file -i *.tr3 -o tr3.lis
hspice -meas measure_file -i *.tr4 -o tr4.lis
```

Expected State of Digital Output Signal (.DOUT)

The digital output (.DOUT) statement specifies the expected final state of an output signal (HSPICE only). During simulation, HSPICE compares the simulated results with the expected output vector. An error results if states are different. The .DOUT statement uses either of two syntaxes. In both syntaxes, the time and state parameters define the expected output of the *nd* node.

- The first syntax specifies a single threshold voltage, VTH. Any voltage level above VTH is high; any level below VTH is low.

```
.DOUT nd VTH (time state time_state)
```

where:

nd is the node name

VTH is the single voltage threshold

time is an absolute time-point (max 60)

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: Do not care

- The second syntax defines a threshold for both a logic high (VHI) and low (VLO).

```
.DOUT nd VLO VHI (time state time_state)
```

where:

nd is the node name

VLO is the voltage of the logic low state

VHI is the voltage of the logic high state

time is an absolute time-point (max 60)

Chapter 12: Simulation Output

Expected State of Digital Output Signal (.DOUT)

`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: Do not care

Note: If you specify both syntaxes (`VTH`, plus `VHI` and `VLO`), then HSPICE processes only `VTH`, and ignores `VHI` and `VLO`.

For both cases, the time, state pair describes the expected output. During simulation, HSPICE compares the simulated results against the expected output vector. If the states are different, HSPICE reports an error.

The legal values for `state` are:

- 0: Expect ZERO
- 1: Expect ONE
- X, x: Do not care.
- U, u: Do not care
- Z, z: Expect HIGH IMPEDANCE (do not care).

Example

The `.PARAM` statement in the following example sets the value of the `VTH` variable to 3. The `.DOUT` statement operates on the `node1` node and uses `VTH` as its threshold voltage.

```
.PARAM VTH = 3.0
.DOUT node1 VTH(0.0n 0 1.0n 1
+ 2.0n X 3.0n U 4.0n Z 5.0n 0)
```

When `node1` is above 3V, HSPICE considers it a logic **1**; otherwise, it is a logic **0**.

- At 0ns, the expected state of `node1` is logic-low
- At 1ns, the expected state is logic-high
- At 2ns, 3ns, and 4ns, the expected state is “do not care”
- At 5ns, the expected state is again logic-low

HSPICE supports multiple nodes in the `.DOUT` statement. This enables you to verify signals at the same time point in a single `.DOUT` statement.

For example: `.DOUT B C D (0n 1 1 0 5n 0 0 0)`

Reusing Simulation Output as Input Stimuli (HSPICE Only)

You can use the `.STIM` statement to reuse the results (output) of one simulation, as input stimuli in a new simulation.

Note: `.STIM` is an abbreviation of `.STIMULI`. You can use either form to specify this statement in HSPICE.

The `.STIM` statement specifies:

- Expected stimulus (PWL source, data card, or VEC file).
- Signals to transform.
- Independent variables.

One `.STIM` statement produces one corresponding output file. To control the precision and data format, you can use the same options as you would in a normal simulation. For example:

```
.option numdgt=6    $ sets precision, range is 0 to 10, numdgt=4  
                   is the default  
.option ingold=0    $ sets format, 0=eng 1=combined 2=exponential
```

These settings affect how data prints out for your entire testcase. There is no way to only affect the `.STIM` command because the simulation data is the source of the output of the `.STIM` command.

For the syntax and description of the `.STIM` statement, see the [.STIM](#) command in the *HSPICE Reference Manual: Commands and Control Options*.

Output Files

The `.STIM` statement generates the following output files:

Output File Type	Extension
PWL Source	*.pwl\$_tr# The <code>.STIM</code> statement writes PWL source results to <i>output_file.pwl\$_tr#</i> . This output file results from a <code>.STIM [tran] pwl</code> statement in the input file.

Chapter 12: Simulation Output
 Element Template Listings (HSPICE Only)

Output File Type	Extension
Data Card	.dat\$_tr#, .dat\$_ac#, or .dat\$_sw#The .STIM statement writes DATA Card results to <i>output_file.dat\$_sw#</i> , <i>output_file.dat\$_ac#</i> , or <i>output_file.dat\$_tr#</i> . This output file is the result of a .stim [tran ac dc] data statement in the input file.
Digital Vector File	.vec\$_tr#The .STIM statement writes Digital Vector File results to <i>output_file.vec\$_tr#</i> . This output file is the result of a .stim [tran] vec statement in the input file.

Symbol	Description
tr ac sw	<ul style="list-style-type: none"> ▪ tr=transient analysis. ▪ ac=AC analysis. ▪ sw=DC sweep analysis.
#	Either a sweep number, or a hard-copy file number. For a single sweep run, the default number is 0.
\$	<p>Serial number of the current .STIM statement, within statements of the same stimulus type (pwl, data, or vec).</p> <p>\$=0 ~ n-1 (n is the number of the .STIM statement of that type). The initial \$ value is 0.</p> <p>For example, if you specify three .STIM pwl statements, HSPICE generates three PWL output files, with the suffix names pwl0_tr#, pwl1_tr#, and pwl2_tr#.</p>

Element Template Listings (HSPICE Only)

A full and extensive listing of MOSFET output templates, is in the *HSPICE Reference Manual: MOSFET Models*, [MOSFET Output Templates](#).

Table 32 Resistor (R-element)

Name	Alias	Description
G	LV1	Conductance at analysis temperature.
R	LV2	Resistance at analysis temperature.

Table 32 Resistor (R-element) (Continued)

Name	Alias	Description
TC1	LV3	First temperature coefficient.
TC2	LV4	Second temperature coefficient.

Table 33 Capacitor (C-element)

Name	Alias	Description
CEFF	LV1	Computed effective capacitance.
IC	LV2	Initial condition.
Q	LX0	Charge, stored in capacitor.
CURR	LX1	Current, flowing through capacitor.
VOLT	LX2	Voltage, across capacitor.

Table 34 Inductor (L-element)

Name	Alias	Description
LEFF	LV1	Computed effective inductance.
IC	LV2	Initial condition.
FLUX	LX0	Flux, in the inductor.
VOLT	LX1	Voltage, across inductor.
CURR	LX2	Current, flowing through inductor.

Table 35 Mutual Inductor (K-element)

Name	Alias	Description
K	LV1	Mutual inductance.

Chapter 12: Simulation Output
 Element Template Listings (HSPICE Only)

Table 36 Voltage-Controlled Current Source (G-element)

Name	Alias	Description
CURR	LX0	Current, through the source, if VCCS.
R	LX0	Resistance value, if VCR.
C	LX0	Capacitance value, if VCCAP.
CV	LX1	Controlling voltage.
CQ	LX1	Capacitance charge, if VCCAP.
DI	LX2	Derivative of the source current, relative to the control voltage.
ICAP	LX2	Capacitance current, if VCCAP.
VCAP	LX3	Voltage, across capacitance, if VCCAP.

Table 37 Voltage-Controlled Voltage Source (E-element)

Name	Alias	Description
VOLT	LX0	Source voltage.
CURR	LX1	Current, through source.
CV	LX2	Controlling voltage.
DV	LX3	Derivative of the source voltage, relative to the control current.

Table 38 Current-Controlled Current Source (F-element)

Name	Alias	Description
CURR	LX0	Current, through source.
CI	LX1	Controlling current.
DI	LX2	Derivative of the source current, relative to the control current.

Table 39 Current-Controlled Voltage Source (H-element)

Name	Alias	Description
VOLT	LX0	Source voltage.
CURR	LX1	Source current.
CI	LX2	Controlling current.
DV	LX3	Derivative of the source voltage, relative to the control current.

Table 40 Independent Voltage Source (V-element)

Name	Alias	Description
VOLT	LV1	DC/transient voltage.
VOLTM	LV2	AC voltage magnitude.
VOLTP	LV3	AC voltage phase.

Table 41 Independent Current Source (I-element)

Name	Alias	Description
CURR	LV1	DC/transient current.
CURRM	LV2	AC current magnitude.
CURRP	LV3	AC current phase.

Table 42 Diode (D-element)

Name	Alias	Description
AREA	LV1	Diode area factor.
AREAX	LV23	Area, after scaling.
IC	LV2	Initial voltage, across diode.

Chapter 12: Simulation Output
 Element Template Listings (HSPICE Only)

Table 42 Diode (D-element) (Continued)

Name	Alias	Description
VD	LX0	Voltage, across diode (VD), excluding RS (series resistance).
IDC	LX1	DC current, through diode (ID), excluding RS. Total diode current is the sum of IDC and ICAP.
GD	LX2	Equivalent conductance (GD).
QD	LX3	Charge of diode capacitor (QD).
ICAP	LX4	Current, through the diode capacitor. Total diode current is the sum of IDC and ICAP.
C	LX5	Total diode capacitance.
PID	LX7	Photo current, in diode.

Table 43 BJT (Q-element)

Name	Alias	Description
AREA	LV1	Area factor.
ICVBE	LV2	Initial condition for base-emitter voltage (VBE).
ICVCE	LV3	Initial condition for collector-emitter voltage (VCE).
MULT	LV4	Number of multiple BJTs.
FT	LV5	FT (Unity-gain bandwidth).
ISUB	LV6	Substrate current (DC only, not accounted for charge-induced current).
GSUB	LV7	Substrate conductance.
LOGIC	LV8	LOG 10 (IC).
LOGIB	LV9	LOG 10 (IB).

Table 43 BJT (Q-element) (Continued)

Name	Alias	Description
BETA	LV10	BETA.
LOGBETA1	LV11	LOG 10 (BETA) current.
ICTOL	LV12	Collector current tolerance.
IBTOL	LV13	Base current tolerance.
RB	LV14	Base resistance.
GRE	LV15	Emitter conductance, 1/RE.
GRC	LV16	Collector conductance, 1/RC.
PIBC	LV18	Photo current, base-collector.
PIBE	LV19	Photo current, base-emitter.
VBE	LX0	VBE.
VBC	LX1	Base-collector voltage (VBC).
CCO	LX2	Collector current (CCO) (DC only, not accounted for charge-induced current).
CBO	LX3	Base current (CBO) (DC only, not accounted for charge-induced current).
GPI	LX4	$g_{\pi}=1ib /1vbe$, constant vbc.
GU	LX5	$g_{\mu}=1ib /1vbc$, constant vbe.
GM	LX6	$g_m=1ic /1vbe+ 1ic /1vbe$, constant vce.
G0	LX7	$g_0=1ic /1vce$, constant vbe.
QBE	LX8	Base-emitter charge (QBE).
CQBE	LX9	Base-emitter charge current (CQBE).
QBC	LX10	Base-collector charge (QBC).

Chapter 12: Simulation Output
 Element Template Listings (HSPICE Only)

Table 43 BJT (Q-element) (Continued)

Name	Alias	Description
CQBC	LX11	Base-collector charge current (CQBC).
QCS	LX12	Current-substrate charge (QCS).
CQCS	LX13	Current-substrate charge current (CQCS).
QBX	LX14	External base-collector charge (QBX).
CQBX	LX15	External base-collector charge current (CQBX).
GXO	LX16	1/Rbeff Internal conductance (GXO).
CEXBC	LX17	Base-collector equivalent current (CEXBC).
CAP_BE	LX19	cbe capacitance ($C\pi$).
CAP_IBC	LX20	cbc internal base-collector capacitance ($C\mu$).
CAP_SCB	LX21	csc substrate-collector capacitance for vertical transistors. csb substrate-base capacitance for lateral transistors.
CAP_XBC	LX22	cbcx external base-collector capacitance.
CMCMO	LX23	$1/(TF*IBE) / 1/vbc$.
VSUB	LX24	Substrate voltage.

Table 44 JFET (J-element)

Name	Alias	Description
AREA	LV1	JFET area factor.
VDS	LV2	Initial condition for drain-source voltage.
VGS	LV3	Initial condition for gate-source voltage.
PIGD	LV16	Photo current, gate-drain in JFET.
PIGS	LV17	Photo current, gate-source in JFET.

Table 44 JFET (J-element) (Continued)

Name	Alias	Description
VGS	LX0	VGS.
VGD	LX1	Gate-drain voltage (VGD).
CGSO	LX2	Gate-to-source (CGSO).
CDO	LX3	Drain current (CDO).
CGDO	LX4	Gate-to-drain current (CGDO).
GMO	LX5	Transconductance (GMO).
GDSO	LX6	Drain-source transconductance (GDSO).
GGSO	LX7	Gate-source transconductance (GGSO).
GGDO	LX8	Gate-drain transconductance (GGDO).
QGS	LX9	Gate-source charge (QGS).
CQGS	LX10	Gate-source charge current (CQGS).
QGD	LX11	Gate-drain charge (QGD).
CQGD	LX12	Gate-drain charge current (CQGD).
CAP_GS	LX13	Gate-source capacitance.
CAP_GD	LX14	Gate-drain capacitance.
QDS	LX16	Drain-source charge (QDS).
CQDS	LX17	Drain-source charge current (CQDS).
GMBS	LX18	Drain-body (backgate) transconductance (GMBS).

Table 45 Saturable Core Element (K-element)

Name	Alias	Description
MU	LX0	Dynamic permeability (μ), Weber/(amp-turn-meter).

Chapter 12: Simulation Output

Vdmargin Output

Table 45 Saturable Core Element (K-element) (Continued)

Name	Alias	Description
H	LX1	Magnetizing force (H), Ampere-turns/meter.
B	LX2	Magnetic flux density (B), Webers/meter ² .

Table 46 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).

Vdmargin Output

Output template `Vdmargin(m*)`, alias: `LX286(m*)`, is provided to probe the `Vdmargin` simulation result of either selected or all MOSFET elements. The template prints in the `*.lis` file and as `.sw0/.tr0` files. For example:

```
.print dc Vdmargin(m1) Vdmargin(m2)
```

For operating point analysis, `Vdmargin` is reported in the OP output; for DC sweep, `Vdmargin` is calculated at every sweep point; for Tran analysis, only the points specified in the `.OP` command are calculated.

Here, HSPICE only calculates `Vdmargin` at the following points: 1n, 2.5n, 10n:

```
.op 1n 2.5n 10n
```

If `Vdmargin` is not found in the sweep range, HSPICE reports `Vdmargin=Vd0`, with the following warning message:

```
**Warning** Vdmargin of m1 is out of range
```

See [.IVDMARGIN](#) and [.OPTION IVDMARGIN](#) in the *HSPICE Reference Manual: Commands and Control Options* for more information.

Output Listing (*.lis) File with .OPTION LIS_NEW Set

The following is a sample of what HSPICE generates during a simulation in the output listing file *.lis when you set `.OPTION LIS_NEW` in the netlist. See [Output Listing File](#) for discussion of the contents of the file.

```
*****
Project : $ring oscillator (Non-Alter) <Project name is the first
line of netlist>

*****Loading Files*****
      Loading... ' <path>/Main File.sp'
      Loading ' <path>/netlist.spc'
      Loading ' <path>/library.lib'
*****Options used simulation file*****
      .option brief list converge=1
*****HSPICE Convergence Details*****
      Trying... Newton Iteration..

      Message: Newton Iteration convergence
failure!,resetting dcon option to 1 and retrying

      Trying... damped pseudo-transient..

      Message: Success!
      (Iterations: 1023)
      Message: (mention if compromised with accuracy,
dcgmin etc.)

      Recommendation:
      You can speed up your simulation by specifying:
      .OPTION CONVERGE=1
***** Analysis Details*****
      Starting <DC/TRAN/AC/...> Analysis
      Step: 1ps
      End Time: 200ns
      writing output to file:top_wdf.tr0
      Format: WDF v2.0
      Precision: Single (32b)

Probed signals: 48

Simulation completed : 10% transient time: 20ns elapsed time: 10
mins
Simulation completed : 20% transient time: 40ns elapsed time: 21
mins
```

Chapter 12: Simulation Output

Output Listing (*.lis) File with .OPTION LIS_NEW Set

```
Simulation completed : 30% transient time: 60ns elapsed time: 30
mins
Simulation completed : 40% transient time: 80ns elapsed time: 42
mins
      Simulation completed : 50% transient time: 100ns
elapsed time: 50 mins
      Simulation completed : 60% transient time: 120ns
elapsed time: 1 hr 1 mins
      Simulation completed : 70% transient time: 140ns
elapsed time: 1hr 10 mins
      Simulation completed : 80% transient time: 160ns
elapsed time: 1hr 21 mins
      Simulation completed : 90% transient time: 180ns
elapsed time: 1hr 40 mins
Simulation completed : 100% transient time:200ns elapsed time:
1hr 60 mins
***** (Print info)**<e.g., .option
LIST,OPTS,NODES,etc.>

*****Measured Values for the Netlist*****

      t_period= 1.0330E-07 targ= 1.5498E-06 trig= 1.4465E-06

*****

***** Circuit Statistics *****
Resistors           : 480
Capacitors          : 282
Current Sources     : 1
MOSFETs             : 22
Voltage Sources     : 24
*****
Total Elements      : 65
Total Nodes         : 47
*****

*****Resource Summary*****
***** HSPICE Multi-Threading Info ***** < To be printed only
if multi-threading is used >
Command Line Threads Count:           1
Available CPU Count:                   2
Actual Model Evaluation(Load) Threads Count: 1
Actual Solver Threads Count:           1

*****
***** HSPICE Multi-Processing Info ***** < To be printed only
if multi-processing is used >
Command Line Core Count:                1
Available CPU core Count:                2
```

```
Actual Model Evaluation(Load) Core Count:      1
Actual Solver Core Count:                    1
```

```
*****
***** Runtime Summary (seconds) *****
Analysis          Time      # Points   tot. iter  conv.iter
op point          1.01           1          455
transient         36.73       3000001     88865     31379 rev=0
readin            0.24
errchk            0.03
setup             0.04
output            0.00
peak memory used          1253 kbytes
total cpu time          12338.31 seconds
total elapsed time      11241 seconds
job started at      09:00:05 05/09/2008
job ended   at      09:00:46 05/09/2008
*****
```

In the Runtime Summary, note the following definitions:

- # Points: TSTOP/TSTEP+1
- tot.iter: Total number of iterations in transient analysis taken by the tool to find the solution.
- conv.iter: Count of only the iterations that converge. This number equals the number of time points which HSPICE evaluates to form the waveforms.
- rev (on the transient row): Number of times that the simulator had to reject timestep (reversals). This measures how difficult the design is to simulate; if rev is very high, it means the circuit is difficult to converge. Each rejected timestep equals eight iterations and no solution.
- When you use .OPTION LIS_NEW=1, any .PRINT statement in your netlist generates a text file containing the simulation results. For a transient analysis, the file has the extension, .printtr#.

For MOSFET Information Use .OPTION LIST

To print the effective width and length after scaling include .OPTION LIST in your netlist. Then search for mosfets in your listing file. For example:

```
**** mosfets
**** BSIM4 Model (Level 54)
element name      1:mn1          1:mn2          1:mn3          1:mn4
```

Chapter 12: Simulation Output

For MOSFET Information Use .OPTION LIST

drain	1:104	1:158	1:164	1:160
gate	1:0	1:102	1:214	1:106
source	1:157	1:195	1:159	1:165
bulk	0:vss	0:vss	0:vss	0:vss
model	0:nch	0:nch	0:nch	0:nch
w eff	288.0000n	288.0000n	288.0000n	288.0000n
l eff	90.0000n	90.0000n	90.0000n	90.0000n

You can see `w eff` and `l eff` (scaled) for each MOS instance in the circuit name directory section. The "1" in `1:mn1` refers to the subckt instance number:

```
***** circuit name directory
circuit number to circuit name directory
  number circuitname          definition          multiplier
    0 main circuit
    1 xmn1.                   nch_mac             1.00
```

Where, "1:mn1" refers to `xmn1.mn1`.

You can also use HSPICE MOSFET output templates. For details see [MOSFET Output Templates](#) in the *HSPICE Reference Manual: MOSFET Models*. To illustrate here, the general syntax is:

```
.print tran | dc LV1(mn1)  $ effective length for all MOSFET
                           models except levels 54, 57, 69 and 70.
.print tran | dc LV2(mn1)  $ effective width for all MOSFET
                           models except levels 54, 57, 69 and 70.
.print tran | dc LX63(mn1) $ effective length for MOSFET model
                           levels 54, 57, 69 and 70.
.print tran | dc LX62(mn1) $ effective width for MOSFET model
                           levels 54, 57, 69 and 70.
```

For MOS models inside subckts, preface the MOS instance name with the subckt name:

```
.print tran LX62(xmn1.mn1) $ using the subckt name
.print tran LX62(1:mn1)    $ using the subckt number
```

You can also wildcard the subckt name and/or the MOS instance name:

```
.print tran LX63(*.mn1)    $ all MN1 lengths one level down
.print tran LX63(XMN1.*)   $ all MOS instance lengths below XMN1
.print tran LX63(*.*)      $ all MOS instance lengths two levels down
.print tran LX63(*)         $ all MOS instance lengths in the design
```

The following is a complete example, naming the outputs width and length:

```
.print tran width=par('lx62(1:mn1)') length=par('lx63(1:mn1)')
```


The output in your listing file then appears as follows:

time	width	length
0.	288.0000n	90.0000n
100.00000p	288.0000n	90.0000n
200.00000p	288.0000n	90.0000n
300.00000p	288.0000n	90.0000n

HPP Status Updates in *.lis File

When you run HPP, the *.lis file shows running updates for the simulation status and the CPU utilization. In the example below the lines translate as follows:

Percentage of simulation completion time = simulation time (microseconds or nanoseconds) us, ns) (etc = (estimated time to completion) sec, ett = (estimated total time)

(wall = (wall clock time) cpu = (total cpu time) s=(number of CPUs - utilization)

```
Running HSPICE Precision Parallel (HPP)
NDD transient simulation
```

```
.
HSolve ...
```

```
NDD 128
```

```
Running 12 threads.
```

```
... ..
```

```
0.1% time = 36.651039 ns ( etc = 16:26:42, ett = 16:27:41 )
( wall = 59.3 sec cpu = 10:17 s=10.4114 )
```

```
0.2% time = 75.222362 ns ( etc = 8:29:41, ett = 8:30:42 )
( wall = 1:01 cpu = 10:40 s=10.4468 )
```

```
0.3% time = 107.888373 ns ( etc = 5:45:15, ett = 5:46:17 )
( wall = 1:02 cpu = 10:52 s=10.4711 )
```

```
0.5% time = 175.565655 ns ( etc = 3:29:14, ett = 3:30:17 )
( wall = 1:03 cpu = 11:01 s=10.4822 )
```

```
0.6% time = 222.354196 ns ( etc = 2:56:22, ett = 2:57:26 )
( wall = 1:03 cpu = 11:10 s=10.4989 )
```

```
0.8% time = 282.247477 ns ( etc = 2:13:33, ett = 2:14:37 )
```

Chapter 12: Simulation Output

Verilog-A Simulation Output

```
( wall = 1:04 cpu = 11:19 s=10.5132 )
****
99.5% time = 34.825176 us ( etc = 25.4 sec, ett = 2:16:27 )
( wall = 2:16:02 cpu = 1:02:18:07 s=11.6009 )
99.6% time = 34.860195 us ( etc = 22.0 sec, ett = 2:16:30 )
( wall = 2:16:08 cpu = 1:02:19:17 s=11.601 )
99.7% time = 34.895010 us ( etc = 16.9 sec, ett = 2:16:37 )
( wall = 2:16:20 cpu = 1:02:21:46 s=11.6011 )
99.8% time = 34.930731 us ( etc = 11.6 sec, ett = 2:16:46 )
( wall = 2:16:35 cpu = 1:02:24:32 s=11.6011 )
99.9% time = 34.966635 us ( etc = 5.9 sec, ett = 2:16:46 )
( wall = 2:16:40 cpu = 1:02:25:36 s=11.601 )
100.0% time = 35.000000 us ( etc = 0.0 sec, ett = 2:16:44 )
( wall = 2:16:44 cpu = 1:02:26:23 s=11.6009 )
*****
```

Verilog-A Simulation Output

The HSPICE standard output files consist of these basic files:

- The *.valog file, Verilog-A log file, which contains Verilog-A specific message from compiling and simulating phase. The contents of *.valog file also echoes to the *.lis file.
- Compiled Verilog-A code (*.pvalib file) (when you compile Verilog-A modules manually).

Verilog-A Output Directory

The Verilog-A output directory -o.pvadir/ contains the following shared output (*.so) intermediate files on Linux/UNIX (and *.dll files on the Windows platform).

File Type	Contains
pvaRTL.log	Log file output from gcc compilation output
pvaRTL_linux .so	Runtime library input to HSPICE/ HSPICE RF simulators
pvaEnv	Storage information for incrementally compiling user Verilog-A files

For example:

Command-line statement	Creates this directory:
<code>hspice test.sp -o output</code>	<code>output.pvadir</code>
<code>hspice test.sp</code>	<code>test.pvadir</code>
<code>hspice test.sp -o test</code>	<code>test.pvadir</code>
<code>hspice test.sp -o /remote/hspice/benchmark</code>	<code>/remote/hspice/ benchmark.pvadir</code>

For further information, see [Using Verilog-A](#) in the *HSPICE User Guide: Advanced Analog Simulation and Analysis*.

Field Solver *.str File

HSPICE creates an *.str file when a field solver model is created. The *.str file is a tcl interpretation of the layerstack. View the layerstack by using the "wish" command on Linux machines. (No equivalent command exists on Windows machines.) Mouse over the dielectric and conductors to see the characteristics of the materials reported in the lower left corner of the window.

Chapter 12: Simulation Output

Redirecting the Simulation Output Results Files to a Different Directory

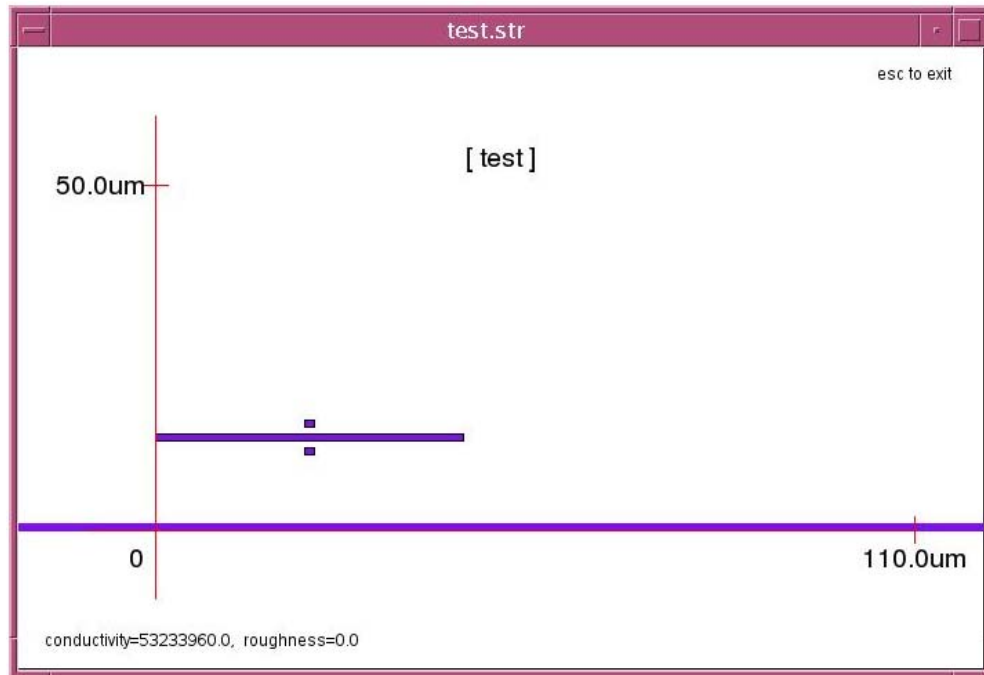


Figure 39 Field Solver *.str file

While you can use the “wish” command to view your layerstack, you cannot zoom, edit, or netlist out a new stack. You can edit the *.str file. However, any changes you make are not reflected in your field solver model.

For detailed information, see [Visualizing Cross-Sectional Geometric Information](#) in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

Redirecting the Simulation Output Results Files to a Different Directory

If you need to redirect the simulation-output result files to a directory other than the current working directory, use either of the following two options. At a command line prompt, enter either:

```
% hspice -i test.sp -o /root/user/hspice/result/test.lis
```

HSPICE redirects the simulation results to the specified location /root/user/hspice/result. Alternatively, enter:

```
% hspice -i test.sp -o results/test.lis
```

Where, HSPICE redirects the simulation results to the specified folder with respect to the current working directory. However, you should create the destination folder before you start the simulation. Otherwise, HSPICE returns an error message and aborts.

Directing .PRINT Output to a Separate File

Set `.OPTION LIS_NEW` in your netlist, to print the data from `.PRINT` statements to a separate file.

Refer to `.OPTION LIS_NEW` in the *HSPICE Reference Manual: Commands and Control Options* for more details.

Getting Data Out of HSPICE Plot Files

The waveform file formats `*.tr0`, `*.ac0`, etc., are proprietary formats. HSPICE designed these formats for native waveform viewers and not for other programs or scripts. HSPICE does not publish proprietary formats, as these files can change between releases following enhancements.

However, there are some alternatives for extracting waveform data from HSPICE. The following example illustrates some common methods. This example performs a simple frequency sweep of a tank circuit and creates some AC analysis points. At about 160Hz (the resonant frequency of the circuit) the current drops to its lowest point then goes back up. The `.probe` command selects the signals desired in the table data. Here is the netlist:

```
Title: tank circuit frequency sweep
* reference: http://www.allaboutcircuits.com/vol_2/chpt_6/2.html
.option ingold=1 probe resmin=1e-15 csdf
v1 1 0 ac 1
c1 1 0 10u ic=1
r1 1 2 1e-12
l1 2 0 100m ic=.1
.ac lin 20 100 200
.print ac im(v1)
.plot ac im(v1)
.probe ac im(v1)
.stim data filename=foo my_current im(v1)
.end
```

Likely, the best option is to use **File > Export Waveform Data** in Custom WaveView (or CTRL-E).

1. You name the output file, set a step size, and the result looks like this:

```
#format table ## [Custom WaveView] 14:00:27 Mon Aug 24 2009
FREQ          im(v1)
 1.000E+002   9.632E-003
 1.010E+002   9.418E-003
 1.020E+002   9.204E-003
 1.030E+002   8.990E-003
 1.040E+002   8.776E-003
 1.050E+002   8.562E-003
...
```

2. You can use `.option csdf` (instead of `post`) to create a column-delimited format that is much easier to parse, especially with a limited number of signals. The `*.ac0` file still stores this data, in but is now in the CSDF format.

```
#N 'im(v1) '
#C 1.00000e+002   1   9.63231e-003 /0.0
#C 1.05263e+002   1   8.50584e-003 /0.0
#C 1.10526e+002   1   7.45516e-003 /0.0
#C 1.15789e+002   1   6.46993e-003 /0.0
...
```

Note: If you accidentally leave `.option post` in your netlist and it occurs after `csdf`, the file reverts back to the traditional plot file format and HSPICE issues the message:

```
**warning** multiple output options specified, using post
```

3. Use the “stim” data card. In the example above, you create an ASCII file `foo.dat0_ac0`. However, the file does not preserve the frequency values, only the magnitudes. In the `.stim` card, “data” is a keyword that creates a file which HSPICE can read in with the `.data` statement but is also user-readable. The output file is takes the name “foo”, and the data set name is “my_current”.

```
.data my_current
imv1
 9.632e-03
 8.506e-03
 7.455e-03
 6.470e-03
 5.542e-03
 4.663e-03
...
```

4. When you use `.print` statements, the listing file saves the data in column form. You have to cut and paste it from the listing file, but the process is effective.

```
freq      i mag
          v1
100.00000  9.632e-03
105.26316  8.506e-03
110.52632  7.455e-03
115.78947  6.470e-03
121.05263  5.542e-03
...
```

To understand how to control the time point intervals when using `.print` to output data to the listing file, see [OPTION INTERP](#) in the *HSPICE Reference Manual: Commands and Control Options*.

5. Use the HSPICE Output Converter Utility, documented in the following section, [Using the HSPICE Output Converter Utility](#). The converter utility converts `*.tr#`, `*.ac#`, and `*.sw#` files to PSF and PWL/DATA/VEC files.

A final important point is when dealing with AC voltages and currents: be sure to specify which of the complex parts you want in your output. The default is real + imaginary but you can select which components you get by adding a “modifier” after the `v` for voltage:

- `.print ac vm(v1)` — voltage magnitude
- `.print ac vr(v1)` — the real part
- `.print ac vi(v1)` — the imaginary part
- `.print ac vdb(v1)` — voltage in dB

For a discussion of how to control the time point intervals of data in a plot file, refer to [How TSTEP Affects a Transient Simulation](#).

Compressing Waveform Files

With the F-2011.09 version, you can generate compressed waveform files. You need to add the `-gz` command to the `hspice` command line. For example:

```
hspice -i test.sp -o test -gz
```

HSPICE generates a compressed format waveform file named `test.tr0.gz`. For many cases, your use of compression saves disk space by 30% to 40%. Compression is supported for transient, AC and, DC analysis. You

can use Custom Explorer/Waveview to view the compressed waveform files generated by HSPICE.

Using the HSPICE Output Converter Utility

This section describes how to convert output generated by HSPICE.

The converter utility is a post-process tool that converts the output files (* .tr#, * .ac#, and * .sw#) generated by HSPICE. Use the converter to get the Parameter Storage Format (PSF) output files directly from the .tr#, .ac#, or .sw# files generated by HSPICE with the POST output control option. Or, use the converter to get the PWL Source, DATA Card, and Digital Vector File (VEC) from the * .tr#, * .ac#, and * .sw# files generated by HSPICE with the POST or CSDF control options. You can reuse these stimuli in a new simulation.

Note: Standalone HSPICE RF does not support the converter utility.

The following sections discuss these topics:

- [PSF Converter](#)
- [PWL/DATA/VEC Converter](#)

PSF Converter

Table 47 Supported Platforms

Linux RHEL	Linux SUSE	Solaris	Windows	IBM AIX5.1
Yes	Yes	Yes	Yes	Yes

Syntax

```
converter -t PSF -i input_file [-o output_file] [-a | -b]
```

Table 48 PSF Converter Parameters

Argument	Description
-t	Specifies the file type (must be psf.)

Table 48 PSF Converter Parameters (Continued)

Argument	Description
-i	Specifies input file name. The input file must be the output file that HSPICE generates with the POST output control option.
-o	Specifies output file name. The converter assigns a <code>.psf</code> as the extension of the output file. If you do not specify the output file name, the converter appends <code>_psf</code> to the root name of the input file, and it remains the extension of the input file.
-a	Specifies the ASCII format for the output file.
-b	Specifies the binary format for the output file. By default, the output file is in binary format. The content included in angled brackets (< >) is optional.

Example

```
converter -t PSF -i testpost.tr0 -o testpsf
```

The input file is `testpost.tr0`, which HSPICE generates with the POST option. The output file name is `testpsf`. After running, HSPICE generates two new files: `testpsf.psf` and `logfile`. The `testpsf.psf` file is a PSF file that you can view with the Analog Waveform Display (AWD). The `logfile` is necessary for the AWD to load the waveform.

PWL/DATA/VEC Converter

Linux SUSE	Linux RHEL	Solaris	Windows	IBM
Yes	Yes	Yes	Yes	Yes

The PWL/DATA/VEC Converter is mainly for reusing previous simulation results directly from the `*.tr#`, `*.ac#`, and `*.sw#` files produced by HSPICE. The converter is in accordance with the `.STIM` statement in the HSPICE netlist.

Syntax

```
converter -t PWL/DATA/VEC -i input_file <-o output_file>
```

Table 49 PWL/DATA/VEC Converter Parameters

Argument	Description
-t	Specifies the type of the stimulus (PWL).
-i	Specifies the input file name. Input files are the output files which HSPICE generates with the <code>POST=x</code> or <code>CSDF=x</code> output control options.
-o	Specifies the output file name. If you do not specify the output file name, the converter automatically assigns the following file names: <ul style="list-style-type: none">▪ <code>input_filename.tr#_PWL#</code>▪ <code>input_filename.ac#_DAT#</code>▪ <code>input_filename.tr#_DAT#</code>▪ <code>input_filename.sw#_DAT#</code>▪ <code>input_filename.tr#_VEC#</code>

The content included by angled brackets (< >) is optional.

Note: Transient analysis must generate the input file for PWL and VEC.

Prompt User Mode

The PWL/DATA/VEC Converter is a prompt user mode. The converter displays corresponding prompts and asks you to input some data after you start it successfully.

Input the following at the command line and press the Enter key:

```
Converter -t PWL -i sample.tr0
```

The following input prompts appear one at a time and require your specified entries on the command line.

1. Enter the number of output variables (>0):
Specify the number of output variables from the waveform file to convert.
2. Enter output variables reused:
Specify the name of the node(s) in the design to convert. The node names must match a node name in the waveform file that you are converting.
3. Enter name of the source (optional):

If you do not specify a source, the source name is *vmnode_name*.

4. Enter positive node name (optional):

If you do not specify a positive node, the positive node name is be the same as node name(s) specified for the output variable(s).

5. Enter negative node name (optional):

If you do not specify a negative node, HSPICE specifies 0 (ground) as the negative node for each node name.

6. Enter independent variable type [1--from/to, 2--dispersed]:

This input line is optional. If you do in put anything and you press the Enter key, the input prompts end, the executable automatically runs and generates the *design_name.tr0_PWL0* stimuli file that contains all time points from the original waveform file.

If you specify an independent variable type, the utility displays the following prompts. For each prompt, you need to provide a value.

7. For 1-- from/to,
Enter the start point:
Starting point of the output file.
8. Enter the end point:
Ending point of the output file.
9. Enter the number of output points:
Number of output points.
10. For 2-- :dispersed
Enter the dispersed points:
Enter a list of time points you want to write to the file.

Once you enter the necessary information at the last prompt and press the Return key, the executable automatically runs and generates the *design_name.tr0_PWL0* stimuli files.

Input Line Dependencies

The input lines you use must adhere to the following conditions:

Chapter 12: Simulation Output

Using the HSPICE Output Converter Utility

- Variables used in a PWL source must be voltage or current signals.
- Variables used in a VEC file can only be voltage signals.
- PWL Source Names must begin with V or I.
- Dispersed time points must be increasing in value when the stimulus type is PWL or VEC.
- For the optional items, you can enter the Return Key directly to adopt the default value.

Running the Converter Utility in Batch Mode

While the converter is interactive, prompting you with a series of questions, you can run in batch mode by redirecting input from an “answer” file.

The command to run the converter in batch mode has two parts and requires two files. The first file (see the following batch file), invokes the converter and tells it the waveform file to use. The second file is the “answer file” containing the answers to the conversion questions. The section titled [Prompt User Mode](#) lists the questions asked by the converter. You can create sample batch files using the following syntax:

```
converter -t pwl -i file1.tr0 < answers1.txt
converter -t pwl -i file2.tr0 < answers2.txt
```

where, file1.tr0 and file2.tr0 are HSPICE generated transient output files. The above creates file1.tr0_PWL0 and file2.tr0_PWL0.

Examples of input files with answers:

```
// single PWL created :
    1           // # of signals
    v(nd)       // names of signals
    vsig1       // name of PWL source
    sig1        // + node of PWL
    0           // - node of PWL
    1           // choose 1 for from/to , 2 to define each point
    0           // start time
    600p        // end time
    100         // # of points
// Answer file to create multiple PWL signals in one tr0_PWL0 file
// after first answer, an answer is needed for each
// signal even if they are the same
    2
    v(sig1) v(sig2)
    vsig1 vsig2
    n1 n2
```

```
0 0
1 1
0 0
100n 100n
100 100
```

Troubleshooting Issues

The following sections discuss these topics:

- [Resolving Inductor/Voltage Source Loop Errors](#)
- [Voltage Source Missing Rising and Falling Edges](#)

Resolving Inductor/Voltage Source Loop Errors

HSPICE issues an inductor/voltage source loop error when:

- Two or more voltage sources connect to the same nodes.
- A voltage source with an inductor connects directly across its nodes.
- Two or more inductors connect in a loop and there is no limit to the current.

Avoid using these topologies.

However, if HSPICE reports this error, then follow these steps to correct the error:

1. Find out where the topology exists and correct it.
2. Combine multiple voltage sources into a single equivalent voltage source.
3. Limit the current by connecting a small series resistance (1n ohm or smaller) to the voltage source loop.

Voltage Source Missing Rising and Falling Edges

If you define rise and fall times in an independent voltage source, and the rise and fall times are missing when you look at the waveform of the source, it is because HSPICE defines the source as:

```
V1 in 0 pulse 0 5 10n 1n 1n 200n 333n
.option post=2
```

See [Figure 40](#) for the resulting waveform.

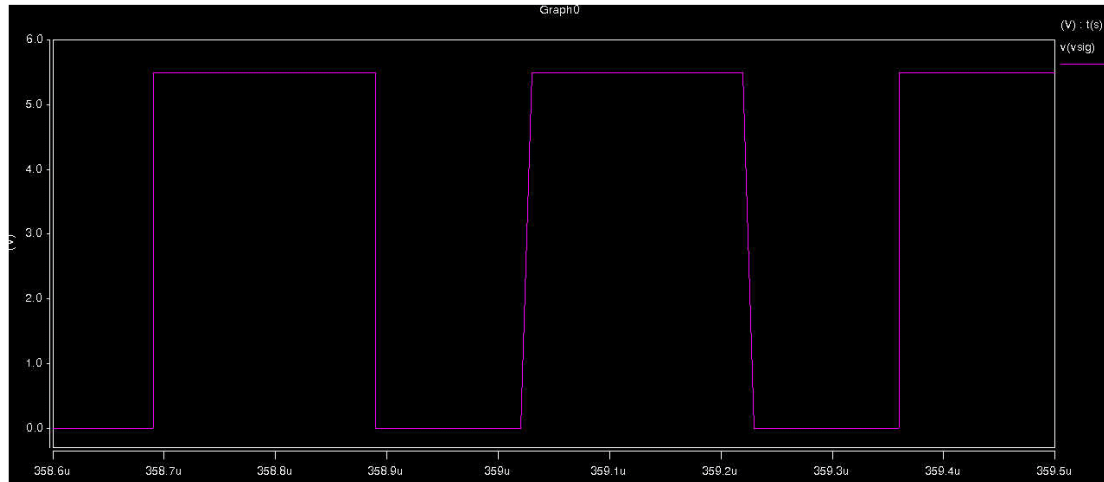


Figure 40 Rise and Fall times missing

When you set `.option POST=2`, HSPICE prints the waveform file as ASCII data.

When used with the default post-processing output version, `POST_VERSION=9601`, limits the number of significant digits. This can cause a loss of resolution in the waveforms.

If you set `.option POST_VERSION=2001` in addition to `.option POST=2`, then the ASCII waveform data contains more significant digits and the resolution increases and the rising and falling edges are present ([Figure 41](#)).

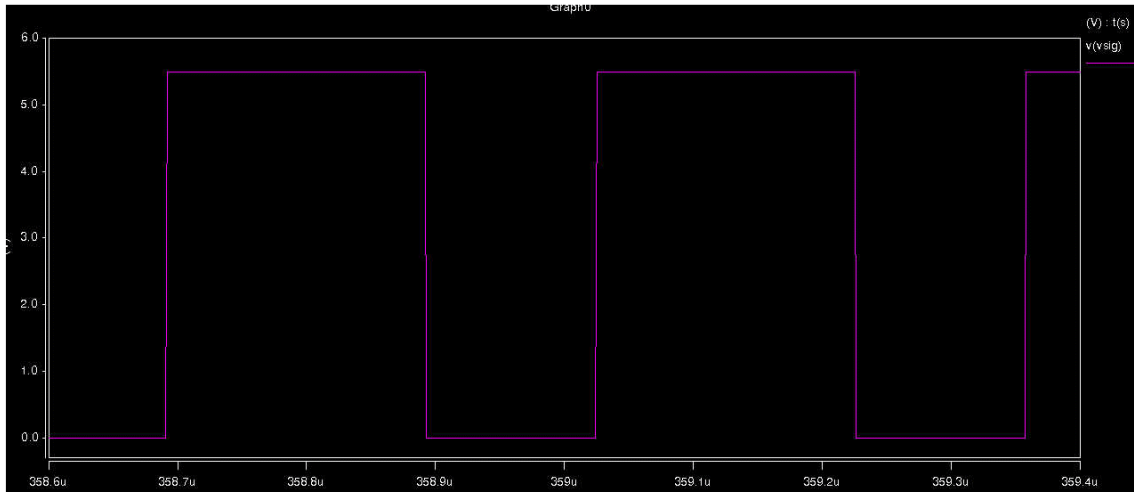


Figure 41 Rising, falling edges present

Using `POST_VERSION=2001` ensures that file header displays the correct number of output variables when the number of variables exceeds 9999.

Chapter 12: Simulation Output
Troubleshooting Issues

Part 3: Analyses and Simulation

This Part contains the following chapters/topics:

- [Chapter 13, Initializing DC-Operating Point Analysis](#)
- [Chapter 14, AC Small-Signal and Noise Analysis](#)
- [Chapter 15, Transient Analysis](#)
- [Chapter 16, Spectrum Analysis](#)
- [Chapter 17, Pole-Zero Analysis](#)
- [Chapter 18, Performing Digital Cell Characterization](#)
- [Chapter 19, MOSFET Model Reliability Analysis \(MOSRA\)](#)
- [Chapter 20, Post-Layout Simulation: RC Network Reduction and Back-Annotation](#)
- [Chapter 21, Multi-Technology Simulation of 3D Integrated Circuit](#)

The following analyses are described in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*

- [Linear Network Parameter Analysis](#)
- [Statistical Eye Analysis](#)

Initializing DC-Operating Point Analysis

Describes DC initialization and operating point analysis.

HSPICE ships numerous examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

For descriptions of individual HSPICE commands referenced in this chapter, see the [HSPICE Reference Manual: Commands and Control Options](#). For discussion of use of the `.DC` command in a subckt block see [Using Isomorphic Analyses in Subckt Blocks on page 36](#).

The following sections cover these topics:

- [Simulation Flow—Initialization and Analysis](#)
- [DC Initialization and Operating Point Calculation](#)
- [.DC Statement—DC Sweeps](#)
- [Other DC Analysis Statements](#)
- [Accuracy and Convergence](#)
- [Reducing DC Errors](#)
- [Diagnosing Convergence Problems](#)

Simulation Flow—Initialization and Analysis

Before it performs `.OP`, `.DC` sweep, `.AC`, or `.TRAN` analyses, HSPICE first sets the DC operating point values for all nodes and sources. To do this, HSPICE does one of the following:

Chapter 13: Initializing DC-Operating Point Analysis

Simulation Flow—Initialization and Analysis

- Calculates all values
- Applies values specified in `.NODESET` and `.IC` statements
- Applies values stored in an initial conditions file.

The `.OPTION OFF` statement, and the `OFF` and `IC=val` element parameters, also control initialization.

Initialization is fundamental to simulation. HSPICE starts any analysis with known-nodal voltages (or initial estimates for unknown voltages) and some branch currents. It then iteratively finds the exact solution. Initial estimates that are close to the exact solution increase the likelihood of a convergent solution and a lower simulation time.

A transient analysis first calculates a DC operating point using the DC equivalent model of the circuit (unless you specify the `UIC` parameter in the `.TRAN` statement). HSPICE then uses the resulting DC operating point as an initial estimate to solve the next timepoint in the transient analysis.

The following describes the process:

1. If you do not provide an initial guess or if you provide only partial information, HSPICE provides a default estimate for each node in the circuit.
2. HSPICE then uses this estimate to iteratively find the exact solution.

The `.NODESET` and `.IC` statements supply an initial guess for the exact DC solution of nodes within a circuit.

3. To set the seed value for the iterative dc algorithm for any circuit node to any value, use the `.NODESET` statement.
4. HSPICE then connects a voltage source equivalent, to each initialized node (a current source, with a `GMAX` parallel conductance, set with a `.OPTION` statement).
5. HSPICE next calculates a DC operating point, with the `.NODESET` voltage source equivalent connected.
6. HSPICE disconnects the equivalent voltage sources, which you set in the `.NODESET` statement, and recalculates the DC operating point.

This is the DC operating point solution.

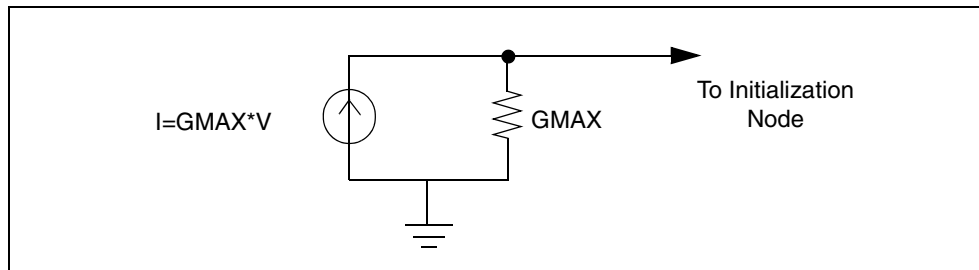


Figure 42 Equivalent Voltage Source: `NODESET` and `.IC`

The `.IC` statement provides both an initial guess and a solution for selected nodes within the circuit. Nodes that you initialize with the `.IC` statement become part of the solution of the DC operating point.

Use `.IC` and `.NODESET` statements in a `.DC` analysis, in addition to `.TRAN` statements, unless you set `.OPTION DCIC=0`. You can also use the `OFF` option to initialize active devices. The `OFF` option works with `.IC` and `.NODESET` voltages as follows:

1. If the netlist includes any `.IC` or `.NODESET` statements, HSPICE sets node voltages, according to those statements.
2. If you set the `OFF` option, HSPICE sets values to zero for the terminal voltages of all active devices (BJTs, diodes, MOSFETs, JFETs, MESFETs) that you do not set in `.IC` or `.NODESET` statements, or by sources.
3. If element statements specify any IC parameters, HSPICE sets those initial conditions.
4. HSPICE uses the resulting voltage settings, as the initial guess at the operating point.

Use `OFF` to find an exact solution, during an operating point analysis, in a large circuit. The majority of device terminals are at zero volts for the operating point solution. To initialize the terminal voltages to zero for selected active devices, set the `OFF` parameter in the element statements for those devices.

After HSPICE finds a DC operating point, use `.SAVE` to store operating-point node voltages in a `design.ic` file. Then use the `.LOAD` statement to restore operating-point values from the `*.ic` file for later analyses.

When you set initial conditions for Transient Analysis:

Chapter 13: Initializing DC-Operating Point Analysis

Simulation Flow—Initialization and Analysis

- If you include UIC in a `.TRAN` statement, HSPICE starts a transient analysis, using node voltages specified in an `.IC` statement.
- Use the `.OP` statement, to store an estimate of the DC operating point, during a transient analysis.
- An internal timestep too small error message indicates that the circuit failed to converge. The cause of the failure can be that HSPICE cannot use stated initial conditions to calculate the actual DC operating point.

Figure 43 shows the simulation flow for DC analysis in HSPICE.

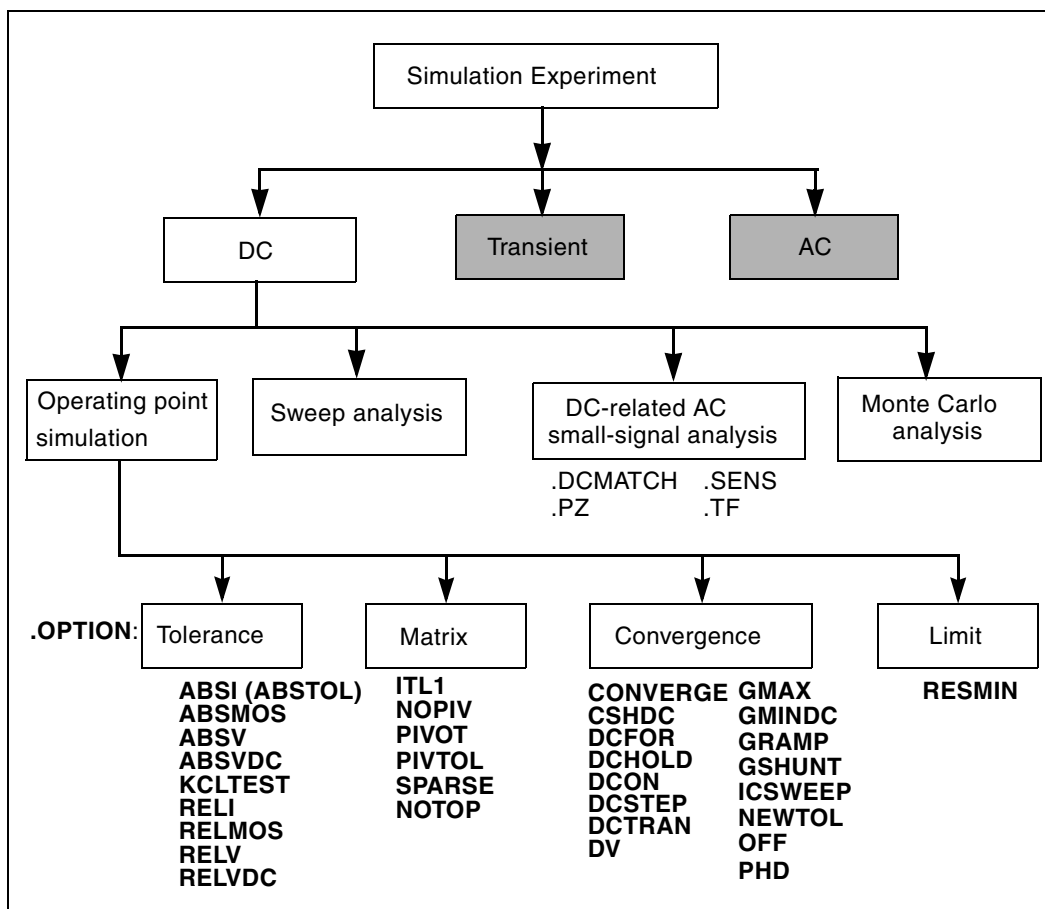


Figure 43 DC Initialization and Operating Point Analysis Simulation Flow

DC Initialization and Operating Point Calculation

Use a `.OP` statement in HSPICE to:

- Calculate the DC operating point of a circuit
- Produce an operating point during a transient analysis

A simulation can only have one `.OP` statement.

The following sections discuss these topics:

- [.OP Statement — Operating Point](#)
- [Element Statement IC Parameter](#)
- [Initial Conditions and UIC Parameters](#)
- [.SAVE and .LOAD Statements \(HSPICE Only\)](#)

.OP Statement — Operating Point

When you include an `.OP` statement in an input file, HSPICE calculates the DC operating point of the circuit. You can also use the `.OP` statement to produce an operating point, during a transient analysis.

If an analysis requires calculating an operating point, you do not need to specify the `.OP` statement; HSPICE calculates an operating point. If you use a `.OP` statement, and if you include the UIC keyword in a `.TRAN` analysis statement, then simulation omits the time=0 operating point analysis, and issues a warning in the output listing.

Chapter 13: Initializing DC-Operating Point Analysis

DC Initialization and Operating Point Calculation

Output

```
***** OPERATING POINT INFORMATION  TNOM=25.000  TEMP=25.000
***** OPERATING POINT STATUS IS ALL  SIMULATION TIME IS 0.
NODE      VOLTAGE  NODE      VOLTAGE  NODE      VOLTAGE
+ 0:2=0   0:3=437.3258M  0:4=455.1343M
+ 0:5=478.6763M  0:6=496.4858M  0:7=537.8452M
+ 0:8=555.6659M  0:10=5.0000  0:11=234.3306M

**** VOLTAGE SOURCES
SUBCKT
ELEMENT  0:VNCE  0:VN7  0:VPCE  0:VP7
VOLTS   0  5.00000  0  -5.00000
AMPS   -2.07407U  -405.41294P  2.07407U  405.41294P
POWER  0.  2.02706N  0.  2.02706N
TOTAL VOLTAGE SOURCE POWER DISSIPATION=4.0541 N WATTS
**** BIPOLAR JUNCTION TRANSISTORS
SUBCKT
ELEMENT  0:QN1  0:QN2  0:QN3  0:QN4
* Note: HSPICE RF does not support qn(element)
* charge output.
MODEL   0:N1    0:N1    0:N1    0:N1
IB      999.99912N  2.000000U  5.000000U  10.000000U
IC      -987.65345N  -1.97530U  -4.93827U  -9.87654U
VBE     437.32588M  455.13437M  478.67632M  496.48580M
VCE     437.32588M  17.80849M  23.54195M  17.80948M
VBC     437.32588M  455.13437M  478.67632M  496.48580M
VS      0.  0.  0.  0.
POWER   5.39908N  875.09107N  2.27712U  4.78896U
BETAD   -987.65432M  -987.65432M  -987.65432M  -987.65432M
GM      0.  0.  0.  0.
RPI     2.0810E+06  1.0405E+06  416.20796K  208.10396K
RX      250.00000M  250.00000M  250.00000M  250.00000M
RO      2.0810E+06  1.0405E+06  416.20796K  208.10396K
CPI     1.43092N  1.44033N  1.45279N  1.46225N
CMU     954.16927P  960.66843P  969.64689P  977.06866P
CCS     800.00000P  800.00000P  800.00000P  800.00000P
BETAAC  0.  0.  0.  0.
FT      0.  0.  0.  0.
```

Element Statement IC Parameter

Use the element statement parameter, `IC=<val>`, to set DC terminal voltages for selected active devices.

HSPICE uses the value, set in `IC=<val>`, as the DC operating point value.

Example

This example describes an H-element dependent-voltage source:

```
HXCC 13 20 VIN1 VIN2 IC=0.5, 1.3
```

The current, through VIN1, initializes to 0.5 mA. The current, through VIN2, initializes to 1.3 mA.

Initial Conditions and UIC Parameters

Use the `.IC` (or `.DCVOLT`), for the IC parameter on an element statement, and the `.NODESET` commands to set transient initial conditions in HSPICE. How it initializes depends on whether the `.TRAN` analysis statement includes the UIC parameter. If you do not specify the UIC parameter in the `.TRAN` statement, HSPICE computes the DC operating point solution before the transient analysis. The node voltages that you specify in the `.IC` statement determine the DC operating point.

HSPICE uses the node voltages that you specify in the `.NODESET` statement only in the first iteration to set an initial guess for the DC operating point analysis. Transient analysis releases the initialized nodes to calculate the second and later time points.

If you specify the UIC parameter in the `.TRAN` statement, HSPICE does not calculate the initial DC operating point, but directly enters transient analysis.

When you use `.TRAN` with UIC, HSPICE determines the `.TRAN` node values (at time zero) by searching for the first value found in this order: from `.IC` value, then IC parameter on an element statement, then `.NODESET` value; otherwise it uses a voltage of zero. Note that forcing a node value of the DC operating point may not satisfy KVL and KCL. In this event you likely see activity during the initial part of the simulation. This may happen if you use UIC and do not specify some node values, specify too many (conflicting) `.IC` values, or force node values and the topology changes. In this event, you likely see activity during the initial part of the simulation. Forcing a node voltage applies a fixed equivalent voltage source during DC analysis and transient analysis removes the voltage sources to calculate the second and later time points.

Therefore, to correct DC convergence problems use `.NODESET` commands (without `.TRAN` with UIC) liberally (when you can provide a good guess) and use `.ICs` sparingly (when you know the exact node voltage).

.SAVE and .LOAD Statements (HSPICE Only)

HSPICE saves the operating point, unless you use the `.SAVE LEVEL=NONE` statement. HSPICE restores the saved operating-point file, only if the input file contains a `.LOAD` statement.

The `.SAVE` statement in HSPICE stores the operating point of a circuit, in a file that you specify. You can save the operating point data as either an `.IC` or a `.NODESET` statement. For quick DC convergence in subsequent simulations, use the `.LOAD` statement to input the contents of this file. HSPICE saves the operating point by default, even if the HSPICE input file does not contain a `.SAVE` statement. To not save the operating point, specify `.SAVE LEVEL=NONE`.

A parameter or temperature sweep saves only the first operating point.

If any node initialization commands, such as `.NODESET` and `.IC`, appear in the netlist after the `.LOAD` command, then they overwrite the `.LOAD` initialization. If you use this feature to set particular states for multistate circuits (such as flip-flops), you can still use the `.SAVE` command to speed up the DC convergence.

`.SAVE` and `.LOAD` work even on changed circuit topologies. Adding or deleting nodes results in a new circuit topology. HSPICE initializes the new nodes, as if you did not save an operating point. HSPICE ignores references to deleted nodes, but initializes coincidental nodes to the values that you saved from the previous run.

When you initialize nodes to voltages, HSPICE inserts Norton-equivalent circuits at each initialized node. The conductance value of a Norton-equivalent circuit is `GMAX=100`, which might be too large for some circuits.

If using `.SAVE` and `.LOAD` does not speed up simulation, or causes simulation problems, use `.OPTION GMAX=1e-12` to minimize the effect of Norton-equivalent circuits on matrix conductances.

HSPICE still uses the initialized node voltages to initialize devices. Do not use the `.LOAD` command for concatenated netlist files.

.DC Statement—DC Sweeps

You can use the `.DC` statement in DC analysis to:

- Sweep any parameter value.
- Sweep any source value.
- Sweep temperature range.
- Perform a DC Monte Carlo (random sweep) analysis.
- Perform a data-driven sweep.
- Perform a DC circuit optimization for a data-driven sweep.
- Perform a DC circuit optimization, using start and stop.
- Perform a DC model characterization.

The `.DC` statement format depends on the application that uses it. For syntax details, refer to the `.DC` command in the *HSPICE Reference Manual: Commands and Control Options*.

The following sections discuss these topics:

- [Multi-dimensional DC Sweep](#)

Multi-dimensional DC Sweep

To get the measure results of a multi-dimensional `.DC` sweep in a single measure file, you need to create a dummy inner sweep, and use the `.DATA` block to sweep the multi-dimensional data points in the outer sweep.

In the following example, the `.DATA` block sweeps `pvdd` for each temperature. The measure results of all the 6 data points is recorded in the same measure file.

```
*****2D sweep measure setup
v0 1 0 pvdd
r0 1 0 1k

.param dummy=1 $ define dummy outer sweep
.dc dummy 1 1 1 SWEEP DATA=vpoint $ vpoint contains 2D sweep params
.DATA vpoint
+ temp pvdd
+ 15 1
+ 15 2
+ 15 3
+ 40 1
+ 40 2
+ 40 3
.ENDDATA
```

```
* Measure current at each pvdd and temperature value
.measure dc cur_res find i(r0) when par('dummy')=1
.end
```

Other DC Analysis Statements

HSPIICE also provides the following DC analysis statements. Each statement uses the DC-equivalent model of the circuit in its analysis. For `.PZ`, the equivalent circuit includes capacitors and inductors.

Statement	Description
<code>.DCMATCH</code>	A technique for computing the effects of local variations in device characteristics on the DC solution of a circuit.
<code>.PZ</code>	Performs pole/zero analysis.
<code>.SENS</code>	Obtains DC small-signal sensitivities of output variables for circuit parameters.
<code>.TF</code>	Calculates DC small-signal values for transfer functions (ratio of output variable, to input source).

HSPIICE includes DC control options, and DC initialization statements, to model resistive parasitics and initialize nodes. These statements enhance convergence properties and accuracy of simulation. This section describes how to perform DC-related, small-signal analysis.

DC Initialization Control Options

Use control options in a DC operating-point analysis, to control DC convergence properties and simulation algorithms. Many of these options also affect transient analysis because DC convergence is an integral part of transient convergence. Include the following options for *both* DC and transient convergence:

- Absolute and relative voltages
- Current tolerances
- Matrix options

See [.OPTION PHD](#) for the PHD flow that can show performance improvement in simulations that require large DC OP convergence iterations. Use `.OPTION` statements to specify the following options, which control DC analysis:

ABSTOL	DCFOR	DV	ICSWEEP	MAXAMP	PIVOT
CAPTAB	DCHOLD	GDCPATH	ITLPTRAN	NEWTOL	PIVTOL
CONVERGE	DCIC	GRAMP	ITL1	NOPIV	RESMIN
CSHDC	DCON	GSHDC	ITL2	OFF	SPARSE
DCCAP	DCSTEP	GSHUNT	KCLTEST	PHD	SYMB

DC and AC analysis also use some of these options. Many of these options also affect the transient analysis because DC convergence is an integral part of transient convergence. For a description of transient analysis, see [Chapter 15, Transient Analysis](#).

Accuracy and Convergence

Convergence is the ability to solve a set of circuit equations, within specified tolerances, and within a specified number of iterations. In numerical circuit simulation, a designer specifies a relative and absolute accuracy for the circuit solution. The simulator iteration algorithm then attempts to converge to a solution that is within these set tolerances. That is, if consecutive simulations achieve results within the specified accuracy tolerances, circuit simulation has converged. How quickly the simulator converges, is often a primary concern to a designer—especially for preliminary design trials. So designers willingly sacrifice some accuracy for simulations that converge quickly.

The following sections discuss these topics:

- [Accuracy Tolerances](#)
- [Autoconverge Process](#)

Accuracy Tolerances

HSPICE uses accuracy tolerances that you specify, to assure convergence. These tolerances determine when, and whether, to exit the convergence loop. For each iteration of the convergence loop HSPICE subtracts previously calculated values from the new solution and compares the result with the accuracy tolerances.

If the difference between two consecutive iterations is within the specified accuracy tolerances, the circuit simulation has converged.

$$| V_n^k - V_n^{k-1} | \leq \text{accuracy tolerance}$$

- V_n^k is the solution at the n timepoint for iteration k .
- V_n^{k-1} is the solution at the n timepoint for iteration $k - 1$.

As [Table 50](#) shows, HSPICE defaults to specific absolute and relative values. You can change these tolerances, so that simulation time is not excessive, and you do not compromise accuracy.

Table 50 Absolute and Relative Accuracy Tolerances

Type	.OPTION	Default
Nodal Voltage Tolerances	ABSVDC	50 μ v
	RELVDC	.001
Current Element Tolerances	ABSI	1 nA
	RELI	.01
	ABSMOS	1 μ A
	RELMOS	.05

HSPICE compares nodal voltages and element currents to the values from the previous iteration.

- If the absolute value of the difference is less than ABSVDC or ABSI, then the node or element has converged.

ABSV and ABSI set the floor value, below which HSPICE ignores values. Values above the floor use RELVDC and RELI as relative tolerances. If the iteration-to-iteration absolute difference is less than these tolerances, then it is convergent.

Note: ABSMOS and RELMOS are the tolerances for MOSFET drain currents.

Accuracy settings directly affect the number of iterations before convergence.

- If accuracy tolerances are tight, the circuit requires more time to converge.
- If the accuracy setting is too loose, the resulting solution can be inaccurate and unstable.

Table 51 shows an example of the relationship between the RELVDC value, and the number of iterations.

Table 51 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

Autoconverge Process

If a circuit does not converge in the number of iterations that `ITL1` specifies, HSPICE initiates an autoconvergence process. This process manipulates `DCON`, `GRAMP`, and `GMINDC`, and even `CONVERGE` in some cases. [Figure 44 on page 484](#) shows the autoconverge process.

Note: HSPICE uses autoconvergence in transient analysis, but it also uses autoconvergence in DC analysis if the Newton-Raphson (N-R) method fails.

In the process flow shown in [Figure 44 on page 484](#):

- Setting `.OPTION DCON=-1` disables Steps 2 and 3.
- Setting `.OPTION CONVERGE=-1` disables Steps 4 and 5.
- Setting `.OPTION DCON=-1 CONVERGE=-1` disables Steps 2, 3, 4, and 5.
- If you set the DV option to a value other than the default, Step 2 uses the value you set for DV, but Step 3 changes DV to 1e6.
- Setting `.OPTION GRAMP` has no effect on autoconverge. Autoconverge sets `GRAMP` independently.
- If you set `.OPTION GMINDC`, then `GMINDC` ramps to the value you set, instead of to 1e-12, in Steps 2 and 3.

Effects of GMINDC

The `GMINDC` option helps stabilize the circuit, during DC operating-point analysis. For MOSFETs, `GMINDC` helps stabilize the device in the vicinity of the threshold region. HSPICE inserts `GMINDC` between:

- Drain and bulk
- Source and bulk
- Drain and source

The drain-to-source `GMINDC` helps to:

- Linearize the transition from cutoff to weakly on
- Smooth-out model discontinuities
- Compensate for the effects of negative conductances.

The pn junction insertion of `GMINDC` in junction diodes linearizes the low conductance region. As a result, the device behaves like a resistor in the low-conductance region. This prevents the occurrence of zero conductance, and improves the convergence of the circuit. If a circuit does not converge, HSPICE automatically sets the `DCON` option. This option invokes `GMINDC` ramping, in steps 2 and 3 of [Figure 44 on page 484](#).

[Figure 45 on page 485](#) shows `GMINDC` for various elements.

Chapter 13: Initializing DC-Operating Point Analysis
Accuracy and Convergence

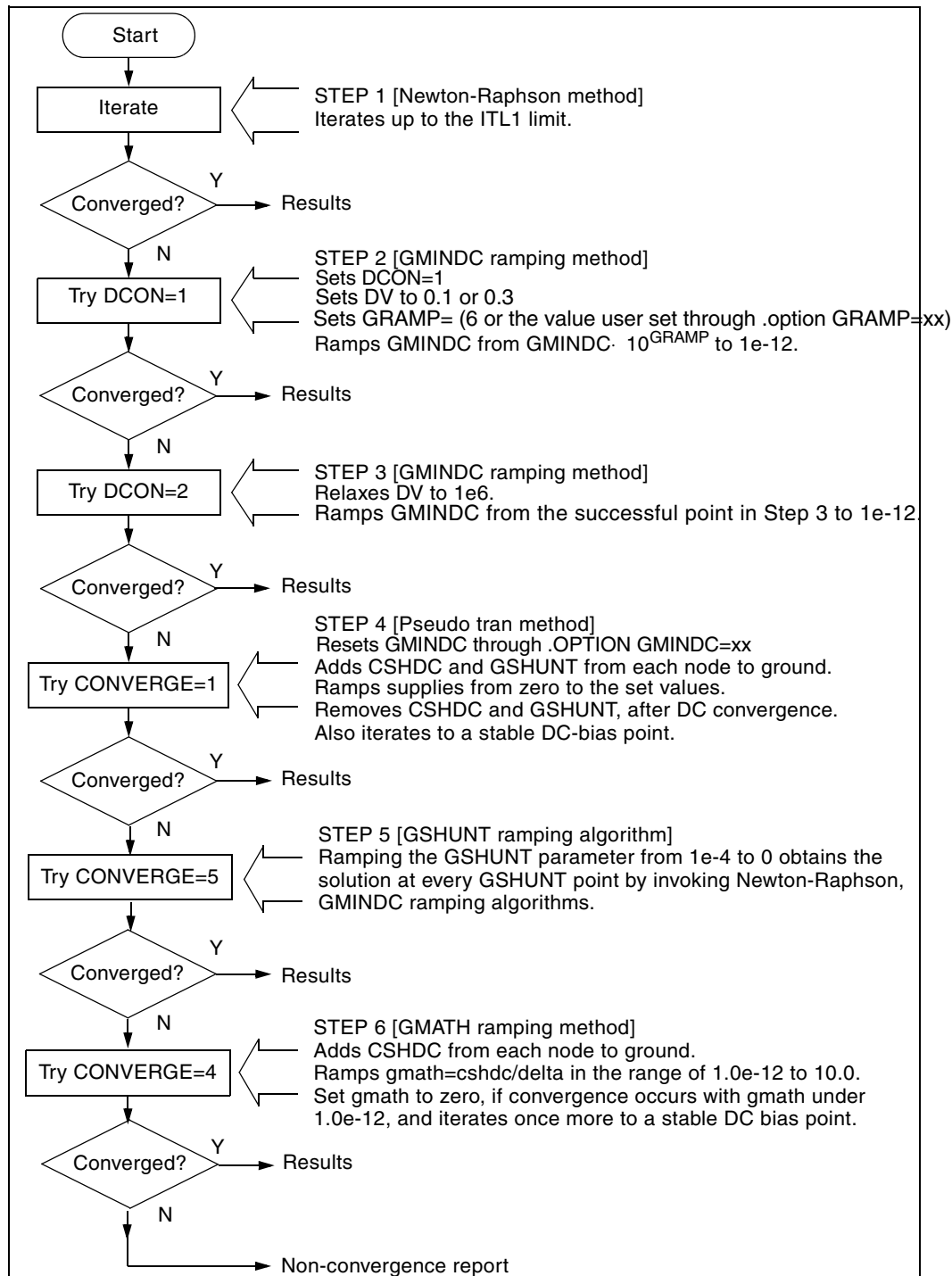


Figure 44 Autoconvergence Process Flow Diagram

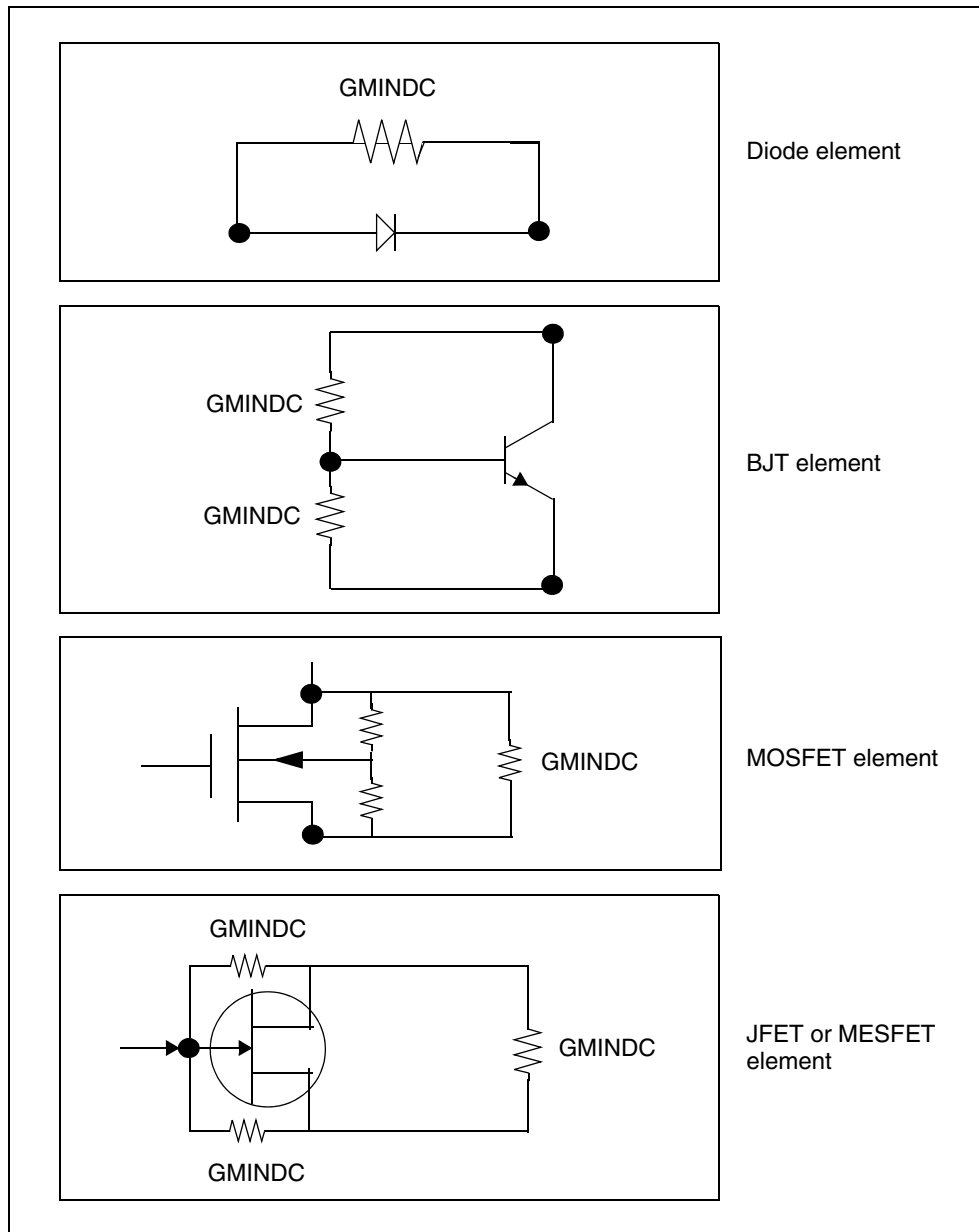


Figure 45 $GMINDC$ Insertion

Reducing DC Errors

To reduce DC errors, perform the following steps:

1. To check topology, set `.OPTION NODE`, to list nodal cross-references.
 - Do all MOS p-channel substrates connect to either VCC or positive supplies?
 - Do all MOS n-channel substrates connect to either GND or negative supplies?
 - Do all vertical NPN substrates connect to either GND or negative supplies?
 - Do all lateral PNP substrates connect to negative supplies?
 - Do all latches have either an OFF transistor, a `.NODESET`, or an `.IC`, on one side?
 - Do all series capacitors have a parallel resistance, or is `.OPTION DCSTEP` set?
2. Verify your `.MODEL` statements.
 - Check all model parameter units. Use model printouts to verify actual values and units because HSPICE multiplies some model parameters by scaling options.
 - Are subthreshold parameters of MOS models at reasonable values?
 - Are JS and JSW set in the MOS model for the DC portion of a diode model? A typical JS value is $1e-4A/M^2$.
 - Are CJ and CJSW set in MOS diode models?
 - Is weak-inversion NG and ND set in JFET/MESFET models?
 - Make sure that DIODE models have non-zero values for saturation current, junction capacitance, and series resistance.
 - Use MOS ACM=1, ACM=2, or ACM=3 source and drain diode calculations to automatically generate parasitics.
3. General remarks:
 - Ideal current sources require large values of `.OPTION GRAMP`, especially for BJT and MESFET circuits. Such circuits do not ramp up with the supply voltages, and can force reverse-bias conditions, leading to excessive nodal voltages.

- Schmitt triggers are unpredictable for DC sweep analysis, and sometimes for operating points for the same reasons that oscillators and flip-flops are unpredictable. Use slow transient.
 - Large circuits tend to have more convergence problems because they have a higher probability of uncovering a modeling problem.
 - Circuits that converge individually, but fail when combined, usually 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, by using a voltage-variable resistor, or a resistor with a large AC value (for AC analysis).
4. Check your options:
- Remove all convergence-related options, and try first with no special `.OPTION` settings.
 - Check non-convergence diagnostic tables for non-convergent nodes. Look up non-convergent nodes in the circuit schematic. They are usually latches, Schmitt triggers, or oscillating nodes.
 - For stubborn convergence failures, bypass DC all together, and use `.TRAN` with `UIC` set. Continue transient analysis until transients settle out, then specify the `.OP` time, to obtain an operating point during the transient analysis. To specify an AC analysis during the transient analysis, add an `.AC` statement to the `.OP` time statement.
 - `SCALE` and `SCALM` scaling options have a significant effect on parameter values in both elements and models. Be careful with units.

The following sections discuss these topics:

- [Shorted Element Nodes](#)
- [Inserting Conductance, Using DCSTEP](#)
- [Floating-Point Overflow](#)

Shorted Element Nodes

HSPICE disregards any capacitor, resistor, inductor, diode, BJT, or MOSFET if all of its leads connect together. The simulation ignores it in its component tally, and issues a warning:

```
**warning**  
all nodes of element x:<name> are connected together
```

Inserting Conductance, Using DCSTEP

In a DC operating-point analysis, failure to include conductances in a capacitor model results in broken circuit loops (because a DC analysis opens all capacitors). This might not be solvable. If you include a small conductance in the capacitor model, the circuit loops are complete, and HSPICE can solve them.

Modeling capacitors as complete opens generates this error:

```
No DC Path to Ground
```

For a DC analysis, use `.OPTION DCSTEP`, to assign a conductance value to all capacitors in the circuit. DCSTEP calculates the value as:

```
conductance=capacitance/DCSTEP
```

In [Figure 46 on page 488](#), HSPICE inserts conductance (G), in parallel with capacitance (C_g). This provides current paths around capacitances, in DC analysis.

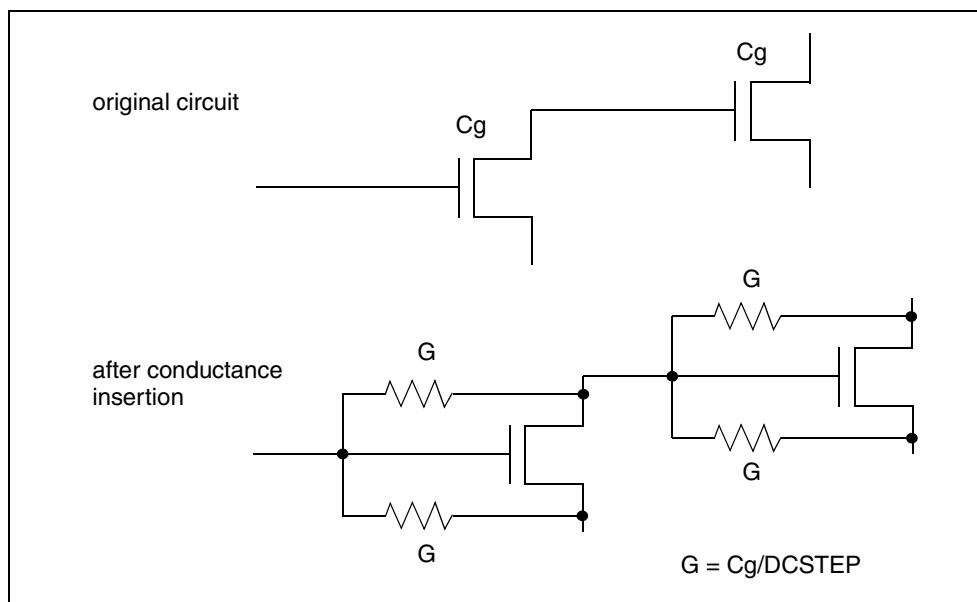


Figure 46 Conductance Insertion

Floating-Point Overflow

If MOS conductance is negative or zero, HSPICE might have difficulty converging. An indication of this type of problem is a floating-point overflow, during matrix solutions. HSPICE detects floating-point overflow, and invokes the Damped Pseudo Transient algorithm (`CONVERGE=1`), to try to achieve DC convergence without requiring you to intervene. If `GMINDC` is $1.0\text{e-}12$ or less when a floating-point overflows, HSPICE sets it to $1.0\text{e-}11$.

Diagnosing Convergence Problems

Before simulation, HSPICE diagnoses potential convergence problems in the input circuit, and provides an early warning, to help you in debugging your circuit. If HSPICE detects a circuit condition that might cause convergence problems, it prints the following message into the output file:

```
Warning: Zero diagonal value detected at node ( ) in equation
        solver, which might cause convergence problems. If your
        simulation fails, try adding a large resistor between
        node ( ) and ground.
```

The following sections discuss these topics:

- [Non-Convergence Diagnostic Table](#)
 - [Traceback of Non-Convergence Source](#)
 - [Solutions for Non-Convergent Circuits](#)
-

Non-Convergence Diagnostic Table

If a circuit cannot converge, HSPICE automatically generates two printouts, called the diagnostic tables:

- Nodal voltage printout: Prints the names of all no-convergent node voltages, and the associated voltage error tolerances (tol).
- Element printout: Lists all non-convergent elements, and their associated element currents, element voltages, model parameters, and current error tolerances (tol).

Chapter 13: Initializing DC-Operating Point Analysis

Diagnosing Convergence Problems

The element-printout diagnostic tables associate the error tolerances: `tolds`, `tolbd`, and `tolbs`. These indicate how close the element currents (drain to source, bulk to drain, and bulk to source) are to a convergent solution.

For the `tolxx` variables, a value close to or below 1.0 is a convergent solution.

The equation for `tol` is:

$$tol = \frac{abs(In - (In - In))}{(RELMOS \cdot \max(abs(In), (abs(In - 1) + ABSMOS)))}$$

where `RELMOS`, `ABSMOS` are HSPICE control options.

`In` --> current value at `n`th iteration

`In-1` --> current value at (`n-1`)th iteration

This equation calculates the values for `tolds`, `tolbs`, and `tolbd` by substituting corresponding currents values to the equation.

To locate the branch current or nodal voltage that causes non-convergence, use the following steps:

1. Analyze the diagnostic tables. Look for unusually large values of branch currents, nodal voltages or tolerances.
2. After you locate the cause, use the `.NODESET` or `.IC` statements, to initialize the node or branch.

If circuit simulation does not converge, HSPICE automatically generates a non-convergence diagnostic table, indicating:

- The quantity of recorded voltage failures.
- The quantity of recorded branch element failures.

Any node in a circuit can create voltage failures, including *hidden* nodes (such as extra nodes that parasitic resistors create).

3. Check the element printout for the subcircuit, model, and element name for all parts of the circuit where node voltages or currents do not converge.

For example, [Table 52 on page 491](#) identifies the `xinv21`, `xinv22`, `xinv23`, and `xinv24` inverters, as problem subcircuits in a ring oscillator. It also indicates that the p-channel transistors, in the `xinv21`, `xinv22`, `xinv24` subcircuits, are nonconvergent elements. The n-channel transistor of `xinv23` is also a nonconvergent element.

The table lists voltages and currents for the transistors, so you can check whether they have reasonable values. The `tolds`, `tolbd`, and `tolbs` error tolerances indicate how close the element currents (drain to source, bulk to

drain, and bulk to source) are, to a convergent solution. For *tol* variables, a value close to or below 1.0 is a convergent solution. In [Table 52](#), the *tol* values that are around 100, indicate that the currents were far from convergence. The table displays the element current and voltage values (*id*, *ibs*, *ibd*, *vgs*, *vds*, and *vbs*). Examine whether these values are realistic, and determine the transistor regions of operation.

Table 52 Subcircuit Voltage, Current, and Tolerance

subckt element model	xinv21 21:mphc1 0:p1	xinv22 22:mphc1 0:p1	xinv23 23:mphc1 0:p1	xinv23 23:mnch1 0:n1	xinv24 24:mphc1 0:p1
<i>id</i>	27.5809f	140.5646u	1.8123p	1.7017m	5.5132u
<i>ibs</i>	205.9804f	3.1881f	31.2989f	0.	200.0000f
<i>ibd</i>	0.	0.	0.	-168.7011f	0.
<i>vgs</i>	4.9994	-4.9992	69.9223	4.9998	-67.8955
<i>vds</i>	4.9994	206.6633u	69.9225	-64.9225	2.0269
<i>vbs</i>	4.9994	206.6633u	69.9225	0.	2.0269
<i>vth</i>	-653.8030m	-745.5860m	-732.8632m	549.4114m	-656.5097m
<i>tolds</i>	114.8609	82.5624	155.9508	104.5004	5.3653
<i>tolbd</i>	0.	0.	0.	0.	0.
<i>tolbs</i>	3.534e-19	107.1528m	0.	0.	0.

Traceback of Non-Convergence Source

To locate a non-convergence source, trace the circuit path for error tolerance. For example, in an inverter chain, the last inverter can have a very high error tolerance. If this is the case, examine the error tolerance of the elements that drive the inverter. If the driving tolerance is high, the driving element could be the source of non-convergence. However, if the tolerance is low, check the driven element as the source of non-convergence.

Examine the voltages and current levels of a non-convergent MOSFET to discover the operating region of the MOSFET. This information can flow to the

location of the discontinuity in the model—for example, subthreshold-to-linear, or linear-to-saturation.

When considering error tolerances, check the current and nodal voltage values. If you set these values extremely low, you divide a relatively large number by a very small number. This produces a large calculation result, which can cause the non-convergence errors. To solve this, increase the value of the absolute-accuracy options.

Use the diagnostic table, with the DC iteration limit (`ITL1` option), to find the sources of non-convergence. When you increase or decrease `ITL1`, HSPICE prints output for the problem nodes and elements for a new iteration—that is, the last iteration of the analysis that you set in `ITL1`.

Solutions for Non-Convergent Circuits

Non-convergent circuits generally result from:

- [Poor Initial Conditions](#)
- [Inappropriate Model Parameters](#)
- [PN Junctions \(Diodes, MOSFETs, BJTs\)](#)
- [Troubleshooting DC Bias Point and DC Sweep Non-Convergence](#)
- [Convergence Failure: Too Many Current Probes in Netlist](#)
- [Troubleshooting: Nodes set to initial conditions with `.IC` may not always begin at those voltage values](#)

Poor Initial Conditions

Multi-stable circuits need state information, to guide the DC solution. You must initialize ring oscillators and flip-flops. These multi-stable circuits can either produce an intermediate forbidden state, or cause a DC convergence problem. To initialize a circuit, use the `.IC` statement, which forces a node to the requested voltage. Ring oscillators usually require you to set only one stage.

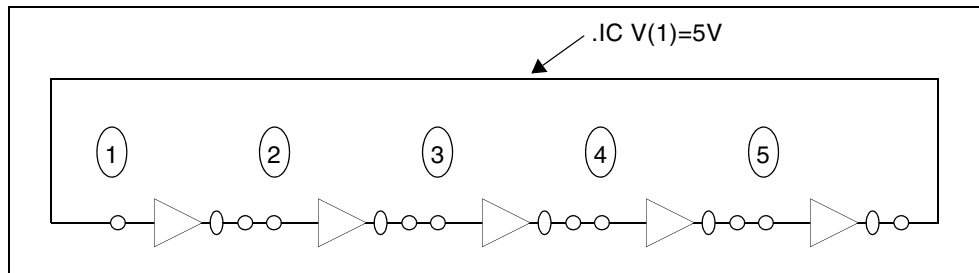


Figure 47 Ring Oscillator

The best way to set up the flip-flop is to use an `.IC` statement in the subcircuit definition.

Example

The following example sets the local `Qset` parameter to 0, and uses this value for the `.IC` statement, to initialize the Q latch output node. As a result, all latches have a default state of Q low. Set `Qset` to `vdd` to call a latch, which overrides this state.

```
.subckt latch in Q Q/ d Qset=0
.ic Q=Qset
...
.ends
Xff data_in[1] out[1] out[1]/ strobe LATCH Qset=vdd
```

Inappropriate Model Parameters

If you impose non-physical model parameters, you might create a discontinuous IDS or capacitance model. This can cause an *internal timestep too small* error, during the transient simulation. The `mosivcv.sp` demonstration file shows IDS, VGS, GM, GDS, GMB, and CV plots for MOS devices. A sweep near threshold from $V_{th}-0.5\text{ V}$ to $V_{th}+0.5\text{ V}$ (using a delta of 0.01 V), sometimes discloses a possible discontinuity in the curves.

Chapter 13: Initializing DC-Operating Point Analysis

Diagnosing Convergence Problems

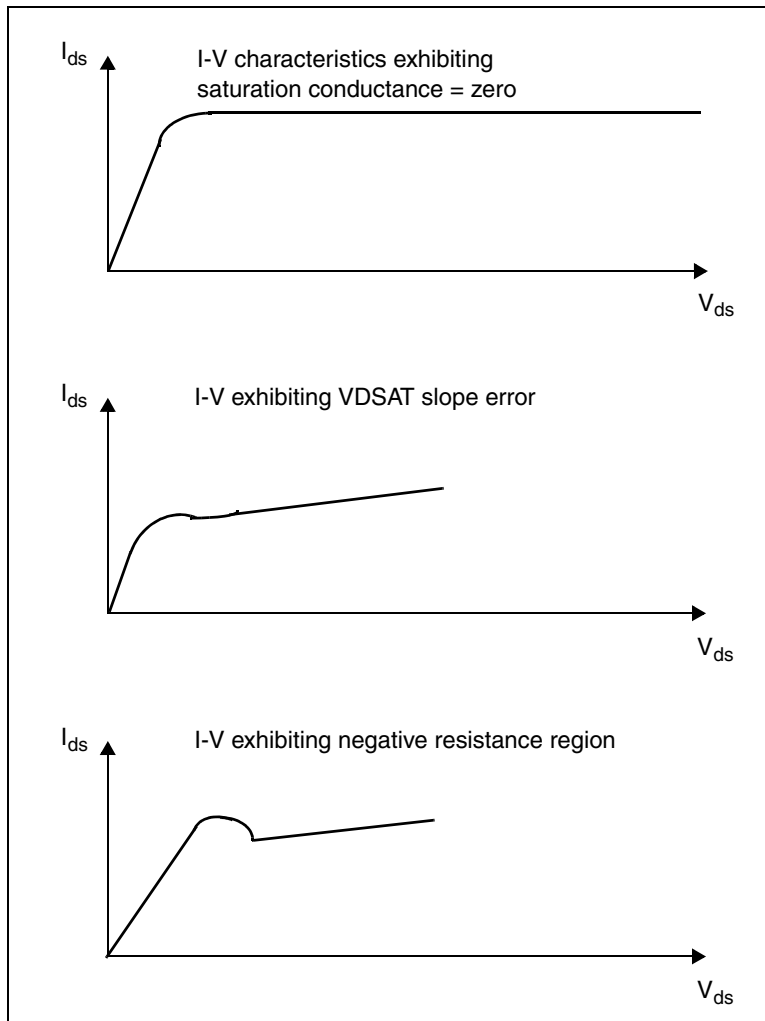


Figure 48 Discontinuous I-V Characteristics

If simulation does not converge when you add a component or change a component value, then the model parameters are not appropriate or do not correspond to physical values they represent.

To locate the problem, follow these steps:

1. Check the input netlist file for non-convergent elements.

Devices with a *TOL* value greater than 1, are non-convergent.

2. Find the devices at the beginning of the combined-logic string of gates that seem to start the non-convergent string.
3. Check the operating point of these devices very closely, to see what region they operate in.

Model parameters associated with this region are probably inappropriate.

Circuit simulation uses single-transistor characterization, to simulate a large collection of devices. If a circuit fails to converge, the cause can be a single transistor, anywhere in the circuit.

PN Junctions (Diodes, MOSFETs, BJTs)

PN junctions found in diode, BJT, and MOSFET models, might exhibit non-convergent behavior, in both DC and transient analysis.

Example

PN junctions often have a high *off* resistance, and result in an ill-conditioned matrix. To overcome this, use `.OPTION GMINDC` and `.OPTION GMIN` to automatically parallel every PN junction in a design, with a conductance.

Non-convergence can occur if you overdrive the PN junction. This happens if you omit a current-limiting resistor, or if the resistor has a very small value. In transient analysis, protection diodes are often temporarily forward-biased (due to the inductive switching effect). This overdrives the diode, and can result in non-convergence, if you omit a current-limiting resistor.

Troubleshooting DC Bias Point and DC Sweep Non-Convergence

The following procedures trade runtime performance and loosen certain tolerance bounds to overcome DC non-convergence. HSPICE steps from one DC convergence algorithm to another other to find a solution. You can assist this process as follows (in the same order).

1. Remove or comment out all simulation control options from your HSPICE testbench/netlists to allow the default auto-convergence procedure to work.
2. For circuits with feedback or multiple bias states (FF and latches), it is important to provide HSPICE with an initial guess that is close to the final solution. Use the `.NODESET` command to set initial voltage guesses. In particular, focus on those nodes the output `*.lis` file lists as nonconvergent.

```
.nodeset v(in)=0 v(out)=3.3
```

Chapter 13: Initializing DC-Operating Point Analysis

Diagnosing Convergence Problems

3. Use the symbolic (.OPTION SYMB) operating point algorithm which finds initial guesses before calculating the operating point.

```
.option SYMB=1
```

Explanation: When you use the SYMB option, HSPICE assumes the circuit is digital and assigns a low/high state to all nodes as a reasonable initial voltage guess. This option improves DC convergence for oscillators, logic, and mixed-signal circuits.

4. Increase ITL1 from default value of 200 up to 500 in steps of 100. To further help DC sweep analysis, you may increase ITL2 in the same manor which increases the number of iterations HSPICE takes at each DC sweep point.

```
.option ITL1=300 ITL2=300
```

Explanation: If increasing ITL2 does not help DC sweep analysis, the problem likely lies in model discontinuities. In other words, if you set ITL2=400 and do not solve the convergence problem, it is unlikely that any further increase of the value of ITL2 can help convergence. As a workaround, try to increase and offset the sweep size in an attempt to miss model problems.

```
Original: .dc vin 0v 3.3v .1v  
Increase: .dc vin 0v 3.3v .2v  
Offset: .dc vin .01v 3.31v .1v
```

5. HSPICE tries various convergence algorithms to achieve DC convergence. Read the .lis file to see where HSPICE was when the job aborted. HSPICE first tries DCON=1,2, then converge=1. If this is not enough, try the other two converge choices along with larger gmindc values (CONVERGE=3 is the source stepping method listed in “Inside SPICE”). However, do not set gmindc larger than 1e-9.

```
.option converge=2 gmindc=1e-11  
.option converge=3 gmindc=1e-11
```

6. If certain active element nodes seem to be non-convergent, you may have HSPICE perform 2 DC bias point calculations. HSPICE performs the first calculation with the active elements turned off. Then, this solution becomes the first guess for the DC solution with the elements turned on. You may choose one or more elements to turn off, declared on the element line.

```
Diode n1 n2 diode_model off  
Qbjt n1 n2 n3 bjt_model off  
Mosfet n1 n2 n3 n4 mos_model off
```

Convergence Failure: Too Many Current Probes in Netlist

HSPICE accomplishes probing current by the insertion of a zero-volt source. When you explicitly add large numbers of current probes or by use of wildcard syntax such as `.probe i(*)`, the size of the solution matrix increases significantly which can lead to convergence failures. These failures generate the message: `**diagnostic** rebuilding matrix with pivot option for special current probe process`

An error message follows: `**error** no convergence in operating point.`

Workarounds:

- Reduce the number of current probes by only probing specific nodes of interest, or adding a qualification to the wildcard.
- Create a saved operating point and tell HSPICE to use those initial conditions in the transient analysis.

The basic steps are:

- Run HSPICE without all the current probes, but include a `.OP` statement to create an initial conditions (`.ic0`) file.
- Include that file in your netlist. Example:
`.include my_design.ic0`
- Add “uic” for Use Initial Conditions on the `.TRAN` line. Example:
`.tran 1n 100n uic`

Then, it is possible that the design runs to completion even with the large number of current probes.

For more information on non-convergence, refer to [Autoconverge Process](#) and [Reducing DC Errors](#) in this chapter.

Troubleshooting: Nodes set to initial conditions with `.IC` may not always begin at those voltage values

The value set by `.IC` is not a voltage source, but a voltage source equivalent in the form of a current source with a parallel conductance. By default, that conductance is 100 mho (siemens) for an effective resistance of .01 ohms. The `GMAX` parameter sets that default conductance.

For example, if a Norton equivalent circuit created by that source is comparable with the conductance of other parts of the circuit, the DC node voltages deviate from those specified in `.IC` statement. Adjusting the `GMAX` parameter can

Chapter 13: Initializing DC-Operating Point Analysis

Diagnosing Convergence Problems

reduce this effect. For instance, setting GMAX to 1e6 creates a very low internal resistance. Here is a simple case that illustrates the effect of changing GMAX from its default value.

```
*** initial condition test ***
.option ingold=1
v01 n1 0 1v
r01 n1 n2 1m
r02 n2 n3 1m
r03 n3 0 1m
*.option gmax=1e6
.ic v(n2)=.25v v(n3)=.25v
.print tran v(n2) v(n3)
.tran 1n 5n
.end
*****
```

Results with GMAX defaulting to 100 siemens...

time	voltage n2	voltage n3
0.	0.6386	0.3160
1.0000e-09	0.6667	0.3333
2.0000e-09	0.6667	0.3333
3.0000e-09	0.6667	0.3333
4.0000e-09	0.6667	0.3333
5.0000e-09	0.6667	0.3333

Results with GMAX set to 1e6 siemens...

time	voltage n2	voltage n3
0.	0.2507	0.2498
1.0000e-09	0.6667	0.3333
2.0000e-09	0.6667	0.3333
3.0000e-09	0.6667	0.3333
4.0000e-09	0.6667	0.3333
5.0000e-09	0.6667	0.3333

Note that the initial conditions are much closer to the desired values.

AC Small-Signal and Noise Analysis

Describes how to perform AC and noise small signal analyses in HSPICE.

This chapter covers AC small signal analysis, AC analysis of an RC network, noise analysis, and other AC analysis statements. For information on output variables, see [AC Analysis Output Variables](#) in the *HSPICE User Guide: Basic Simulation and Analysis*.

HSPICE ships numerous examples for your use; see [Applications of General Interest Examples](#) for paths to demo files.

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

For discussion of use of the `.AC` command in subcircuit blocks, see [Using Isomorphic Analyses in Subckt Blocks on page 36](#) in this manual.

For discussion of [Transient Noise Analysis](#) and [Simulation of Random Noise](#), see the *HSPICE User Guide: Advanced Analog Simulation and Analysis*.

These topics are covered in the following sections:

- [Using the .AC Statement](#)
- [AC Small Signal Analysis](#)
- [AC Analysis of an RC Network](#)
- [Using .NOISE for Small-Signal Noise Analysis](#)
- [Using .AC/.NOISE Analyses with .TRAN](#)
- [Measuring Total Noise](#)
- [Other AC Analysis Statements - .LSTB, .DISTO, and .SAMPLE](#)

Using the .AC Statement

You can use the .AC statement for the following applications:

- Single/double sweeps
- Sweeps using parameters
- .AC analysis optimization
- Random/Monte Carlo analyses

For .AC command syntax, see the .AC command in the *HSPICE Reference Manual: Commands and Control Options*.

.AC Control Options

You can use the following .AC control options when performing an AC analysis:

ABSH	RELH	UNWRAP
MAXAMP	DI	

For syntax descriptions for these options, see Chapter 3, [HSPICE Netlist Simulation Control Options](#) in the *HSPICE Reference Manual: Commands and Control Options*.

.AC Command Examples

Example 1

```
.AC DEC 10 1K 100MEG
```

This example performs a frequency sweep by 10 points per decade from 1kHz to 100MHz.

Example 2

```
.AC LIN 100 1 100HZ
```

This example runs a 100-point frequency sweep from 1- to 100-Hz.

Example 3

```
.AC DEC 10 1 10K SWEEP cload LIN 20 1pf 10pf
```

This example performs an AC analysis for each value of `cload`. This results from a linear sweep of `cload` between 1- and 10-pF (20 points), sweeping the frequency by 10 points per decade from 1- to 10-kHz.

Example 4

```
.AC DEC 10 1 10K SWEEP rx POI 2 5k 15k
```

This example performs an AC analysis for each value of `rx`, 5k and 15k, sweeping the frequency by 10 points per decade from 1- to 10-kHz.

Example 5

```
.AC DEC 10 1 10K SWEEP DATA=datanm
```

This example uses the `.DATA` statement to perform a series of AC analyses, modifying more than one parameter. The `datanm` file contains the parameters.

Example 6

```
.AC DEC 10 1 10K SWEEP MONTE=30
```

This example illustrates a frequency sweep and a Monte Carlo analysis with 30 trials.

Example 7

```
AC DEC 10 1 10K SWEEP MONTE=10 firstrun=15
```

This example illustrates a frequency sweep and a Monte Carlo analysis from the 15th to the 24th trials.

Example 8

```
.AC DEC 10 1 10K SWEEP MONTE=list(10 20:30 35:40 50)
```

This example illustrates a frequency sweep and a Monte Carlo analysis at 10th trial and then from the 20th to 30th trial, followed by the 35th to 40th trial and finally at 50th trial.

AC Small Signal Analysis

AC small signal analysis in HSPICE computes AC output variables as a function of frequency (see [Figure 49 on page 502](#)). HSPICE first solves for the DC operating point conditions. It then uses these conditions to develop linear, small-signal models for all non-linear devices in the circuit.

Chapter 14: AC Small-Signal and Noise Analysis

AC Small Signal Analysis

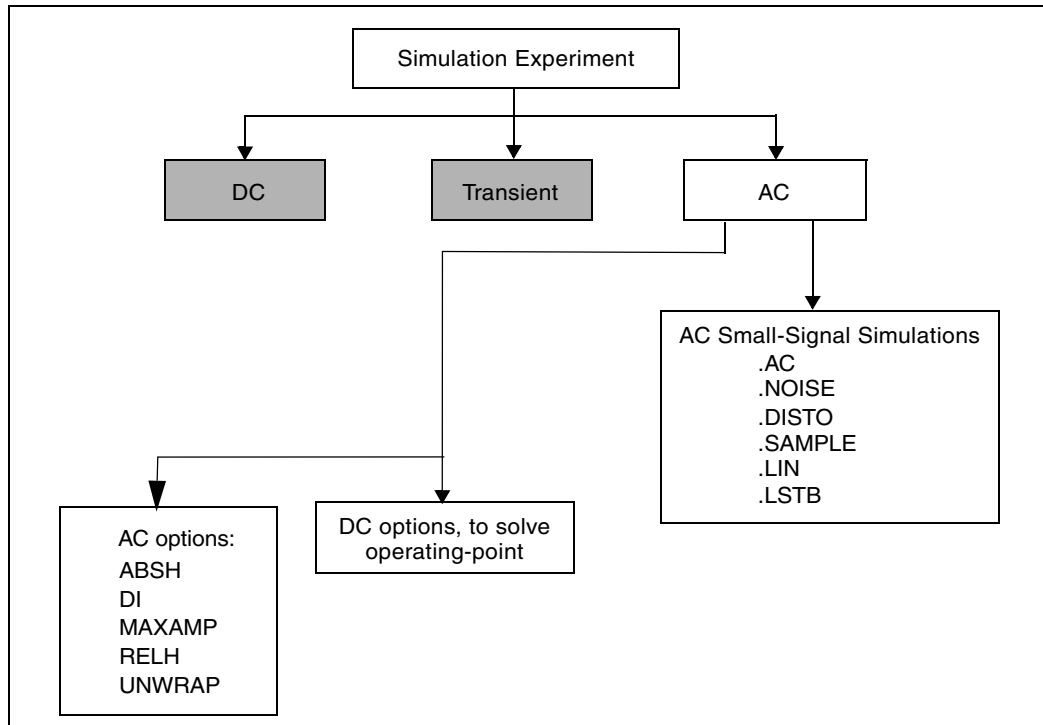


Figure 49 AC Small Signal Analysis Flow

In HSPICE, the output of AC Analysis includes voltages and currents.

HSPICE converts capacitor and inductor values to their corresponding admittances:

$$y_C = j\omega C \quad \text{for capacitors}$$

$$y_L = \frac{1}{j\omega L} \quad \text{for inductors}$$

Resistors can have different DC and AC values. If you specify `AC=<value>` in a resistor statement, HSPICE uses the DC value of resistance to calculate the operating point, but uses the AC resistance value in the AC analysis. When you analyze operational amplifiers, HSPICE uses a low value for the feedback resistance to compute the operating point for the unity gain configuration. You can then use a very large value for the AC resistance in AC analysis of the open loop configuration.

AC analysis of bipolar transistors is based on the small-signal equivalent circuit, as described in the *HSPICE Elements and Device Models Manual*. MOSFET

AC-equivalent circuit models are described in the *HSPICE Elements and Device Models Manual*.

The AC analysis statement can sweep values for:

- Frequency.
- Element.
- Temperature.
- Model parameter.
- Randomized (Monte Carlo) distribution.
- Optimization and AC analysis.

Additionally, as part of the small-signal analysis tools, HSPICE provides:

- Noise analysis.
- Distortion analysis.
- Network analysis.
- Sampling noise.

You can use the `.AC` statement in several different formats, depending on the application. You can also use the `.AC` statement to perform data-driven analysis in HSPICE.

AC Analysis of an RC Network

[Figure 50 on page 504](#) shows a simple RC network with a DC and AC source applied. The circuit consists of:

- Two resistors, R1 and R2.
- Capacitor C1.
- Voltage source V1.
- Node 1 is the connection between the source positive terminal and R1.
- Node 2 is where R1, R2, and C1 are connected.
- HSPICE ground is always node 0.

Chapter 14: AC Small-Signal and Noise Analysis

AC Analysis of an RC Network

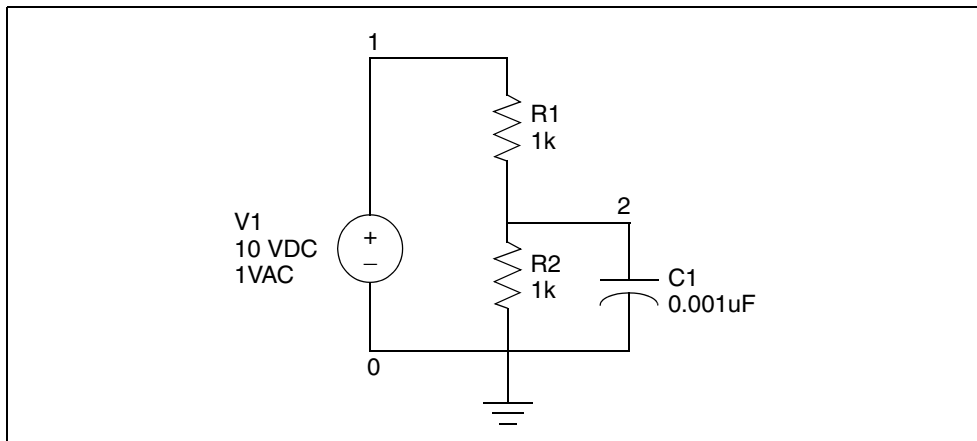


Figure 50 RC Network Circuit

The netlist for this RC network is based on demonstration netlist `quickAC.sp`, which is available in directory `$installdir/demo/hspice/apps`:

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

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

1. Type the above netlist into a file named `quickAC.sp`.
2. To run a HSPICE analysis, type:

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

When the run finishes, HSPICE displays:

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

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

Your run directory includes the following new files:

- `quickAC.ac0`

- quickAC.ic0
 - quickAC.lis
 - quickAC.st0
3. Use an editor to view the .lis and .st0 files to examine the simulation results and status.
 4. Run WaveView.
 5. From the **File** menu, select **File > Import > Waveform File**.
 6. Select the quickAC.ac0 file from the Open: Waveform Files window.
 7. Display the voltage at node 2 by using a log scale on the x-axis.

Figure 51 on page 505 shows the waveform that HSPICE produces if you sweep the response of node 2, as you vary the frequency of the input from 1 kHz to 1 MHz.

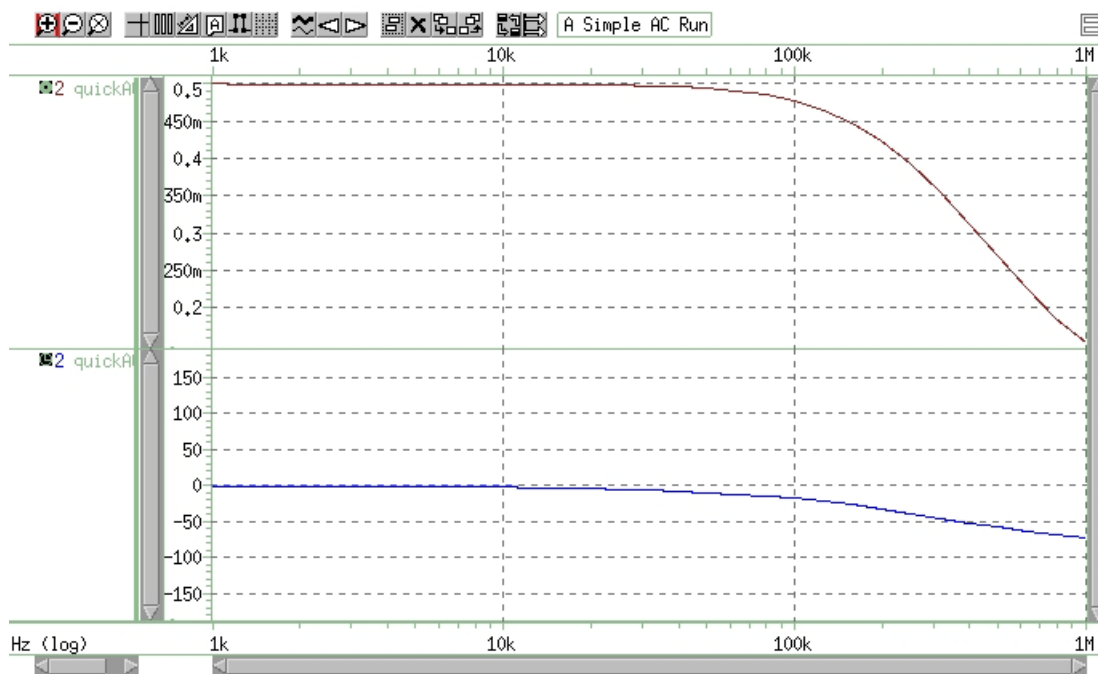


Figure 51 RC Network Node 2 Frequency Response

As you sweep the input from 1 kHz to 1 MHz, the quickAC.lis file displays:

- Input netlist.
- Details about the elements and topology.

Chapter 14: AC Small-Signal and Noise Analysis

Using .NOISE for Small-Signal Noise Analysis

- Operating point information.
- Table of requested data.

The `quickAC.ic0` file contains information about DC operating point conditions. The `quickAC.st0` file contains information about the simulation run status.

To use the operating point conditions for subsequent simulation runs, execute the `.LOAD` statement.

Using .NOISE for Small-Signal Noise Analysis

A circuit noise analysis can be performed associated with a small-signal `.AC` analysis. The `.NOISE` command will activate a noise analysis that calculates the output noise generated based on the contributions from all noise sources within the circuit. This noise may be from passive elements, such as thermal (Johnson) noise in resistors, or from sources such as shot, channel, and flicker noise present within transistors. Most transistors will have several noise sources. For descriptions of noise models for each device type, see the [HSPICE Reference Manual: Elements and Device Models](#). In most cases, the individual noise sources in HSPICE lack statistical correlation, and this allows their contributions to output noise to be computed independently. The total output noise voltage is the RMS sum of the individual noise contributions:

Equation 45

$$onoise = \sqrt{\sum_{k=0}^N |Z_k|^2 \overline{|i_{nk}|^2}}$$

Where,

onoise is the output noise spectral density (V/\sqrt{Hz}) at the AC analysis frequency.

$\overline{|i_{nk}|^2}$ is the mean-squared noise spectral density (A^2/Hz) for each noise current source due to thermal, shot, flicker, or other noise.

Z_k is the equivalent transimpedance between each noise current source and the output.

N is the number of noise sources associated with all circuit elements.

This analysis will be performed for every frequency specified with the `.AC` command. The output for noise analysis is specified in the `.NOISE` syntax:

Basic Syntax:

```
.NOISE v(out <,ref>) src <interval>
```

The output noise (`onoise`) voltage is computed at the out node specified; if the (optional) `ref` node is also given, the output is taken as the differential output noise voltage `v(out,ref)`. Noise analysis requires the specification of an independent input source (`src`). This allows the calculation of the equivalent input noise given by

Equation 46

$$inoise = \frac{onoise}{|G|}$$

Where,

inoise is the equivalent input noise spectral density at the input source. G is the gain between the input source (`src`) and the output.

The `.NOISE` analysis can also generate a summary for how each noise generator within the circuit will contribute to output noise. Specify an integer value for `interval` to include a device noise summary for every `interval` frequency points in the HSPICE output listing. No summary is included unless `interval` is specified. The `.NOISE` analysis will also compute the total integrated noise over the AC frequency range, which will also be included in the output listing. The output summary will include values for the device noise sources given in [Table 53 on page 508](#) — [Table 55 on page 509](#).

To request `.NOISE` analysis results (magnitude and decibel) with `.print/.probe use`:

```
.probe noise onoise onoise(m) onoise(db)  
.probe noise inoise inoise(m) inoise(db)
```

Results will be included in the `*.ac0` file. Output noise voltage or current units are either V/\sqrt{Hz} or A/\sqrt{Hz} , respectively. Device-level noise source contributions will also be included in the `*.ac0` file (unless you have set `.option probe=1`). The naming convention and units for device level noise contributions is also shown in the following tables.

To ensure that device noise models will be included in the analysis, verify that noise parameters are being set in your transistor models. Include values for AF

Chapter 14: AC Small-Signal and Noise Analysis

Using .NOISE for Small-Signal Noise Analysis

and KF, for example, if you wish to activate flicker noise models for your devices.

See also, [Using Noise Analysis Results as Input Noise Sources](#) in the *HSPICE User Guide: RF Analysis*.

For a complete description of the .NOISE command syntax and examples, see the .NOISE command in the *HSPICE Reference Manual: Commands and Control Options*. Note that the .NOISE analysis requires an .AC statement, and that if more than one .NOISE statement is included, HSPICE will run only the last statement.

Table 53 .NOISE Measurements Available for MOSFETs

.ac	.lis	Unit	Description
nd	rd	$\frac{\sqrt{V^2}}{\text{Hz}}$	Output thermal noise due to drain resistor
ns	rs	$\frac{\sqrt{V^2}}{\text{Hz}}$	Output thermal noise due to source resistor
ni	id	$\frac{\sqrt{V^2}}{\text{Hz}}$	Output channel thermal noise
nf	fn	$\frac{\sqrt{V^2}}{\text{Hz}}$	Output flicker noise
ntg	total	$\frac{\sqrt{V^2}}{\text{Hz}}$	Total output noise: TOT=RD + RS + ID + FN

Table 54 .NOISE Measurements Available for BJTs

.ac	.lis	Unit	Description
rb	rb	$\frac{\sqrt{V^2}}{\text{Hz}}$	Output thermal noise due to base resistor
rc	rc	$\frac{\sqrt{V^2}}{\text{Hz}}$	Output thermal noise due to collector resistor

Table 54 .NOISE Measurements Available for BJTs (Continued)

.ac	.lis	Unit	Description
re	re	$\frac{\sqrt{2}}{\text{Hz}}$	Output thermal noise due to emitter resistor
nb	ib	$\frac{\sqrt{2}}{\text{Hz}}$	Output noise due to base shot noise source
nc	ic	$\frac{\sqrt{2}}{\text{Hz}}$	Output thermal noise due to collector shot noise source
nf	fn	$\frac{\sqrt{2}}{\text{Hz}}$	Output noise due to flicker noise source
nt	total	$\frac{\sqrt{2}}{\text{Hz}}$	Total output noise: TOT=RB + RC + RE + IB + IC + FN

Table 55 .NOISE Measurements Available for Diodes

.ac	.lis	Unit	Description
nr	rs	$\frac{\sqrt{2}}{\text{Hz}}$	Output noise due to diode series resistance
ni	id	$\frac{\sqrt{2}}{\text{Hz}}$	Output noise due to shot noise
nf	fn	$\frac{\sqrt{2}}{\text{Hz}}$	Output noise due to flicker noise
nt	total	$\frac{\sqrt{2}}{\text{Hz}}$	Total output noise: TOT=RS + ID + FN

Using .AC/.NOISE Analyses with .TRAN

In some situations, a .TRAN analysis may be needed to establish the operating point used for .AC or .NOISE analysis.

To do this, use the combination of commands as shown in the example below:

```
.TRAN 1n 5u $ Transient analysis
.OP 1u 2 u 3u $ Request operating point analysis
.AC DEC 100 1 20e9 $ AC analysis
.NOISE V(out) V1 $ NOISE analysis
```

HSPICE performs separate .AC analyses for all time values specified as well as one .AC run at time zero. This happens during the .TRAN analysis as it does the .OP and .AC evaluations, and results in separate *.AC0 files with unique labels for each time value specified. An *.ac0@tranop.grp file is also created when you run .AC/.NOISE with .TRAN analysis.

In addition, the .OP used at the time values specified is fully dynamic, meaning it uses all sources and nonlinearities involved at that time value during the .TRAN. The charges and currents of the .TRAN are used and preserved for the .AC. It does this by using the derivatives ($C=dQ/dv$, $G=dl/dv$) at that point in the .TRAN for computing the .AC small-signal analysis.

The .AC and .NOISE analysis is then performed at this operating point.

For additional information, see .OP analysis in the *HSPICE Reference Manual: Commands and Control Options*.

Measuring Total Noise

You can measure total noise using the .MEASURE command with ac. After performing a small signal noise analysis, HSPICE prints the total output noise voltage to the *.lis file. For example:

```
**** total output noise voltage = 51.8809m volts
```

You can get the total noise output as reported in the *.lis or the *.noise# file by using the following two .MEASURE commands:

```
.meas ac int_noise_sq integ par('onoise(m)*onoise(m)')
+ from='fstart' to='fstop'

.meas ac total_noise_v param='sqrt(int_noise_sq)'
```

Other AC Analysis Statements - .LSTB, .DISTO, and .SAMPLE

The following sections describe the commands you can use to perform other types of AC analyses:

- [Using .LSTB for Loop Stability Analysis](#)
- [Using .DISTO for Small-Signal Distortion Analysis](#)
- [Using .SAMPLE for Noise Folding Analysis on page 518](#)

Use the `.NOISE` and `.AC` statements to control the noise analysis of the circuit.

Using .LSTB for Loop Stability Analysis

Stability analysis can be applied on feedback circuits to analyze loop gain and phase characteristics in the frequency domain. Examples of circuits with feedback are amplifiers, bandgaps, oscillators, and voltage regulators. The `.LSTB` command aids in studying the analog circuit stability margin both for oscillators and the circuits which are not supposed to be oscillating. Stability analysis is performed without breaking the feedback loop of the circuit on AC analysis while maintaining the DC operating point and considering AC loading.

A 0V DC voltage source is required to place in series in the loop of interest. Two voltage sources are required for differential or common-mode loop analysis. The `.LSTB` command measures the loop gain by successive injection (Middlebrook Technique, see <http://authors.library.caltech.edu/4537/1/MIDieemm06.pdf>).

A zero voltage source is placed in series in the loop: one pin of the voltage loop must be connected to the loop input, the other pin to the loop output. The orientation of inserted voltage sources in differential/common-mode testing is significant. It is required that the positive terminal of both voltage sources go to the input of amplifier or go to the output of amplifier.

The `.LSTB` command can be used for these modes: single-ended, differential, and common. This analysis supports `.PRINT` / `.PROBE` / `.MEASURE` statements and can be used with `.ALTER` to generate multiple loop analyses. The functionality is available on all platforms.

For command syntax and additional examples, refer to [.LSTB](#) and [.MEASURE](#) [LSTB](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Note: When measuring loop gain for the circuit by using AC analysis, if you open the feedback loop and use AC analysis, you run the risk of changing the DC operating points on either side of the loop and changing the impedances on both sides of the loop from their closed loop impedance. This will give you errors in your results.

However, there is a two-step AC analysis technique that uses ideal voltage and current sources to measure the voltage and current gains. After you get the voltage and current gains, you need to post-process the results to get the final loop gain. This method works without introducing errors into the results.

Output Formats for Loop Stability (.LSTB) Analysis

The outputs for loop stability analysis are as follows:

- The gain margin (GM), phase margin (PM), unity gain frequency (FU) and gain at minimum frequency (ADC) are reported in the *.lis file.
- The Loop Gain is reported to the *.cx# file, which is always produced for .LSTB analysis. The *.cx# file is a general file for all the complex outputs. It contains the data for waveforms as complex vectors.
- If you specify .probe ac lstb(db) lstb(mag) lstb(real) lstb(imag) lstb(phase), the specific format of loop gain goes to the *.ac# file for viewing.
- If an *.ac# file is produced with .probe ac lstb, then both *.ac# and *.cx# file could be used to view magnitude, phase, real, and imaginary versus frequency as complex vectors.

Loop Stability Analysis Usage

```
.LSTB mode=[single|diff|comm]  
+ vsource=[vlstb|vlstbp,vlstbn]
```

Examples

Single-mode loop analysis on loop indicated by vx voltage source:

```
.LSTB mode=single vsource=vx
```

Differential-mode loop analysis on loops indicated by vp and vn voltage sources:

```
.LSTB mode=diff vsource=vp,vn
```

Common-mode loop analysis on loops indicated by v_p and v_n voltage sources:

```
.LSTB mode=comm vsource=vp,vn
```

Single-Ended Mode Example: Ideal Inverter

The following is an example available in the demo directory that ships with HSPICE showing use of a single-ended mode input netlist. (See `$installdir/demo/hspice/lstb/single.sp`.)

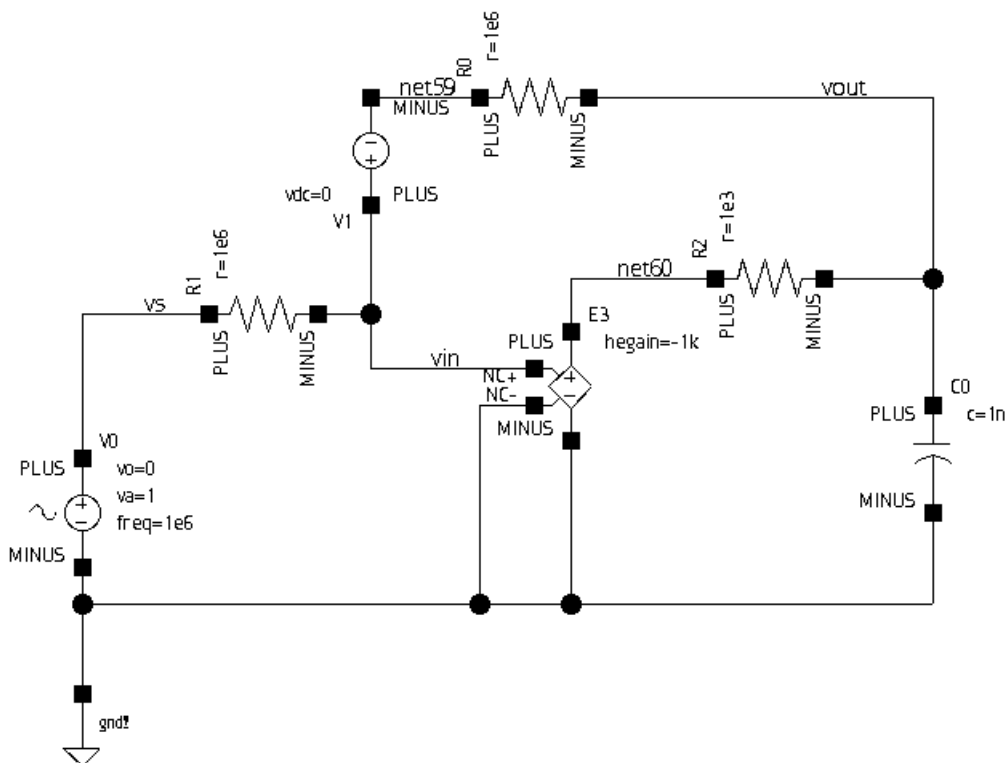


Figure 52 Ideal inverter amplifier with single pole

Chapter 14: AC Small-Signal and Noise Analysis

Other AC Analysis Statements - .LSTB, .DISTO, and .SAMPLE

This example netlist simulates single mode loop stability for a single-pole ideal inverter amplifier.

```
.GLOBAL gnd!

v1 vin net59 dc=0
v0 vs gnd! dc=0 ac=1 sin ( 0 1 1e6 0 0 0 )
e3 net60 gnd! vcvs vin gnd! -1000 max=1 min=-1 abs=0
r0 net59 vout r=1e6
r1 vs vin r=1e6
r2 net60 vout r=1e3
c0 vout gnd! c='1n'

.ac DEC '100' '100' '100e9'
.lstb mode=single vsource=v1
.option post
.probe ac lstb(m) lstb(db) lstb(p) lstb(r) lstb(i)
.end
```

The sequence of commands, controls, and parameters is as follows:

1. Insert a 0V voltage source in the feedback path
2. Specify an input source
3. Specify an ideal amplifier
4. Specify a feedback resistor
5. Specify a source resistor
6. Specify an output pole
7. Perform an .AC analysis (required)
8. Perform a single-mode .LSTB analysis
9. Add the .PROBE command for signals to be plotted

Differential Mode Example: Bandgap

The following is partial netlist showing the addition of two 0V DC sources for stability:

```
.subckt bandgap_low_voltage agnd avdd vbg
xi57 agnd net317 avdd v4_i v2_i vgate opamp_p
vlstbn v4_i v4 0      $0V DC source for stability loop analysis
vlstbp v2_i v2 0      $0V DC source for stability loop analysis
*
*
.ends bandgap_low_voltage
```


Chapter 14: AC Small-Signal and Noise Analysis
 Other AC Analysis Statements - .LSTB, .DISTO, and .SAMPLE

```
.AC DEC 100 100 10e9           $ AC analysis is required
.LSTB mode=diff vsource=xi4.vlstop,xi4.vlstbn $ LSTB analysis
.option post probe
.probe AC lstb(m) lstb(p) lstb(db) $ Probe mag, phase, dB
.measure LSTB pm phase_margin    $ Measure phase margin
.end
```

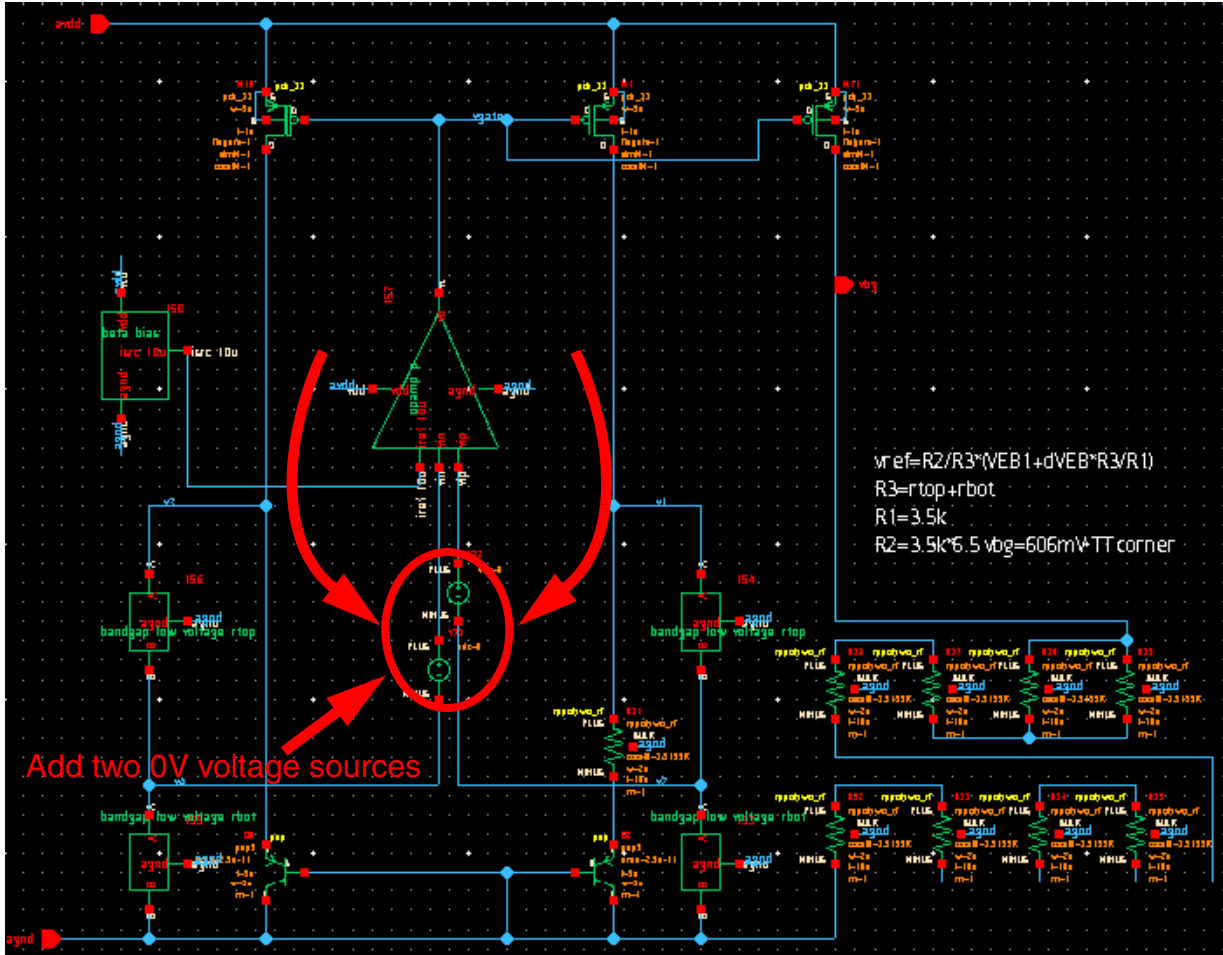


Figure 53 Adding two 0V voltage sources to bandgap subcircuit

Differential mode output can be viewed in both X-Y and Polar plots:



Figure 54 Custom WaveView plots enable you to access differential output

The *.lis output for this differential mode analysis is as follows:

```
*** LSTB analysis ***
gain_margin(dB) = 13.23320
phase_margin(deg) = 51.28824
unity_gain_freq(Hz) = 9.6913777E+07
loop_gain_mini_freq(dB) = 50.47896
```

Controlling LSTB Warning Messages

To avoid re-running simulations if there a syntax error in the .LSTB statement you can use .OPTION STRICT_CHECK. Use of this option is especially useful when running long AC and LSTB analyses on a batch machine and the AC analysis succeeds.

You can use `.OPTION STRICT_CHECK` to control the following messages:

1. The second v-source for diff/comm mode testing', ' is missing in LSTB statement.', ' Analysis is omitted.'
2. 'The v-source name in LSTB', ' statement is invalid. Analysis is omitted.
3. 'The mode type is missing in', ' LSTB statement. Analysis is omitted.'
4. 'The mode type in LSTB statement', ' is invalid. Analysis is omitted.'
5. 'Expected a keyword in LSTB statement.', ' Analysis is omitted.'
6. 'The v-sources are missing in', ' LSTB statement. Analysis is omitted.'
7. 'The v-source for single mode is', ' missing in LSTB statement.', ' Analysis is omitted.'
8. 'The first v-source for diff/comm mode testing', ' is missing in LSTB statement.', ' Analysis is omitted.'
9. 'The second v-source for diff/comm', ' mode in LSTB statement is missing.', ' Analysis is omitted.'
10. 'Found an invalid keyword "', "1a7," for LSTB analysis.'
11. 'The second v-source for diff/comm mode testing', ' is missing in LSTB statement.', ' Analysis is omitted.'

For details on usage, see [.OPTION STRICT_CHECK](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Using .DISTO for Small-Signal Distortion Analysis

The `.DISTO` statement computes the distortion characteristics of the circuit in an AC small-signal, sinusoidal, steady-state analysis. HSPICE computes and reports five distortion measures at the specified load resistor. The analysis is performed assuming that one or two signal frequencies are imposed at the input. The first frequency, F1 (used to calculate harmonic distortion), is the nominal analysis frequency set by the `.AC` statement frequency sweep. The optional second input frequency, F2 (used to calculate intermodulation distortion), is set implicitly by specifying the `skw2` parameter, which is the ratio $F2/F1$.

For command syntax and examples, see the [.DISTO](#) command in the *HSPICE Reference Manual: Commands and Control Options*.

Using .SAMPLE for Noise Folding Analysis

For data acquisition of analog signals, data sampling noise often needs to be analyzed. This is accomplished with the `.SAMPLE` statement used in conjunction with the `.NOISE` and `.AC` statements. The `SAMPLE` analysis performs a simple noise folding analysis at the output node.

For the syntax and description of the `.SAMPLE` statement, see the [.SAMPLE](#) command in the *HSPICE Reference Manual: Commands and Control Options*.

Transient Analysis

Describes how to use transient analysis to compute the circuit solution.

Transient analysis computes the circuit solution, as a function of time, over a time range specified in the `.TRAN` statement.

For descriptions of individual HSPICE commands referenced in this chapter, see the [HSPICE Reference Manual: Commands and Control Options](#).

For discussion of use of the `.TRAN` command in subcircuit blocks, see [Using Isomorphic Analyses in Subckt Blocks on page 36](#).

For full description of analyzing time-variant noise, see [Transient Noise Analysis](#) in the *HSPICE User Guide: Advanced Analog Simulation and Analysis*.

HSPICE ships hundreds of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

The following sections present these topics:

- [Simulation Flow](#)
- [Overview of Transient Analysis](#)
- [Transient Control Options](#)
- [Simulation Speed and Accuracy Using the RUNLVL Option](#)
- [Numerical Integration Algorithm Controls](#)
- [Dynamic Check Using the .BIASCHK Statement](#)
- [Storing and Restoring Checkpoint Files](#)
- [Troubleshooting: Internal Timestep, Measurement Errors](#)

Simulation Flow

Figure 55 on page 520 illustrates the simulation flow for transient analysis in HSPICE.

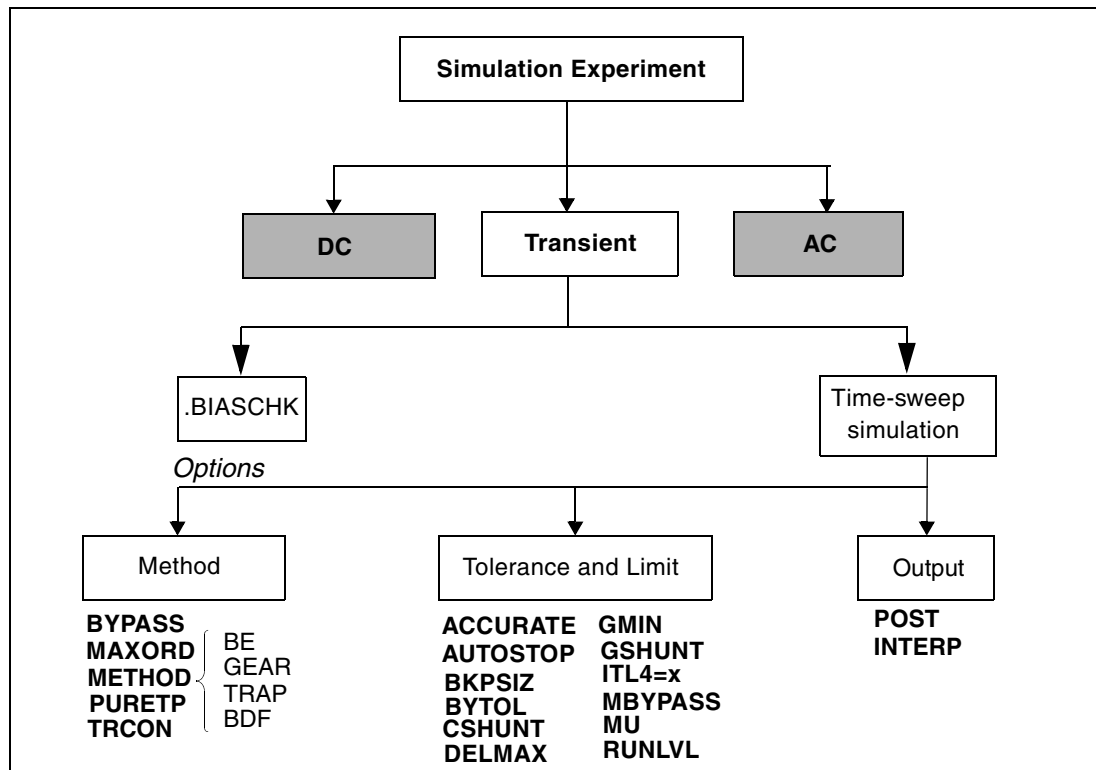


Figure 55 Transient Analysis Simulation Flow

Overview of Transient Analysis

Transient analysis simulates a circuit at a specific time. *Some* of its algorithms, control options, convergence-related issues, and initialization parameters are different than those used in DC analysis. However, a transient analysis first performs a DC operating point analysis, unless you specify the UIC option in the .TRAN statement.

Unless you set the initial circuit operating conditions, some circuits (such as oscillators, or circuits with feedback) do not have stable operating point solutions. For these circuits, either:

- Break the feedback loop, to calculate a stable DC operating point, or
- Specify the initial conditions in the simulation input.

For setting initial conditions, see [Initial Conditions and UIC Parameters](#).

Example

In the following example, the UIC parameter (in the `.TRAN` statement) bypasses the initial DC operating point analysis. The `.OP` statement calculates the transient operating point (at $t=20$ ns), during the transient analysis.

```
.TRAN 1ns 100ns UIC  
.OP 20ns
```

In a transient analysis, the internal timestep too small error message indicates that the circuit failed to converge. The cause of this convergence failure might be that stated initial conditions are not close enough to the actual DC operating point values. Use the commands in this chapter to help achieve convergence in a transient analysis. See also: [Troubleshooting: Internal Timestep, Measurement Errors](#) at the end of this chapter.

The following sections discuss these topics.

- [Data-Driven vs. Outer Parameter Sweeps](#)
- [Transient Analysis Output](#)
- [Transient Analysis of an RC Network](#)
- [Transient Analysis of an Inverter](#)

Data-Driven vs. Outer Parameter Sweeps

The following defines the differences between a data-driven sweep and an outer parameter sweep.

Data-Driven Sweep

The use of a data set allows the sweeping of both nonuniform values and multiple parameters. You need to specify each value to vary in the simulation. This method generates one output file for the entire simulation. When viewing signals, the traces correspond to each parameter sweep.

Chapter 15: Transient Analysis

Overview of Transient Analysis

Example: Data-Driven Sweep

```
.tran 1n 100n sweep data=mydata
.data mydata param1 param2 ...
val1 vala ...
val2 valb ...
....
.enddata
```

Parameter Sweep

When you express the values of a parameter by using decade, octave, linear, or point-of-interest variation, you can use the `sweep` keyword to control the parameter. This method does not allow for the sweeping of multiple parameters. Similar to the data-driven sweep, HSPICE creates only one output file with the signals having multiple traces. Be sure to sequence the `var` (param) before the `type` (DEC, LIN).

Examples

In this example, `param` varies 10 times for each decade from 1u to 10u and a transient analysis runs for each value.

```
.tran 1n 100n sweep param DEC 10 1u 10u
```

In this example, `param` varies 5 equal times from 1u to 10u with a transient analysis for each value.

```
.tran 1n 100n sweep param LIN 5 1u 10u
```

Sweeping Multiple Parameters

Although HSPICE does not directly provide the facility to sweep multiple parameters, it does offer the `.DATA` table structure. A perl script is available to allow you to specify lists of parameters and values at <https://solvnet.synopsys.com/retrieve/021478.html>

This script creates a `.DATA` table with all permutations of the listed values. It also allows you to create `.ALTERS` instead of a `.DATA` table, if preferred. For usage details, run `hspice_param_sweeper -h`.

The script's output goes to `STDOUT`, so redirect it to a file, for example: `hspice_param_sweeper > param_sweep.sp`), and then `.INCLUDE` the file into your HSPICE netlist. If you choose to create `.ALTERS`, make sure you `.INCLUDE` them at the very end of your netlist.

If you create a `.DATA` table, you can invoke it as follows:

```
.TRAN 10p 100n SWEEP DATA=sweeper_params
```


Here is a sample input to the script.

```
vddr: 1.1, 1.0, 0.9  
vssr: 0.0  
temp: 0, 55, 100
```

Note that `temp` is a special parameter that sweeps the simulation temperature. This example produces a `.DATA` table with 9 rows (3*1*3) containing all combinations of the listed parameter values, or 1 base sim + 8 `.ALTERS` if you use the `-alter` option.

Here is the output produced by the sample input.

```
.DATA sweeper_params temp vddr vssr  
  0 1.1 0.0  
 55 1.1 0.0  
100 1.1 0.0  
  0 1.0 0.0  
 55 1.0 0.0  
100 1.0 0.0  
  0 0.9 0.0  
 55 0.9 0.0  
100 0.9 0.0  
.ENDDATA
```

After you download the script named `hspice_param_sweeper.gz` (right-click and select “Save Target As...”), be sure to modify the first line of the script to point to your local installation of perl. The default path should work on most systems.

Specifying Data Driven Timesteps

Instead of using a constant timestep in a `.TRAN` statement, you can specify the timesteps using an inline data statement for the transient simulation.

The data defined in the `.DATA` statement should define the time point and current value for a PWL current source. In the following example, the `.DATA` statement `tstep_val` defines the time step, `step_val` and the current value, `ival`. HSPICE uses the timesteps defined in the `.DATA` statement during the transient simulation.

```
Ipwl nd1 0 PWL (step_val ival)  
.tran DATA = tstep_val  
.DATA tstep_val step_val ival  
+ 10p 1m  
+ 30p 10m  
+ 70P 10m  
+ 100p 100m  
.ENDDATA
```

The timestep value specified in the data table (`DATA=tstep_val`) controls the print intervals.

Transient Analysis Output

```
.print tran ov1 [ov2 ... ovN]
.probe tran ov1 [ov2 ... ovN]
.measure tran measspec
```

The `ov1, ... ovN` output variables can include the following:

- `V(n)`: voltage at node *n*.
- `V(n1<,n2>)`: voltage between the *n1* and *n2* nodes.
- `Vn(d1)`: voltage at *n*th terminal of the *d1* device.
- `In(d1)`: current into *n*th terminal of the *d1* device.
- `'expression'`: expression, for the plot variables above

You can use wildcards to specify multiple output variables in a single command. Output content depends on `.OPTION POST` or `.OPTION PROBE`.

Parameter	Description
<code>*.print</code>	Writes the output from the <code>.PRINT</code> statement to a <code>*.print</code> file. HSPICE does not generate a <code>*.print#</code> file. <ul style="list-style-type: none">▪ The header line contains column labels.▪ The first column is time.▪ The remaining columns represent the output variables specified with <code>.PRINT</code>.▪ Rows that follow the header contain the data values for simulated time points.
<code>*.tr#</code>	Writes output from the <code>.PROBE</code> , <code>.PRINT</code> , or <code>.MEASURE</code> statement to a <code>*.tr#</code> file.

Transient Analysis of an RC Network

Follow these steps to run a transient analysis of a RC network with a pulse source, a DC source, and an AC source:

1. Type the following netlist into a file named `quickTRAN.sp`.

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

This example uses demonstration netlist `quickTRAN.sp`, which is available in directory `$installdir/demo/hspice/apps`.

Note: The V1 source specification includes a pulse source. For the syntax of pulse sources and other types of sources, see [Chapter 10, Sources and Stimuli](#).

2. To run HSPICE, type the following:

```
hspice quickTRAN.sp > quickTRAN.lis
```
3. To examine the simulation results and status, use an editor and view the `.lis` and `.st0` files.
4. Run WaveView and open the `.sp` file.
5. From the File menu, select **File > Import Waveform > File**.
6. Select the `quickTRAN.tr0` file from the Open: Waveform Files window.
7. Display the voltage at nodes 1 and 2 on the x-axis.

[Figure 56](#) shows the waveforms.

Chapter 15: Transient Analysis

Overview of Transient Analysis

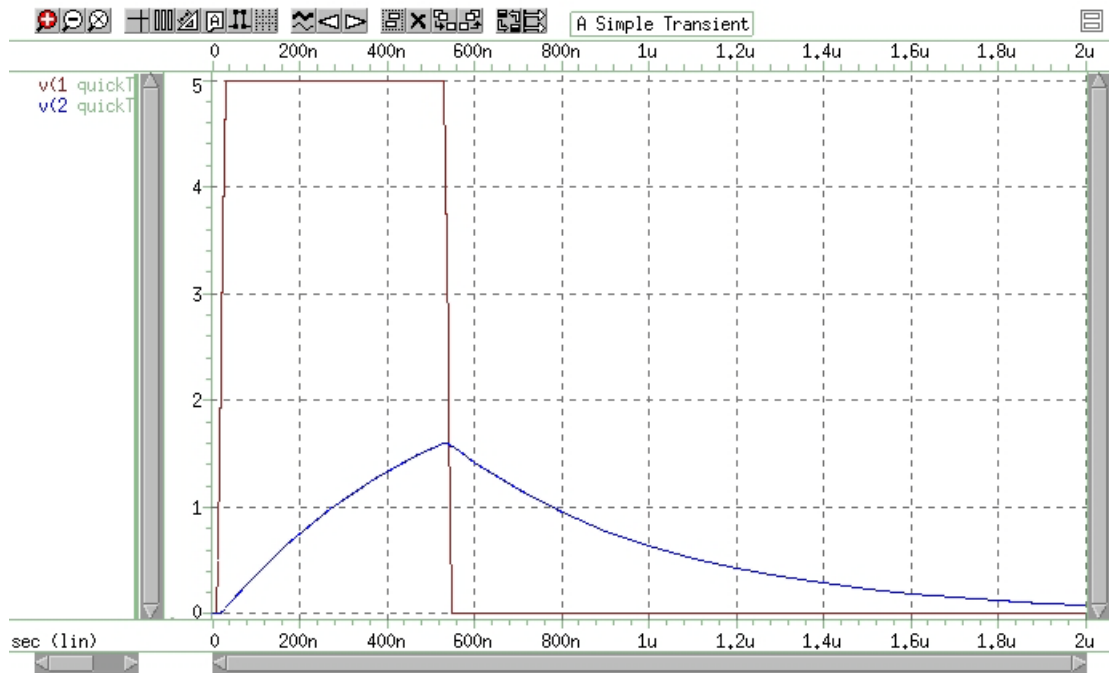


Figure 56 Voltages at RC Network Circuit Node 1 and Node 2

Transient Analysis of an Inverter

As a final example, you can analyze the behavior of the simple MOS inverter shown in Figure 57.

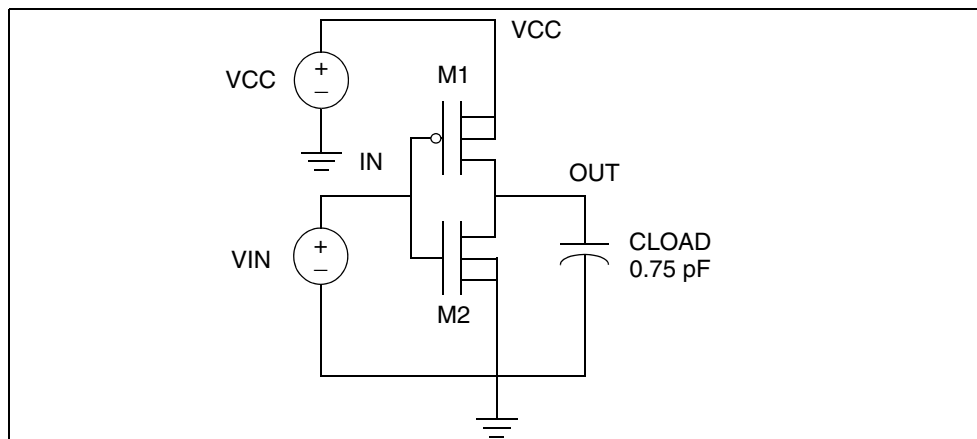


Figure 57 MOS Inverter Circuit

Follow these steps to analyze this behavior:

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

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

You can find the complete netlist for this example in directory
`$installdir/demo/hspice/apps/quickINV.sp`.

2. To run HSPICE, type the following:
`hspice quickINV.sp > quickINV.lis`
3. Use WaveView to examine the voltage waveforms, at the inverter IN and OUT nodes. [Figure 58 on page 528](#) shows the waveforms.

Chapter 15: Transient Analysis

Transient Control Options

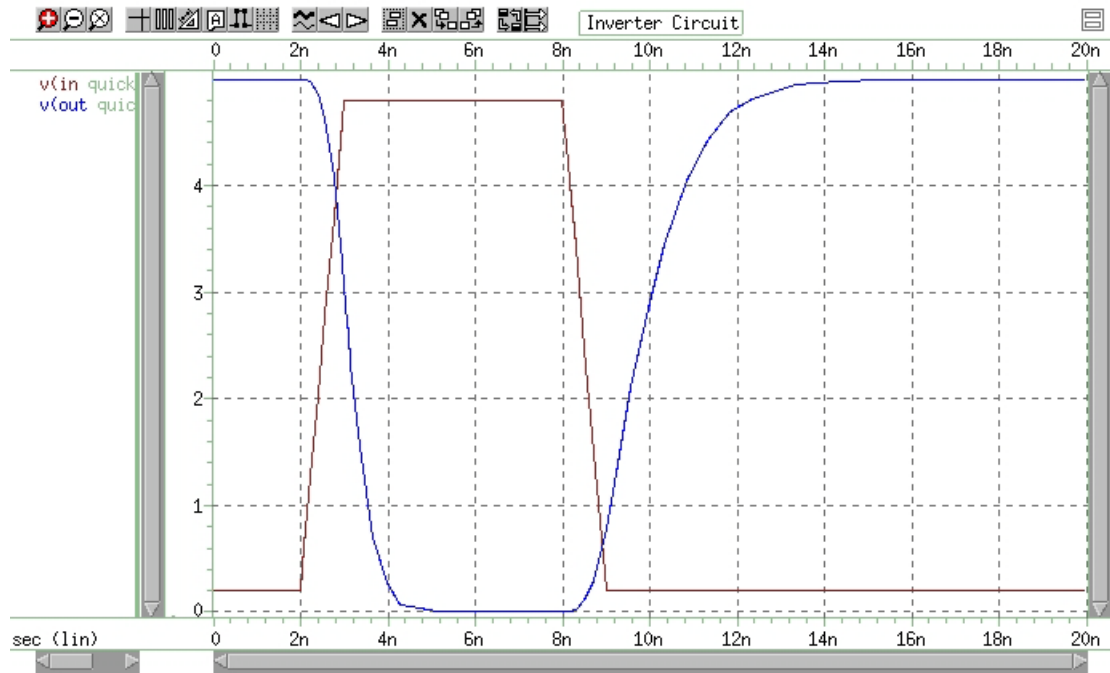


Figure 58 Voltage at MOS Inverter Node 1 and Node 2

Transient Control Options

Method, tolerance, and limit options in this section modify the behavior of transient analysis integration routines. Delta is the internal timestep. TSTEP and TSTOP are the step and stop values in the .TRAN statement.

[Table 56](#) lists the options for RUNLVL.

Table 56 Transient Control Options with RUNLVL Turned On, by Category

Method	Tolerance and Limit		Output
BYPASS	ACCURATE	GMIN	POST
MAXORD	AUTOSTOP	GSHUNT	
METHOD=	BKPSIZ	ITL4	INTERP
Backward-Euler (BE)	BYTOL	MBYPASS	
GEAR	CSHUNT	MAXAMP	
TRAP	DELMAX	MU	
BDF		RUNLVL	
PURETP			
TRCON			

For discussion of METHOD options, see [Numerical Integration Algorithm Controls on page 533](#).

Simulation Speed and Accuracy Using the RUNLVL Option

The RUNLVL algorithm, which is on by default in HSPICE, focuses on a balance between speed and accuracy. The RUNLVL algorithm:

1. Uses an enhanced Local Truncation Error (LTE) method based on nodal voltage for timestep control. This is advantageous because voltage is the target result users want from a simulation, and there is a clear mathematical relation between error tolerance and time step.
2. Adopts a new Newton-Raphson (NR) iteration method for transient analysis. It not only improves the convergence but also makes the convergence faster.
3. Improves the BYPASS algorithm, as well.

The following sections discuss these topics:

- [RUNLVL Features](#)

RUNLVL Features

The RUNLVL algorithm provides the following characteristics:

- Simplifies accuracy control by setting RUNLVL values between 1 and 6 with 6 discrete settings (1=fastest, 6=most accurate).
- Avoids interpolation error in .MEASURE statements by using the interpolating polynomial used by the time integration method.
- Dynamically checks for correct handling of input signals and controlled sources between computed time steps to avoid setting small time steps before transient simulation start.
- Allows HSPICE to take time steps no larger than $(T_{stop}-T_{start})/20$. DELMAX automatically sets $(T_{stop}-T_{start})/20$ if there is no specific setting of DELMAX. The effect is that, for example, HSPICE can take larger time steps for flat regions.

The RUNLVL algorithm scales all simulation tolerances simultaneously and affects time step control, convergence, and model bypass all at once.

This algorithm activates only by use of `.OPTION RUNLVL=value`. Higher values of RUNLVL result in smaller time steps (more Newton-Raphson iterations) to meet stricter error tolerances, and higher simulation accuracy.

A valid value for `.OPTION RUNLVL` is an integer from 1 to 6. Values outside of this range cause an error. The default value for RUNLVL is 3. This is the recommended starting setting. For simulations that require high accuracy use higher values. Use lower values for simulating pure digital or mostly digital circuits. Set `RUNLVL=0` to turn off the option. HSPICE uses the last setting if you specify multiple settings of RUNLVL options.

The `.OPTION RUNLVL` invokes the advanced simulation algorithm, with the default value of `RUNLVL=3`. This is the recommended starting setting. However, you can set it to a higher value if the circuit type is pure analog and/or the simulation needs high accuracy.

Table 57 Guidelines for RUNLVL Settings

Circuit type	RUNLVL Setting
Digital	RUNLVL=1-3

Table 57 Guidelines for RUNLVL Settings

Analog or mixed signal accuracy	RUNLVL=3-5
Cell characterization	RUNLVL=5-6

A *.lis file reports the RUNLVL flag and its effective value. HSPICE sets the RUNLVL option automatically in the `$install_dir/hspice/hspice.ini` file, when it generates the file during the installation process.

All HSPICE simulations first try to find ONE implicit hspice.ini file and take it as the first include file; the search order for hspice.ini is:

1. Current working directory
2. User's home directory
3. `$install_dir/hspice` directory

Interactions Between .OPTION RUNLVL and Other Options

Refer to [Table 58](#) for information on how RUNLVL affects the values of other options. Since the latest algorithm invoked by RUNLVL sets the timestep and error tolerance internally, many transient error tolerance and timestep control options are no longer valid; furthermore, to assure the greatest efficiency of the RUNLVL algorithm, you should let the new engine manage everything itself. HSPICE recommends other Options *not to tune* in the table, as well.

Note: If you set no value for RUNLVL, its value defaults to =3.

Table 58 Options and Interactions

Option	Default value when RUNLVL=0	Default value with RUNLVL=3	User definition ignored	Recommended not to tune
ABSV/VNTOL	50u	50u		x
ABSVAR	500m	500m	x	
ACCURATE* ¹	0	0		
BYPASS* ^a	2	2		
CHGTOL	1.0f	1.0f	x	
DI	100	100		x

Chapter 15: Transient Analysis

Simulation Speed and Accuracy Using the RUNLVL Option

Table 58 Options and Interactions (Continued)

Option	Default value when RUNLVL=0	Default value with RUNLVL=3	User definition ignored	Recommended not to tune
DVDT	4	4	x	
DVTR	1.0k	1.0k	x	
FAST** 2	0	0	x	
FS	250m	250m		x
FT	250m	250m	x	
IMIN/ITL3	3	3	x	
LVLTIM	1	4	x	
METHOD*** 3	TRAP	TRAP		
RELQ	10m	10m	x	
RELTOL	1.0m	1.0m		x
RELV	1.0m	1.0m		x
RELVAR	300.0m	300.0m	x	
RMAX	5	5	x	
RMIN	1.0n	1.0n		x

1. * *ACCURATE* and *BYPASS* notes:

1. If you set *.option ACCURATE* then *HSPICE* limits the *RUNLVL* value to 5 or 6. Specifying a *RUNLVL* less than 5 results in a simulation at *RUNLVL=5*. When both *ACCURATE* and *RUNLVL* are set, the *RUNLVL* algorithm will be used.

2. When *RUNLVL* is set, *BYPASS* is set to 2. Users can re-define the *BYPASS* value by setting *.option BYPASS=<value>*; this behavior is independent of the order of *RUNLVL* and *BYPASS*;

2. **The *FAST* option is disabled by the *RUNLVL* option; setting the *RUNLVL* value to 1 is comparable to setting the *FAST* option.

3. ****RUNLVL* can work with *METHOD=GEAR*; in cases where *GEAR* only determines the numeric integration method during transient analysis, the other options that were previously set by *GEAR* (when there is no *RUNLVL*) now are determined by the *RUNLVL* mode. This behavior is independent of the order of *RUNLVL* and *METHOD*. See the following table.

The interactions of RUNLVL and GEAR are shown in [Table 59](#).

Table 59 RUNLVL option and GEAR method interactions

Option	GEAR without RUNLVL	GEAR with RUNLVL=1-6
BYPASS	0	2
BYTOL	50u	100u
LVLTIM	2	Disabled by runlvl
MAXORD	2	3 for RUNLVL=6 2 for RUNLVL=1-5
MBYPASS	1	2
RMAX	2	Disabled by runlvl

Numerical Integration Algorithm Controls

In HSPICE transient analysis, you can select one of several options solve the circuit differential algebraic equations:

- Backward-Euler
- Gear
- Trapezoidal
- BDF (Backward Differentiation Formulae)

Table 60 Integration Method

Integration Algorithm	Option Settings	Comments
Backward-Euler (BE)	METHOD=GEAR MAXORD=1 or METHOD=GEAR MU=0	Backward-Euler only
GEAR	METHOD=GEAR METHOD=GEAR MAXORD=2 3	Combines GEAR and BE 2nd/3rd order increases accuracy
TRAP	METHOD=TRAP METHOD=TRAP PURETP	Combines Trapezoidal and BE Trapezoidal only

Table 60 Integration Method (Continued)

Integration Algorithm	Option Settings	Comments
BDF	METHOD=BDF	Higher order integration (Backward Differentiation Formulae)
	METHOD=GEAR	HSPICE automatically selects BDF based on circuit type when METHOD=GEAR; To prohibit GEAR from automatically selecting BDF, use .OPTION MAXORD.

The advanced multicore algorithm (-mn) which you can use in multithread simulation (-mt) supports 2nd order GEAR, TRAP, and BDF methods. TRAP is the default method. Each integration algorithm has advantages and disadvantages.

The following sections discuss these topics:

- [TRAP](#)
- [GEAR and Backward-Euler](#)
- [BDF](#)

TRAP

The trapezoidal is often the preferred algorithm because of its high accuracy level and low simulation time. Recommendation: use the pure trapezoidal (PURETP) for oscillators to avoid numerical damping which can cause oscillator simulations to die out when you view output in a waveform plot. Unlike GEAR, using the TRAP method (.OPTION METHOD = TRAP) is not subject to automatic selection after the circuit fails to converge using GEAR method.

GEAR and Backward-Euler

The GEAR method is an appropriate algorithm for convergence. 2nd-order GEAR is more accurate than Backward-Euler and 3rd-order GEAR is more accurate than 2nd-order GEAR. Recommendation: use the GEAR method for those circuit simulations that require high accuracy on current such as leakage current measurement.

If the circuit fails to converge using the Trapezoidal integration method, HSPICE uses the autoconvergence process where it changes to the GEAR method to run the transient analysis again from time=0.

When appropriate, to take advantage of high accuracy on medium to large analog and mixed signal circuits and also improve performance by using fewer iterations per time point, HSPICE automatically switches from method GEAR to BDF. However, circuits with some design constructs such as oscillators may not simulate correctly using BDF. In these cases, you can override the automatic selection of BDF by also using `.OPTION MAXORD = [1|2|3]`.

BDF

The BDF method is a high-order integration method that uses the backward differentiation formulae. Since BDF is for high accuracy applications, HSPICE recommends a RUNLVL setting of 3 or above. The you can use the BDF method with multithreading.

Two tolerance options are available to the user for the BDF method: `.OPTIONS BDFRTOL` (relative) and `BDFATOL` (absolute); each has a default of 1e-3. BDF can provide a speed enhancement to mixed-signal circuit simulation, especially for circuits with a large number of devices. The BDF method provides no performance advantage for use with small circuits in standard cell characterization. BDF could be an alternative to GEAR when TRAP fails to converge.

`METHOD=BDF` supports the following models/devices/elements:

- Bulk MOSFET, levels 1-54
- SOI MOSFET, levels 57, 70
- BJT, levels 1, 2, 3
- Diodes, all
- Resistors, all
- Capacitors (excludes DC block)
- Independent sources: V and I
- Dependent sources: E/F/G/H
- L (excludes AC choke)

Chapter 15: Transient Analysis

Dynamic Check Using the .BIASCHK Statement

- K (excludes magnetic core, ideal transformer)
- Signal integrity elements: B (IBIS buffer)/S/ W/ T

Note: BDF issues a warning in the `.lis` file if it encounters an unsupported model. The message is similar to: `WARNING!!!, netlist contains 'unsupported models', HSP-BDF is disabled.`

The `BDFATOL` and `BDFRTOL` options operate independent of `.OPTIONS RUNLVL` and `ACCURATE` settings with the following exception:

If either `.OPTION RUNLVL` or `ACCURATE` follows an `.OPTION BDFATOL` or `BDFRTOL` value, the `RUNLVL` or `ACCURATE` setting overrides the tolerance of the BDF algorithm. If you set `ACCURATE` with or without `RUNLVL`, the default for `BDFATOL` or `BDFRTOL` always defaults to `1e-5`.

RUNLVL	BDFATOL and BDFRTOL
0	1e-4
1	1e-2
2	1e-2
3	1e-3
4	1e-4
5	1e-4
6	1e-5

Dynamic Check Using the .BIASCHK Statement

The `.BIASCHK` statement can monitor the voltage bias, current, device-size, expression and region during transient analysis, and reports:

- Element name
- Time
- Terminals
- Bias that exceeds the limit
- Number of times the bias exceeds the limit for an element

For the syntax and description of this statement, see the `.BIASCHK` command in the *HSPICE Reference Manual: Commands and Control Options*.

HSPICE saves the information as both a warning and a BIASCHK summary in the *.lis file. You can use this command only for active elements, resistors, capacitors, and subcircuits.

You can also use .OPTION BIASFILE and .OPTION BIAWARN with a .BIASCHK statement.

The following limitations apply to the .BIASCHK statement:

- Only these models support .BIASCHK: diode, jfet, nmos, pmos, bjt, and r/c models, as well as subcircuits.
- Only W and L MOSFET models support device-size checks.
- There is support for wildcards in element and model names and *except* definitions but *not* in expressions.

Four methods are available to check the data with the .BIASCHK command:

- Limit and noise method
- Maximum method
- Minimum method
- Region method

Note: Only MOSFET models support the region method of data checking.

Limit and Noise Method

For a transient simulation that uses the limit and noise method to check the data, use the following syntax:

For local_max

```
v(tn-1) > limit_value
```

The bias corresponds to any one of the following two conditions:

- $v(tn-1) > v(tn) \ \&\& \ v(tn-1) \geq v(tn-2)$
- $v(tn-1) \geq v(tn) \ \&\& \ v(tn-1) > v(tn-2)$

local_min: The minimum bias after the time last local max occurs.

A transient analysis records the local_max if it is greater than the limit. In the summary HSPICE reports after transient analysis, the local_max(next) replaces the local_max(current) when the following comparison is true:

Chapter 15: Transient Analysis

Dynamic Check Using the .BIASCHK Statement

$local_max(current) - local_min < noise \ \&\& \ local_max(next) - local_min < noise$
 $\&\& \ local_max(current) < local_max(next)$

At the end of the simulation, the listing file reports all `local_max` values as BIASCHK warnings. During other analyses, HSPICE issues warnings when the value you want to check is greater than the `limit_value` you specify.

Maximum Method

Use this syntax for a transient simulation that specifies the maximum method:

For `local_max`:

$v(tn-1) > max_value$

The bias corresponds any one of the following two conditions:

- $v(tn-1) > v(tn) \ \&\& \ v(tn-1) \geq v(tn-2)$
- $v(tn-1) \geq v(tn) \ \&\& \ v(tn-1) > v(tn-2)$

HSPICE lists all `local_max` values as BIASCHK warnings during a transient analysis. During other analyses, HSPICE issues warnings when the value you want to check is greater than `max_value` you specify.

Minimum Method

For a transient simulation that specifies the minimum method to check the data, use the following syntax:

For `local_min`:

$v(tn) < min_value$

The bias corresponds any one of the following two conditions:

- $v(tn-1) < v(tn) \ \&\& \ v(tn-1) \leq v(tn-2)$
- $v(tn-1) \leq v(tn) \ \&\& \ v(tn-1) < v(tn-2)$

During a transient analysis, all `local_min` values are listed as BIASCHK warnings. During other analyses, warnings are issued when the value you want to check is smaller than `min_value` you specify.

Region Method

This method is only for MOSFET models. Three regions exist:

- cutoff
- linear
- saturation

When the specified transistor enters and exits during transient analysis, the specified region is reported.

The `biaschk.sp` demo example is a netlist that uses the `.BIASCHK` command for a transient simulation. You can find the sample netlist for this example in: `$installdir/demo/hspice/apps/biaschk.sp`

Using BIASCHK with HPP

The `.BIASCHK` command is available for use with HSPICE Precision Parallel (HPP) in the following applications: as an expression monitor and as an element and model monitor.

HPP plus `.BIASCHK` netlist syntax for the expression monitoring function is:

```
.BIASCHK 'expression' [limit=lim] [noise=ns]
+ [max=max] [min=min]
+ [simulation=op|dc|tr|all] [monitor=v|i|w|l]
+ [tstart=time1] [tstop=time2] [autostop]
+ [interval=time]
```

HPP plus `.BIASCHK` netlist syntax for an element and model monitoring function is:

```
.BIASCHK type terminal1=t1 [terminal2=t2]
+ [limit=lim] [noise=ns] [max=max] [min=min]
+ [simulation=op|dc|tr|all] [monitor=v|i]
+ [name=name1,name2,...]
+ [mname=modname_1,modname_2,...]
+ [tstart=time1] [tstop=time2] [autostop]
+ [except=name_1,name_2,...]
+ [interval=time] [sname=subckt_name1,subckt_name2,...]
```

See the `.BIASCHK` command in the *HSPICE Reference Manual: Commands and Control Options*. See also [HSPICE Precision Parallel \(-hpp\)](#).

Storing and Restoring Checkpoint Files

Store/restore is a feature that creates checkpoint files describing a running process during transient analysis; the operating system can later reconstruct

the process from the contents of this file. This feature is not supported in standalone HSPICE-RF.

Table 61 Supported Platforms

Linux RHEL	Linux SUSE	Sun/Solaris	Windows
Yes	Yes	No	No

Note: The `-restore` operation should be submitted on a machine that has the same kernel version as the machine used to store, otherwise, a failure may occur.

The following sections discuss these topics:

- [Store by Using the .STORE Command](#)
- [Store by Interrupting the Simulation Process](#)
- [Restore Operation](#)
- [Usage Requirements](#)

Store by Using the .STORE Command

You can trigger the store function either by issuing a `.STORE` command in the netlist or by interrupting the running simulation process.

Use the following syntax in a netlist:

```
.STORE [TYPE=IC/MEMDUMP]
+      [FILE=save_file_prefix]
+      [TIME=time1] [TIME=time2] ... [TIME=timeN]
+      [REPEAT=period]
+      [TRANTIME=0/1]
+      [SAVE_ON_KILL=0/1]
```

Argument	Description
TYPE=IC/MEMDUMP	Stores checkpoint data to either an IC type or a memory dump file. If unspecified, the default checkpoint file is of the TYPE=IC and the name prefix is same as the HSPICE output file.
FILE=save_file_prefix	Changes the prefix of the output file names.

Argument	Description
TIME=time1,time2,...timeN	Collects checkpoint data beginning at <i>time1</i> after the start of transient analysis. It then updates the checkpoint data every 21,600 wall-clock seconds if no checkpoint period is specified.
REPEAT=period	If you specify a nonzero period, new checkpoint data is collected at every period, starting at transient time=0 and overwriting previous interval checkpoint data. If a nonzero time1 is specified, checkpoint data is collected at time1 + period * n, where n is an integer. Period is always calculated based on time1. If repeat=0, the store operation is disabled. If you set both time=0 and repeat=0, checkpoint data is saved at transient time=0 only.
TRANTIME=0/1	<ul style="list-style-type: none">▪ If set to 0, time1 and period are taken as wall-clock time.▪ If set to 1, time1 and period are transient times or times is smaller than TSTOP. Note: If TYPE=MEMDUMP, TRANTIME is ignored.
SAVE_ON_KILL=0/1	If set to 1, the checkpoint data is saved on kill and halts the simulation.

For more information of using the `.STORE` command, see *HSPICE Reference Manual: Commands and Control Options User Guide*.

Store by Interrupting the Simulation Process

The following system command can interrupt HSPICE transient analysis simulation and create checkpoint files: `kill -USR2 process_id`. This command allows users to trigger a checkpoint without exiting; the simulation continues after the checkpoint is done.

Note: Interrupting a transient analysis simulation while checkpoint files are being written may cause HSPICE to crash.

After the checkpoint files are generated, use `kill -9 process_id` to stop the simulation, if needed.

Example

If system command `kill -USR2 process_id` is submitted, HSPICE generates checkpoint files, then continues the current simulation. If the simulation is terminated later on, the checkpoint files can be used to continue the simulation when you enter the command-line `restore` command.

Restore Operation

The restore operation takes place at the command line:

```
hspice -restore checkpoint_file
```

The `checkpoint_file` specifies from which simulation the checkpoint data is to be restored.

Any output files generated by the previous simulation should not be removed. After the restore simulation is done, the output files will be updated.

Example

In this example, HSPICE output file will be tagged with “1e-7”, like `test.1e-7.tr0`:

```
HSPICE -i test.sp -restore test.1e-7.ic0 -o test
```

In this example, HSPICE output file will have “save” tagged, like `test.save.tr0`:

```
HSPICE -i test.sp -restore test.save.ic0 -o test
```

Usage Requirements

- Store and restore can be performed on 64-bit Linux operation systems.
- Store and restore can be performed on 32-bit Linux operation systems with `exec_shield_randomize` turned off. Check with your system administrator for details.
- Restore must be performed on the same platform that the store operation used.
- The restore-machine must have at least as much (available) RAM as the process was using when the checkpoint data was saved.
- Store/restore is not available if you are using Cadence® PSF output.

Specifying Monte Carlo or Temperature Sweeps

Using the `.TRAN` statement you can either specify a Monte Carlo sweep or a temperature sweep. A `.temp` statement with more than one value can be used to sweep the temperatures of interest.

In the following example, HSPICE performs a transient Monte Carlo analysis at each temperature value specified in the `.temp` statement— -30C, 40C, and 125C, respectively:

```
.temp -30 40 125  
.tran 1n 20n sweep monte=10
```

Example netlist: Simulating the netlist below results in three waveform files, `netlist.tr0`, `netlist.tr1`, and `netlist.tr2` with each displaying the results of the Monte Carlo sweep at each temperature value.

```
.options post  
.temp -30 40 125  
.tran 1n 5n sweep monte=10  
.param resval=aunif(1000,400)  
vsrc_one 1 0 5v  
r_one 1 2 resval tc1=0.02 tc2=0.01  
c_one 2 0 1u  
.end
```

Troubleshooting: Internal Timestep, Measurement Errors

- [Troubleshooting ‘Timestep Too Small’ Errors](#)
- [Stepsize Increases During a Simulation](#)
- [How TSTEP Affects a Transient Simulation](#)
- [Troubleshooting .MEASUREMENT Issues](#)

Troubleshooting ‘Timestep Too Small’ Errors

These are the usual steps to follow when you get an “internal time step too small” in transient analysis errors. The best approach is to incrementally change the values of these options, one at a time. Note the time immediately following the timestep error in the list file. If your simulation gets further into the run, then the option is beneficial and you may wish to try higher or lower values as appropriate.

1. Be sure you are using the latest version of HSPICE if you can. Improvements are continuously made to convergence algorithms.

Chapter 15: Transient Analysis

Troubleshooting: Internal Timestep, Measurement Errors

2. Comment out all timestep and convergence options you already have and try increasing the value of `.OPTION RUNLVL` as a first step.

The `RUNLVL` option is turned on by default starting with version 2006.09 to `level=3`. It implements improved convergence techniques. If a higher `RUNLVL` such as 5 or 6 is set, try a lower `RUNLVL` to get convergence.

Note: Remove any other convergence options when you use `RUNLVL`.

3. Increase `.OPTION ITL4`

This is the number of iterations HSPICE will try at one time point, before giving up and taking a smaller time step. The default is 8.

Suggested values:

```
.option itl4 = 50  
.option itl4 = 100
```

Note: `.OPTION ITL4` is the same as `.OPTION IMAX`. 100 is the maximum value.

4. Use `GSHUNT` and `CSHUNT` to add small amounts of conductance and or capacitance from each node to ground. Together or alone, these options can help solve timestep too small problems caused by either high-frequency oscillations or numerical noise.

```
.option gshunt=1e-13 cshunt=1e-17  
.option gshunt=1e-12 cshunt=1e-16  
.option gshunt=1e-11 cshunt=5e-15  
.option gshunt=1e-10 cshunt=1e-15  
.option gshunt=1e-9 cshunt=1e-14
```

5. Increase the timestep value, to step over possible model discontinuities.

From original timestep settings, change the `.TRAN` statement to incrementally increase `TSTEP`:

```
.tran (2)*tstep tstop  
.tran (2.5)*tstep tstop  
.tran (3)*tstep tstop
```

6. Using `.OPTION METHOD=GEAR` may help certain high gain analog (such as op-amps) and/or oscillatory circuits (such as a ring oscillators) during transient analysis by changing integration methods.

```
.option method=gear
```

- Investigate the device models used. Be sure the version of the models was developed for or qualified with the version of HSPICE you are using. For CMOS devices, make sure you have finite terminal capacitances and resistances. For level 49, be sure you have the model parameters as specified in the following example: (these are samples, not defaults)

```
.model mname nmos level=49 version=3.2  
+ cj=5e-4 cjsw=1e-10 cgd0=1e-10 cgs0=1e-10 rs=1e-9 rd=1e-9
```

In the case of BJT device, be sure to have the following model parameters set (again, examples, not defaults).

```
.model mname npn rb=50 r c=.4 re=1e-3
```

Stepsize Increases During a Simulation

If you are using `.option POST=2` and creating an ASCII output file, due to limitations when formatting ASCII output files, HSPICE must increase the spacing between points after writing 100k time points. This allows HSPICE to output data from 0 to TSTOP.

Note: NOTE: The simulation accuracy is not affected, only the ASCII `*.tr0` file will show this issue.

This can be fixed in either of the following ways.

- Add `.option POST_VERSION=2001` to the netlist. The time points are now output as double-precision and no digits are lost.
- Use `.option POST=1` in the netlist. This will create a binary output file.

How TSTEP Affects a Transient Simulation

With the introduction of the RUNLVL algorithm, the impact of TSTEP has been greatly reduced to the point that it can be ignored except for niche applications like INTERP. When using RUNLVL=0, however, you need to be careful not to set TSTEP too small for long, low frequency simulations in which the soft upper limit of $RMAX * TSTEP$ may still force more timestep solutions than is necessary.

TSTEP is specified in the transient analysis netlist command.

```
.TRAN TSTEP1 TSTOP1 TSTEP2 TSTOP2 ... TSTEPN TSTOPN
```

The most common usage is a single TSTEP/TSTOP pair. For example:

Chapter 15: Transient Analysis

Troubleshooting: Internal Timestep, Measurement Errors

```
.TRAN 0.1ps 100ns
```

TSTEP has a variety of effects on the operation of HSPICE.

1. When the RUNLVL algorithm is in use, the minimum timestep is determined by the RUNLVL algorithm:
 - RUNLVL “in use” is defined as HSPICE version 2004.09 and OPTION RUNLVL > 0. For HSPICE version \geq 2006.09, RUNLVL=3 by default.
 - The RMIN and RMAX options are ignored.
 - The maximum timestep has a “soft” limit of $(T_{\text{stop}} - T_{\text{start}}) / 20$. The RUNLVL algorithm allows the timestep to exceed $R_{\text{MAX}} * T_{\text{STEP}}$.
 - The DELMAX option can be used to override the RUNLVL algorithm's choice of maximum timestep. When DELMAX is used, TSTEP has no impact on simulation accuracy (except when used in conjunction with .OPTION INTERP (see # 3 below)).
2. When RUNLVL=0 (discouraged):
 - The minimum timestep is defined as $T_{\text{STEP}} * R_{\text{MIN}}$.
 - The default value of RMIN is 1e-9.
 - The maximum timestep is defined as $T_{\text{STEP}} * R_{\text{MAX}}$.
 - The default value of RMAX is 5.
 - These limits are “hard” limits.
 - If the simulation engine attempts to choose a timestep $< T_{\text{STEP}} * R_{\text{MIN}}$, a Timestep Too Small Error will result. The simulation engine is never allowed to take a larger timestep than $T_{\text{STEP}} * R_{\text{MAX}}$.
3. When OPTION INTERP is used (off by default):
 - INTERP forces the probed expressions to be printed to the output file every multiple of TSTEP. Probed expression will *only* be printed to the output at multiples of TSTEP.
 - INTERP has no impact on the simulator's internal timesteps.
 - The probed values printed to the output are linearly interpolated between the simulator solved timesteps preceding and following the $N * T_{\text{STEP}}$ time to print at.

Note: Only use INTERP when specifically needed. For example, FFT post-processing.

Troubleshooting .MEASUREMENT Issues

If `.MEASURE` results are incorrect compared to the waveforms, the differences you see may be due to one or more of the following issues.

- You are not comparing the same point. Make sure that the proper nodes and sweep points are being used for each comparison.
- If `.option INTERP` is in your netlist, remove it. HSPICE only saves data points at the interval defined by the `tstep` parameter in the `.TRAN` statement. For example, for the `.TRAN` statement:

```
.tran 1n 100n
```

HSPICE saves 100 points at 1ns intervals to the `.tr0` file. The lack of precision can cause issues with your measurements.

- In your `.TRAN` statement, use of the `START` keyword to delay output generation should be removed as it interferes with `.measure` calculations.
- If a `.measure` statement uses the result of previous `.meas` statement, then the calculation starts when the previous result is found. Until the previous result is found, it outputs zero.

Chapter 15: Transient Analysis

Troubleshooting: Internal Timestep, Measurement Errors

Spectrum Analysis

Describes HSPICE implementation of spectrum analysis based on the Fourier transforms.

Spectrum analysis represents a time-domain signal within the frequency domain. It most commonly uses the Fourier transform. A Discrete Fourier Transform (DFT) uses sequences of time values to determine the frequency content of analog signals in circuit simulation.

HSPICE ships numerous examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files and [Fourier Analysis Examples](#).

The following sections discuss these topics:

- [Spectrum Analysis \(Fourier Transform\)](#)
- [.FFT Analysis](#)
- [Examining the FFT Output](#)
- [AM Modulation](#)
- [Balanced Modulator and Demodulator](#)
- [Signal Detection Test Circuit](#)

Spectrum Analysis (Fourier Transform)

This section describes the Fourier and FFT Analysis flow for HSPICE.

Chapter 16: Spectrum Analysis

Spectrum Analysis (Fourier Transform)

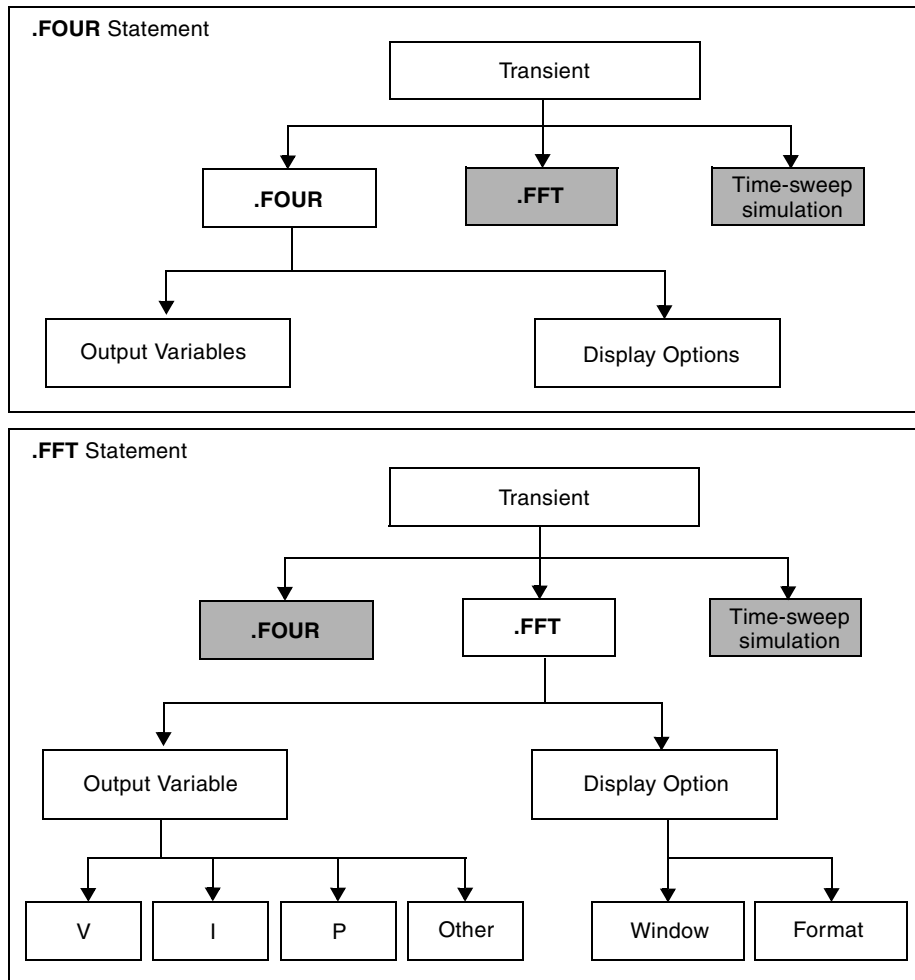


Figure 59 Fourier and FFT Analysis

HSPICE provides two different Fourier analyses.

- .FOUR is the same as is available in SPICE 2G6: a standard, fixed-window analysis tool. The .FOUR statement performs a Fourier analysis, as part of the transient analysis.
- .FFT is a much more flexible Fourier analysis tool. Use it for analysis tasks that require more detail and precision.

Using the Fourier-Related Statements and Options

For syntax and examples, see the following commands and control options in the *HSPICE Reference Manual: Commands and Control Options*:

- `.FFT`
- `.FOUR`
- `.MEASURE FFT`
- `.OPTION FFT_ACCURATE`
- `.OPTION ACCURATE`
- `.OPTION FFTOUT`

Fourier Accuracy

Fourier Accuracy is dependent on transient simulation accuracy. For best accuracy, set small values for `.OPTION RMAX` or `.OPTION DELMAX`. For maximum accuracy, set `.OPTION DELMAX` to $1/(500 \cdot \text{frequency})$. For circuits with very high resonance factors (high-Q circuits, such as crystal oscillators, tank circuits, and active filters), set `DELMAX` to less than $1/(500 \cdot \text{frequency})$ where, frequency refers to fundamental frequency.

Fourier Equation

The total harmonic distortion is the square root of the sum of the squares, of the second through ninth normalized harmonic, times 100, expressed as a percent:

Equation 47

$$THD = \frac{1}{R1} \cdot \left(\sum_{m=2}^9 R_m^2 \right)^{1/2} \cdot 100\%$$

The following equation calculates the Fourier coefficients:

Chapter 16: Spectrum Analysis
Spectrum Analysis (Fourier Transform)

Equation 48

$$g(t) = \sum_{m=0}^9 C_m \cdot \cos(mt) + \sum_{m=0}^9 D_m \cdot \sin(mt)$$

The following equations calculate values for the preceding equation:

Equation 49

$$C_m = \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} g(t) \cdot \cos(mt) \cdot dt$$

Equation 50

$$D_m = \frac{1}{\pi} \cdot \int_{-\pi}^{\pi} g(t) \cdot \sin(mt) \cdot dt$$

Equation 51

$$g(t) = \sum_{m=0}^9 C_m \cdot \cos(m \cdot t) + \sum_{m=0}^9 D_m \cdot \sin(m \cdot t)$$

The following equations approximate the C and D values:

Equation 52

$$C_m = \sum_{n=0}^{500} g(n \cdot \Delta t) \cdot \cos\left(\frac{2 \cdot \pi \cdot m \cdot n}{501}\right)$$

Equation 53

$$D_m = \sum_{n=0}^{500} g(n \cdot \Delta t) \cdot \sin\left(\frac{2 \cdot \pi \cdot m \cdot n}{501}\right)$$

The following equations calculate the magnitude and phase:

Equation 54

$$R_m = (C_m^2 + D_m^2)^{1/2}$$

Equation 55

$$\Phi_m = \arctan\left(\frac{C_m}{D_m}\right)$$

Example 1

The following is input-content for an .OP, .TRAN, or .FOUR analysis. This example uses demonstration netlist `four.sp`, which is available in the directory `$installdir/demo/hspice/apps`.

```
CMOS INVERTER
*
M1 2 1 0 0 NMOS W=20U L=5U
M2 2 1 3 3 PMOS W=40U L=5U
VDD 3 0 5
VIN 1 0 SIN 2.5 2.5 20MEG
*
.MODEL NMOS NMOS LEVEL=3 CGDO=0.2N CGSO=0.2N CGB0=2N
.MODEL PMOS PMOS LEVEL=3 CGDO=0.2N CGSO=0.2N CGB0=2N
.OP
.TRAN 1N 500N
.FOUR 20MEG V(2)
.PRINT TRAN V(2) V(1)
.END
```

Example 2

```
*****
cmos inverter
**** fourier analysis   tnom = 25.000 temp = 25.000 ****
fourier components of transient response v(2)
dc component=2.430D+00
harmonic   frequency   fourier      normalized   phase      normalized
  no        (hz)      component    component    (deg)      phase (deg)
1          20.0000x    3.0462      1.0000      176.5386    0.
2          40.0000x   115.7006m   37.9817m   -106.2672   -282.8057
3          60.0000x   753.0446m   247.2061m   170.7288    -5.8098
4          80.0000x    77.8910m    25.5697m   -125.9511   -302.4897
5         100.0000x  296.5549m   97.3517m   164.5430    -11.9956
6         120.0000x   50.0994m    16.4464m   -148.1115   -324.6501
7         140.0000x  125.2127m   41.1043m   157.7399    -18.7987
8         160.0000x   25.6916m     8.4339m    172.9579     -3.5807
9         180.0000x   47.7347m    15.6701m    154.1858    -22.3528
total harmonic distortion= 27.3791 percent
```

.FFT Analysis

The `.FFT` statement uses the internal time point values. By default, `.FFT` uses a second-order interpolation to obtain waveform samples, based on the number of points that you specify.

You can use windowing functions to reduce the effects of waveform truncation on the spectral content. You can also use the `.FFT` command to specify:

- Output format
- Output frequency range
- Start and stop time point
- Fundamental frequency
- Window type
- Number of sampling time points

Using Windows in FFT Analysis

One problem with spectrum analysis in circuit simulators is that the duration of the signals is finite, although adjustable. Applying the FFT method to finite-duration sequences can produce inadequate results. This occurs because DFT assumes periodic extensions and causes spectral leakage.

The effect occurs when the finite duration of the signal does not result in a sequence that contains a whole number of periods. This is especially true when you use FFT to detect or estimate signals – that is, to detect weak signals in the presence of strong signals, or to resolve a cluster of equal-strength frequencies.

In FFT analysis, windows are frequency-weighting functions that HSPICE applies to the time-domain data, to reduce the spectral leakage associated with finite-duration time signals. Windows are smoothing functions, which peak in the middle frequencies, and decrease to zero at the edges. Windows reduce the effects of discontinuities, as a result of finite duration. [Figure 60](#) shows the windows available in HSPICE. [Table 62 on page 555](#) lists the common performance parameters, for FFT windows.

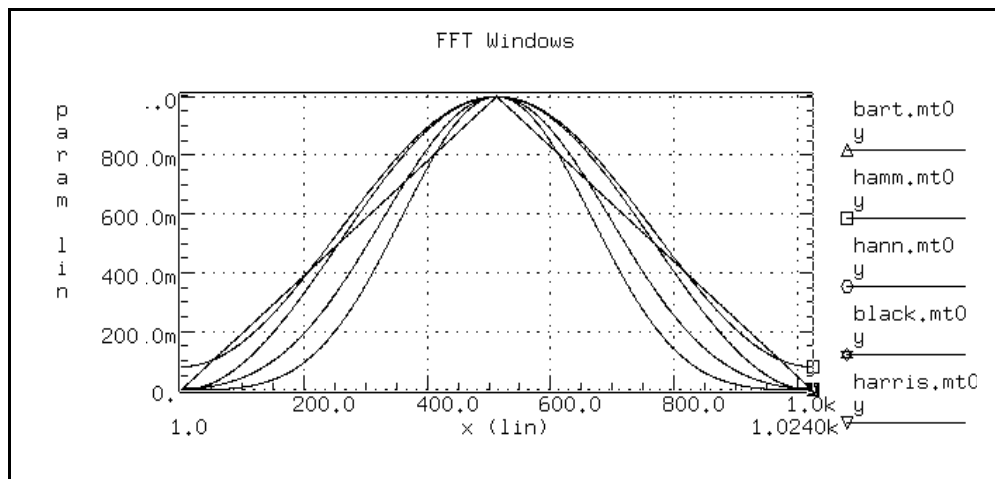


Figure 60 FFT Windows

The most important parameters in Table 62 are:

- Highest side-lobe level (to reduce bias, the lower the better).
- Worst-case processing loss (to increase detectability, the lower the better).

Table 62 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	$W(n)=1,$ $0 \leq n < NP^1$	-13	-6	0.89	3.92
Bartlett	$W(n)=2n/(NP-1),$ $0 \leq n \leq (NP/2)-1$ $W(n)=2-2n/(NP-1),$ $NP/2 \leq n < NP$	-27	-12	1.28	3.07
Hanning	$W(n)=0.5-0.5[\cos(2\pi n/(NP-1))],$ $0 \leq n < NP$	-32	-18	1.44	3.18
Hamming	$W(n)=0.54-0.46[\cos(2\pi n/(NP-1))],$ $0 \leq n < NP$	-43	-6	1.30	3.10

Table 62 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	$W(n)=0.42323$ $-0.49755[\cos(2\pi n/(NP-1))]$ $+0.07922\cos[\cos(4\pi n/(NP-1))]$, $0 \leq n < NP$	-58	-18	1.68	3.47
Blackman-Harris	$W(n)=0.35875$ $-0.48829[\cos(2\pi n/(NP-1))]$ $+0.14128[\cos(4\pi n/(NP-1))]$ $-0.01168[\cos(6\pi n/(NP-1))]$, $0 \leq n < NP$	-92	-6	1.90	3.85
Gaussian a=2.5	$W(n)=\exp[-0.5a^2(NP/2-1-n)^2/(NP)^2]$, $0 \leq n \leq (NP/2)-1$	-42	-6	1.33	3.14
a=3.0	$W(n)=\exp[-0.5a^2(n-NP/2)^2/(NP)^2]$,	-55	-6	1.55	3.40
a=3.5	$NP/2 \leq n < NP$	-69	-6	1.79	3.73
Kaiser-Bessel a=2.0	$W(n)=I_0(x_2)/I_0(x_1)$ $x_1=pa$	-46	-6	1.43	3.20
a=2.5	$x_2=x_1*\sqrt{1-(2(NP/2-1-n)/NP)^2}$, $0 \leq n \leq (NP/2)-1$	-57	-6	1.57	3.38
a=3.0	$x_2=x_1*\sqrt{1-(2(n-NP/2)/NP)^2}$,	-69	-6	1.71	3.56
a=3.5	$NP/2 \leq n < NP$ I_0 is the zero-order modified Bessel function	-82	-6	0.89	3.74

1. NP is the number of points used for the FFT analysis.

Some compromise usually is necessary, to find a suitable window-filtering for each application. As a rule, window performance improves with functions of higher complexity (those listed lower in the table).

- The Kaiser window has an ALFA parameter, which adjusts the compromise between different figures of merit for the window.
- The simple rectangular window produces a simple bandpass truncation, but results in the classical Gibbs phenomenon.

- The Bartlett or triangular window has good processing loss, and good side-lobe roll-off, but lacks sufficient bias reduction.
- The Hanning, Hamming, Blackman, and Blackman-Harris windows use progressively more complicated cosine functions. These functions provide smooth truncation, and a wide range of side-lobe level and processing loss.
- The last two windows in the table, Kaiser-Besser, are parameterized-windows. Use these windows to adjust the side-lobe level, the 3 dB bandwidth, and the processing loss.

Figure 61 and Figure 62 show the characteristics of two typical windows.

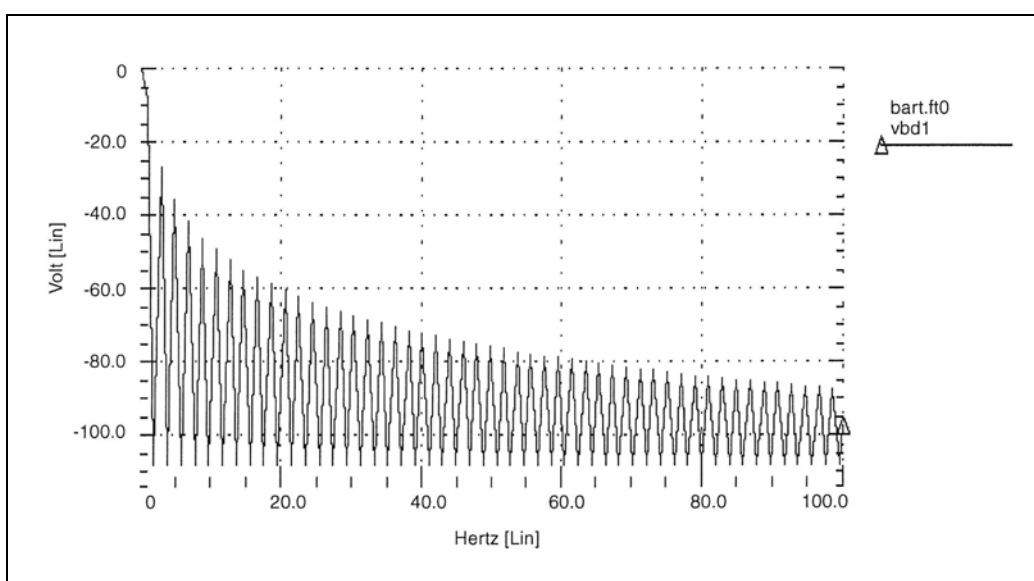


Figure 61 Bartlett Window Characteristics

Chapter 16: Spectrum Analysis

Examining the FFT Output

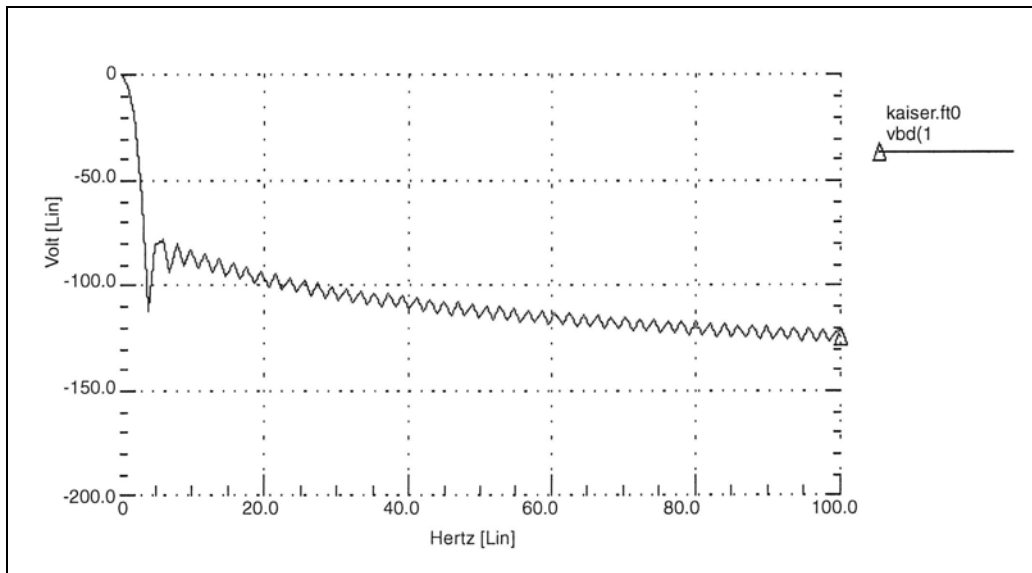


Figure 62 Kaiser-Bessel Window Characteristics, ALFA=3.0

Examining the FFT Output

HSPICE prints FFT analysis results in tabular format, in the `.lis` file. These results use parameters in the `.FFT` statement. HSPICE prints normalized magnitude values, unless you specify `FORMAT=UNORM`, in which case it prints unnormalized magnitude values. The number of printed frequencies is half the number of points (NP) specified in the `.FFT` statement.

- If you use `FMIN` to specify a minimum frequency, or `FMAX` to specify a maximum frequency, HSPICE prints only the specified frequency range.
- If you use `FREQ` to specify a frequency, HSPICE outputs harmonics of this frequency.

HSPICE generates a `.ft#` file and the listing file for each FFT output variable that contains data to display in FFT analysis waveforms (such as in Custom WaveView). You can view the magnitude in dB, and the phase in degrees.

In the following sample FFT analysis `.lis` file output, the header defines parameters in the FFT analysis.

```

***** Sample FFT output extracted from the .lis file
fft test ... sine
***** fft analysis   tnom= 25.000   temp= 25.000
***** fft components of transient response v(1)
Window:   Rectangular
First Harmonic:   1.0000k
Start Freq:   1.0000k
Stop Freq:   10.0000k
dc component: mag(db)= -1.132D+02 mag= 2.191D-06 phase= 1.800D+02

frequency   frequency   fft_mag   fft_mag   fft_phase
index       (hz)       (db)     (deg)
2   1.0000k   0.       1.0000   -3.8093m
4   2.0000k  -125.5914  525.3264n  -5.2406
6   3.0000k  -106.3740  4.8007u   -98.5448
8   4.0000k  -113.5753  2.0952u   -5.5966
10  5.0000k  -112.6689  2.3257u   -103.4041
12  6.0000k  -118.3365  1.2111u   167.2651
14  7.0000k  -109.8888  3.2030u   -100.7151
16  8.0000k  -117.4413  1.3426u   161.1255
18  9.0000k  -97.5293   13.2903u   70.0515
20  10.0000k -114.3693  1.9122u   -12.5492

```

The preceding example specifies a frequency of 1 kHz, and a THD up to 10 kHz, which corresponds to the first ten harmonics.

The highest FFT output frequency might not match the specified FMAX, due to adjustments that HSPICE makes.

Table 63 describes the output of the FFT analysis.

Table 63 .FFT Output Description

Column Heading	Description
Frequency Index	Runs from 1 to NP/2, or the corresponding index for FMIN and FMAX. The DC component, corresponding to the 0 index, displays independently.
Frequency	The actual frequency, associated with the index.
fft_mag (dB), fft_mag	The first FFT magnitude column is in dB. The second FFT magnitude column is in units of the output variable. HSPICE normalizes the magnitude, unless you specify UNORM format.
fft_phase	Associated phase, in degrees.

Notes:

- Use the following formula as a guideline to specify a frequency range for FFT output:
frequency increment = 1.0/(STOP - START)
Each frequency index is a multiple of this increment. To obtain a finer frequency resolution, maximize the duration of the time window or specify more time points (larger NP).
- FMIN and FMAX have no effect on the .ft0, .ft1, ..., .ftn files.
- If you specify FFTOUT in an .OPTION statement, HSPICE can print results of THD, SNR, SFDR, SNDR, and ENOB and then sort the harmonics of fundamental by magnitude size.

Assume that f_0 is the fundamental frequency.

- THD is total harmonic distortion, which [Equation 56](#) computes as:

Equation 56

$$THD = \frac{\sqrt{(\text{sum}(\text{mag}(n \cdot f) \cdot \text{mag}(n \cdot f)))}}{\text{mag}(f_0)}$$

- SNR is the ratio of signal to noise, which following formula computes as:

$$SNR = 10\log \left[\frac{(\text{mag}(f_0) \cdot \text{mag}(f_0))}{\text{sum}(\text{mag}(f) \cdot \text{mag}(f))} \right]$$

The f loop over the whole spectrum except f_0 and its harmonics. If you specify FMIN/FMAX, f loops over the spectrum between FMIN and FMAX except f_0 and its harmonics, as well as all the frequency components above FMAX.

- SFDR is the spurious-free dynamic range and is the distance from the fundamental input signal to the highest spur (in dB). SFDR involves both magnitude distance and frequency separation.
- SNDR is the signal to noise and distortion ratio, which is the level of the fundamental divided by the square root of the sum of squares of all frequency components other than the fundamental frequency. The following formula computes SNDR:

Equation 57

$$SNDR = 10\log \left[\frac{\text{mag}(f_0) \cdot \text{mag}(f_0)}{\text{sum}(\text{mag}(f) \cdot \text{mag}(f))} \right]$$

- ENOB is the effective number of bits; the following formula computes ENOB:

$$ENOB = \frac{(SNDR - 1.76\text{db})}{6.02}$$

The `f` loops over the whole spectrum except f_0 . If you specify `FMIN/FMAX`, `f` loops over the spectrum between `FMIN` and `FMAX` except f_0 ; `f` also loops all the frequency components above `FMAX`.

Measuring FFT Output Information

You can measure all of the FFT output information in the previous section by using the following syntax:

Measuring frequency component at certain frequency point:

```
.MEASURE FFT result  
+ Find [vm|vp|vr|vi|vdb|im|ip|ir|ii|idb] (signal) AT=freq
```

Measuring THD of a signal spectrum up to a certain harmonic:

```
.MEASURE FFT result THD signal_name [NBHARM=num]
```

Default: `NBHARM`=maximum harmonic in FFT result.

Measuring SNR/SNDR/ENOB of a signal up to a certain frequency point:

```
.MEASURE FFT result [SNR|SNDR|ENOB] signal_name  
+ [NBHARM=num|MAXFREQ=val] [BINSIZ=num]
```

Default: `NBHARM`=maximum harmonic in FFT result. HSPICE considers all the frequency components above `NBHARM` to be noise.

`MAXFREQ`=maximum frequency in FFT result. HSPICE considers all the frequency components above `MAXFREQ` to be noise. `BINSIZ=num` filters out noise components. HSPICE calculates the noise component above index “`fundamental_freq_idx+BINSIZ+1`”. HSPICE filters out the noise components within the bin.

The default value of `BINSIZ` is 0 in HSPICE.

When you set the window function in FFT analysis, you can use `BINSIZ` to filter out the noise component caused by window function.

Measuring SFDR of a signal from minfreq to maxfreq:

```
.MEASURE FFT result SFDR signal_name [MAXFREQ=val] [MINFREQ=val]
```

Default: MINFREQ=0.

MAXFREQ=Maximum frequency in FFT result.

Syntax to perform FFT measurements from previous simulation results:

```
hspice -i *.tr0 -meas measure_file
```

For more information, see [.MEASURE FFT](#) in the *HSPICE Reference Manual: Commands and Control Options*.

AM Modulation

This example uses demonstration netlist `exam1.sp`, which is available in directory `$installdir/demo/hspice/fft`. The example shows a 1 kHz carrier (FC), which a 100 Hz signal (FM) modulates.

AM Modulation

```
.OPTION post
.PARAM sa=10 offset=1 fm=100 fc=1k td=1m
VX 1 0 AM(sa offset fm fc td)
Rx 1 0 1

.TRAN 2u 50m
.FFT V(1) START=10m STOP=40m FMIN=833 FMAX=1.16K
.END
```

The following equation describes the voltage at node 1, which is an AM signal:

Equation 58

$$v(1) = sa \cdot (\text{offset} + \sin(\omega_m(\text{Time} - td))) \cdot \sin(\omega_c(\text{Time} - td))$$

You can expand the preceding equation, as follows.

Equation 59

$$v(1) = (sa \cdot \text{offset} \cdot \sin(\omega_c(\text{Time} - td)) + 0.5 \cdot sa \cdot \cos((\omega_c - \omega_m)(\text{Time} - td)))$$

Equation 60

$$- 0.5 \cdot sa \cdot \cos((\omega_c + \omega_m)(Time - td))$$

where

Equation 61

$$\omega_c = 2\pi f_c$$

$$\omega_m = 2\pi f_m$$

The preceding equations indicate that $v(1)$ is a summation of three signals, with the following frequency:

$$f_c, (f_c - f_m), \text{ and } (f_c + f_m)$$

This is the carrier frequency and the two sidebands.

See also [Behavioral Amplitude Modulator](#) in the *HSPICE User Guide: Advanced Analog Simulation and Analysis*.

Graphical Output

[Figure 63](#) and [Figure 64 on page 564](#) display the results.

- [Figure 63](#) shows the time-domain curve for node 1.
- [Figure 64](#) shows the frequency-domain components, for the magnitude of node 1.

The carrier frequency is 1 kHz, with two sideband frequencies 100 Hz apart.

The third, fifth, and seventh harmonics are more than 100 dB below the fundamental, indicating excellent numerical accuracy. The time-domain data contains an integer multiple of the period, so you do not need to use windowing.

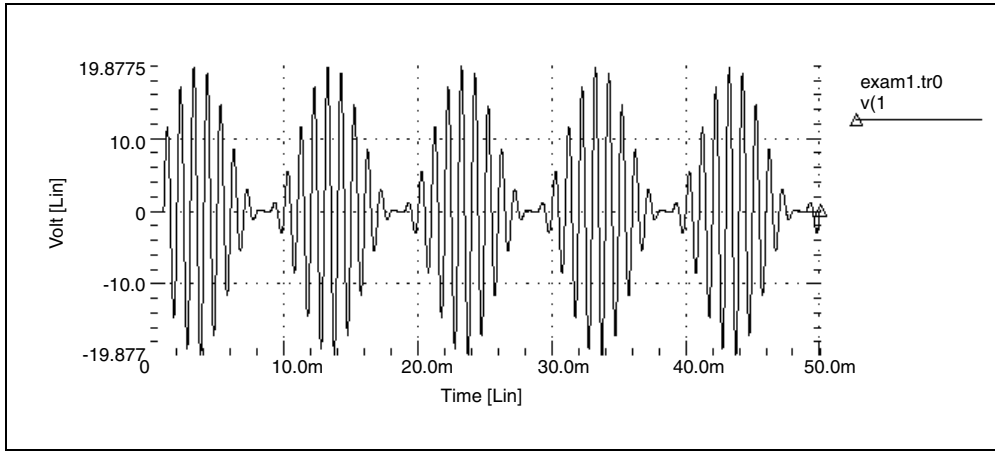


Figure 63 AM Modulation

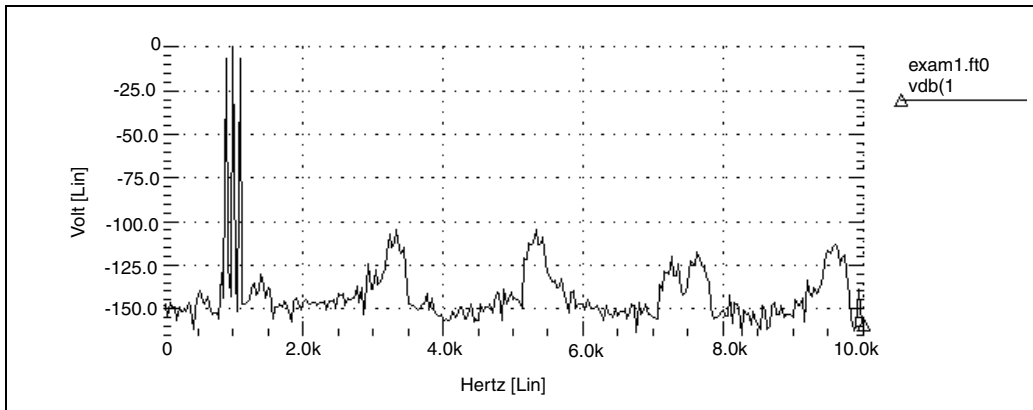


Figure 64 AM Modulation Spectrum

Balanced Modulator and Demodulator

Demodulation, or detection, recovers a modulating signal from the modulated output voltage. The netlist, in the Input and Output Listing section that follows, shows this process. This example uses HSPICE behavioral models, and FFT analysis, to confirm the validity of the process, in the frequency domain.

The low-pass filter uses the Laplace element. This filter introduces delay in the output signal, which causes spectral leakage if you do not use FFT windowing. However, if you use window-weighting to perform FFT, you eliminate most of the spectral leakage. The THD of the two outputs, shown in “Input and Output Listing,” verifies this. HSPICE expects a 1 kHz output signal, so specify a 1 kHz frequency in the `.FFT` command. Also specify `FMAX`, to provide the first few harmonics in the output listing, for THD calculations.

Input and Output Listing

The sample input and output listing files are in the following directory:

```
$installdir/demo/hspice/fft/balance.sp
```

Figure 65 through Figure 73 show the signals, and their spectral content. The modulated signal contains only the sum, and the difference of the carrier frequency and the modulating signal (1 kHz and 10 kHz). At the receiver end, this example recovers the carrier frequency, in the demodulated signal. This example also shows a 10 kHz frequency shift, in the above signals (to 19 kHz and 21 kHz).

A low-pass filter uses a second-order Butterworth filter, to extract the carrier frequency. A Harris window significantly improves the noise floor in the filtered output spectrum, and reduces THD in the output listing (from 9.23 percent to 0.047 percent). However, this example needs a filter with a steeper transition region, and better delay characteristics, to suppress modulating frequencies below -60 dB. Figure 68 is a normalized filtered output signal waveform.

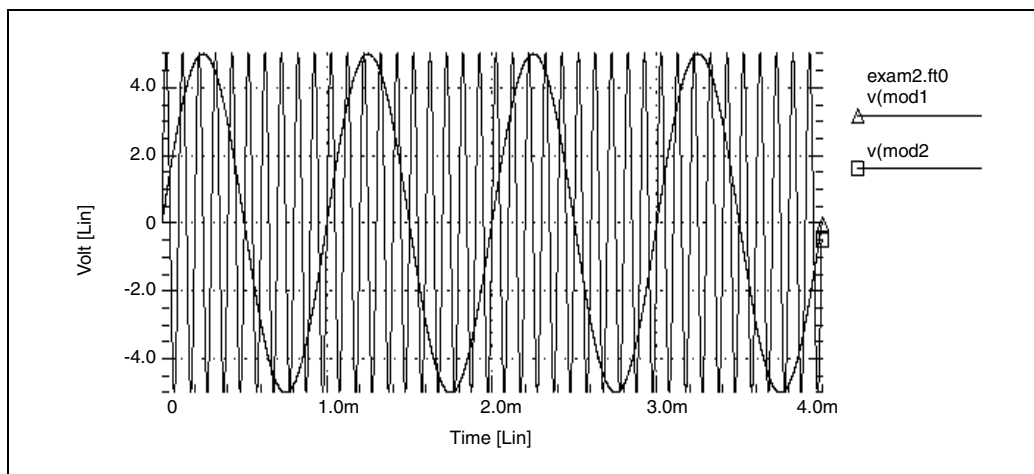


Figure 65 Modulating and Modulated Signals

Chapter 16: Spectrum Analysis
Balanced Modulator and Demodulator

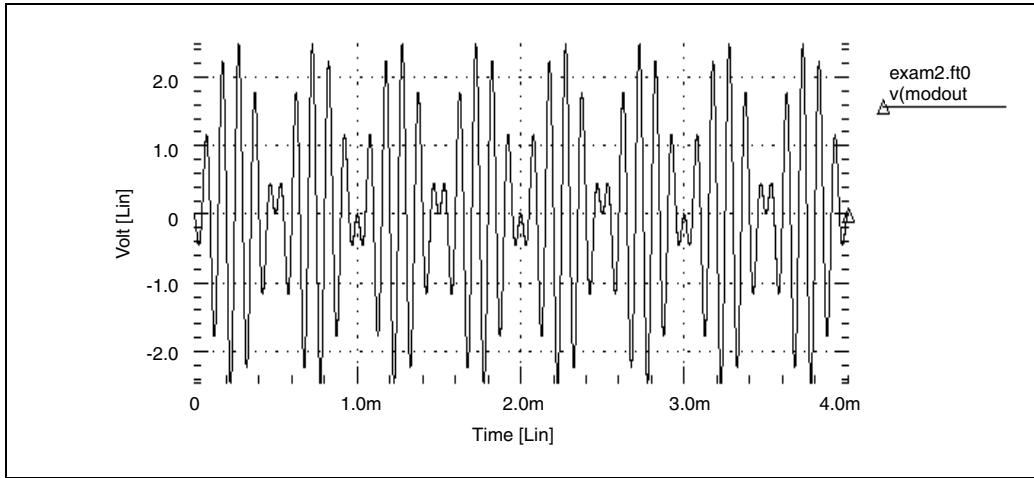


Figure 66 Modulated Signal

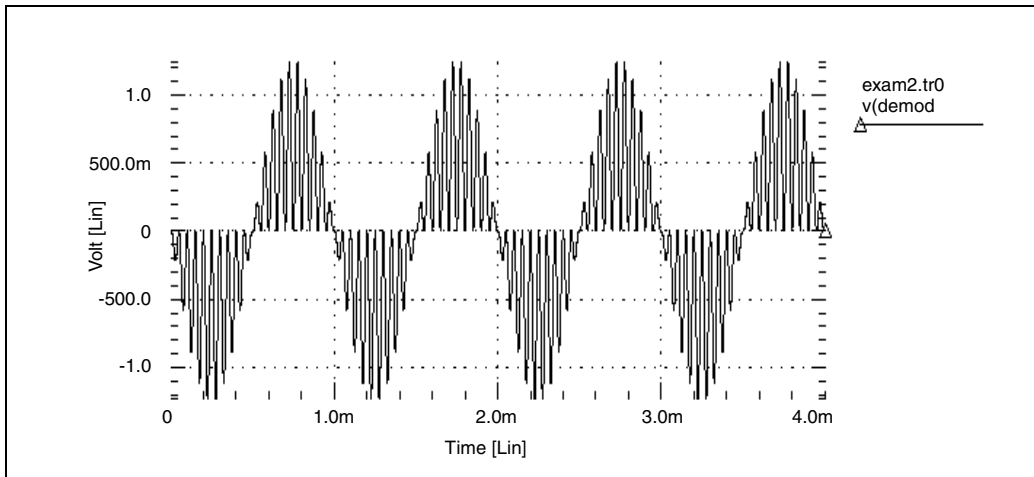


Figure 67 Demodulated Signal

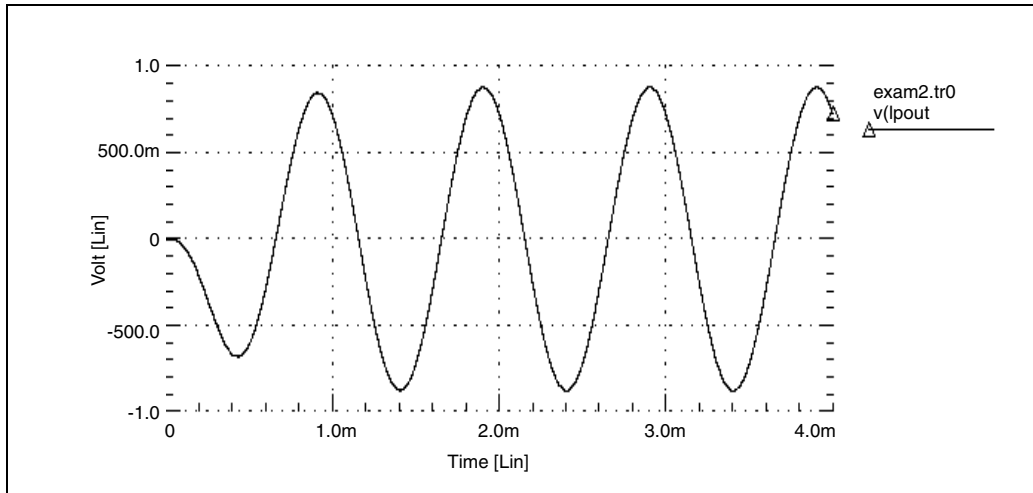


Figure 68 Filtered Output Signal

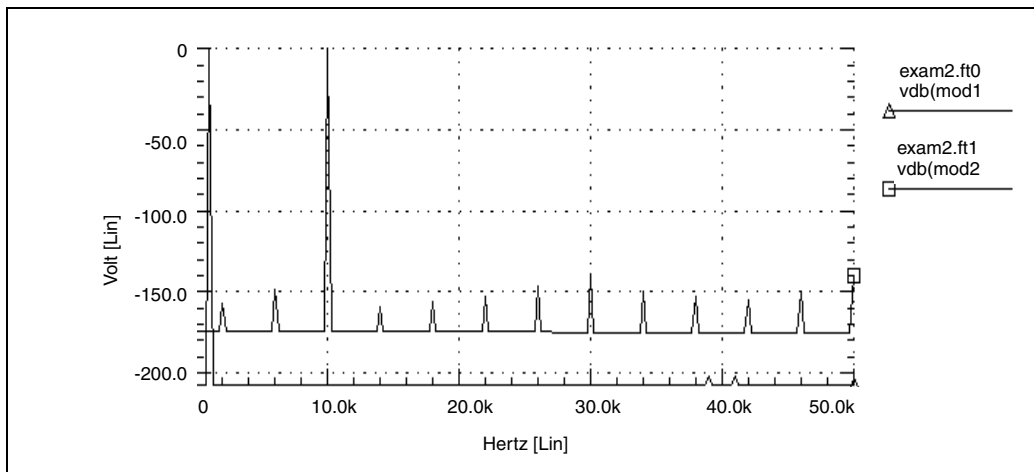


Figure 69 Modulating and Modulated Signal Spectrum

Chapter 16: Spectrum Analysis
Balanced Modulator and Demodulator

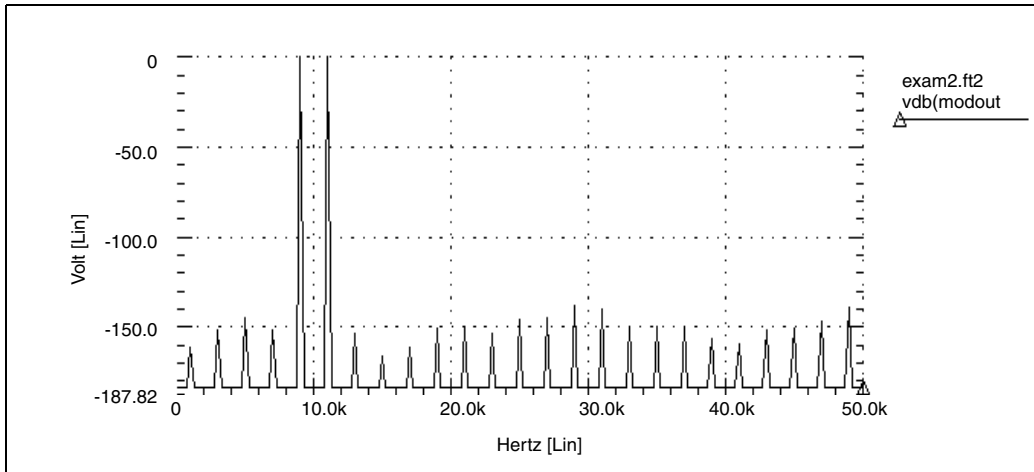


Figure 70 Modulated Signal Spectrum

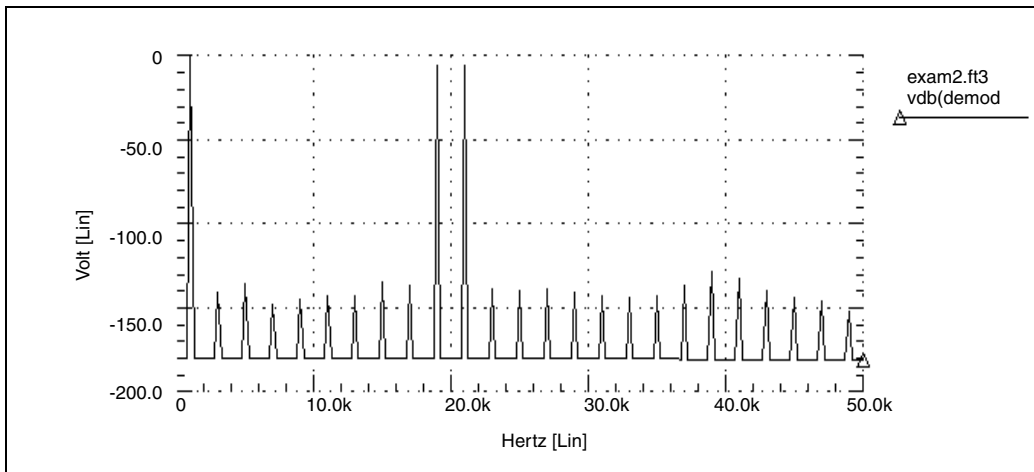


Figure 71 Demodulated Signal Spectrum

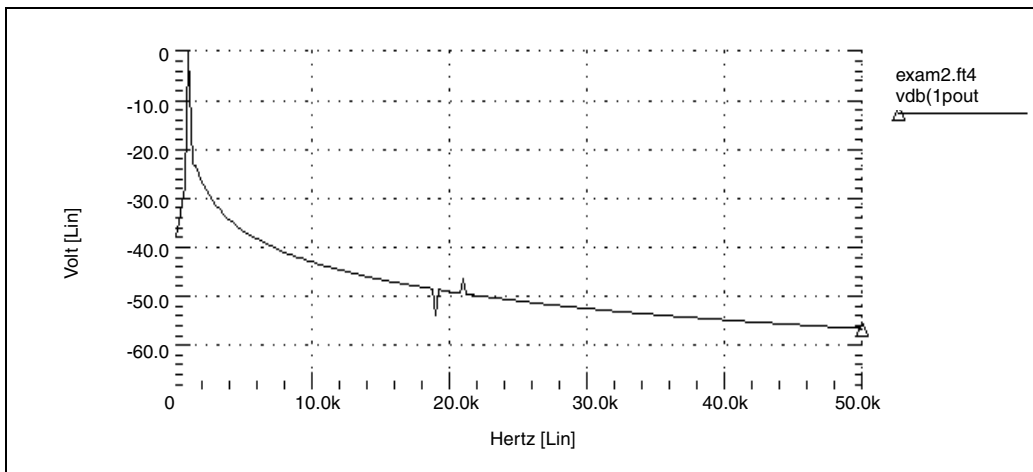


Figure 72 Filtered Output Signal (no window)

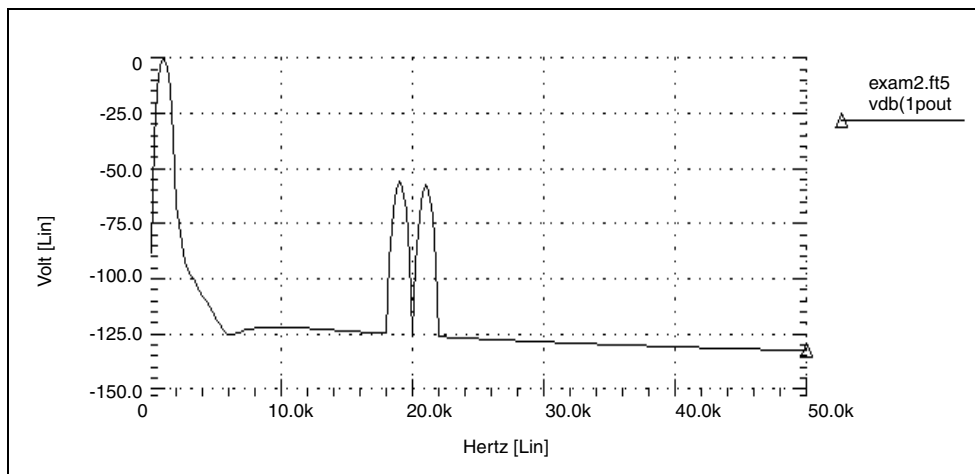


Figure 73 Filtered Output Signal (Blackman-Harris window)

Signal Detection Test Circuit

This example is a high-frequency mixer test circuit. It illustrates the effect of using a window to detect a weak signal, in the presence of a strong signal that

is at a nearby frequency. This example adds two high-frequency signals, with a 40 dB separation (amplitudes are 1.0 and 0.01).

Input Listing

The sample input listing file is in the following directory:

```
$installdir/demo/hspice/fft/exam3.sp
```

Output

Figure 74 shows the rectangular window. Compare this with the spectra of the output for all FFT window types, as shown in Figure 75 through Figure 81. Without windowing, HSPICE does not detect the weak signal because of spectral leakage.

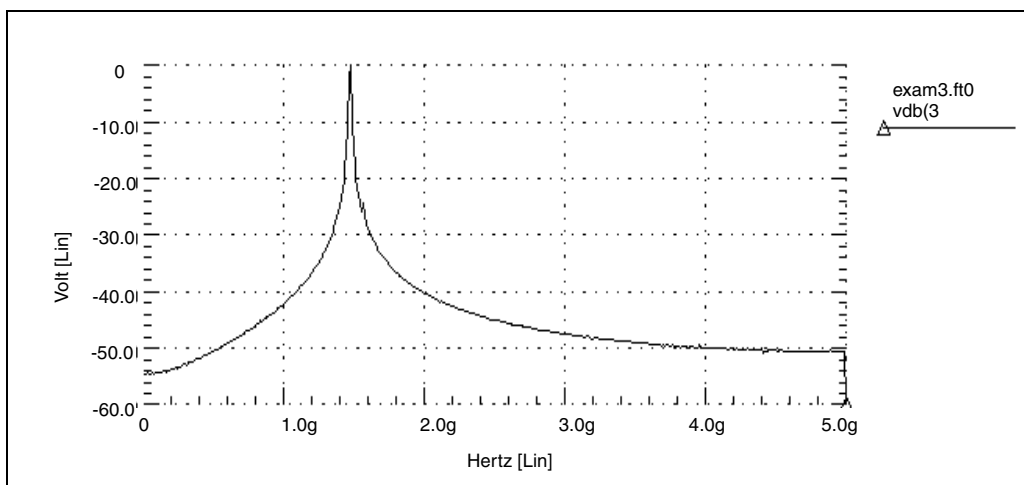


Figure 74 Mixer Output Spectrum, Rectangular Window

- In the Bartlett window (Figure 75), the noise floor increases dramatically, compared to the rectangular window (from -55, to more than -90 dB).
- The cosine windows (Hanning, Hamming, Blackman, and Blackman-Harris) all produce better results than the Bartlett window. However, the Blackman-Harris window provides the highest degree of separation for the two tones, and the lowest noise floor.
- The final two windows (Figure 80 and Figure 81) use the $ALFA=3.0$ parameter, which is the default value in HSPICE. These two windows also produce acceptable results, especially the Kaiser-Bessel window, which sharply separates the two tones, and has a noise floor of almost -100-dB.

HSPICE processes high frequencies, and this example demonstrates the numerical stability and accuracy of the FFT spectrum analysis algorithms.

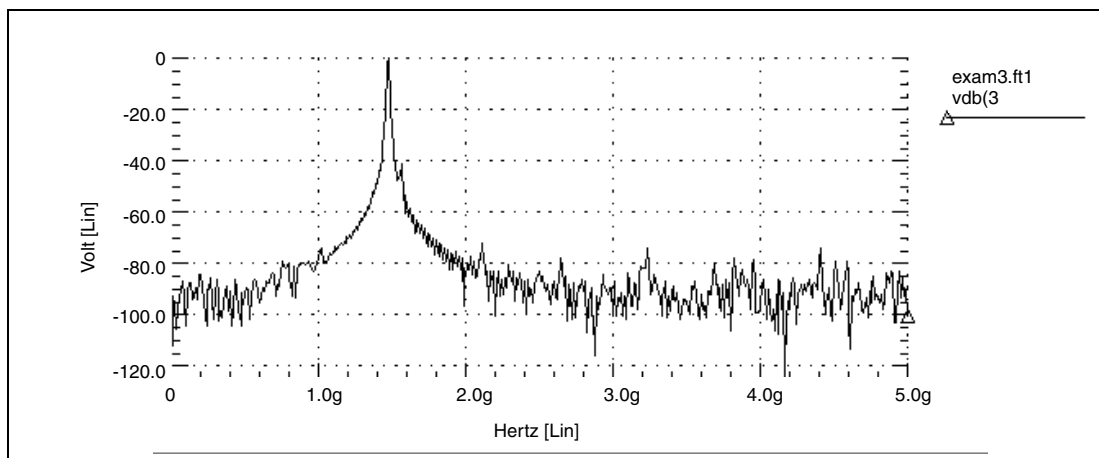


Figure 75 Mixer Output Spectrum, Bartlett Window

Chapter 16: Spectrum Analysis
Signal Detection Test Circuit

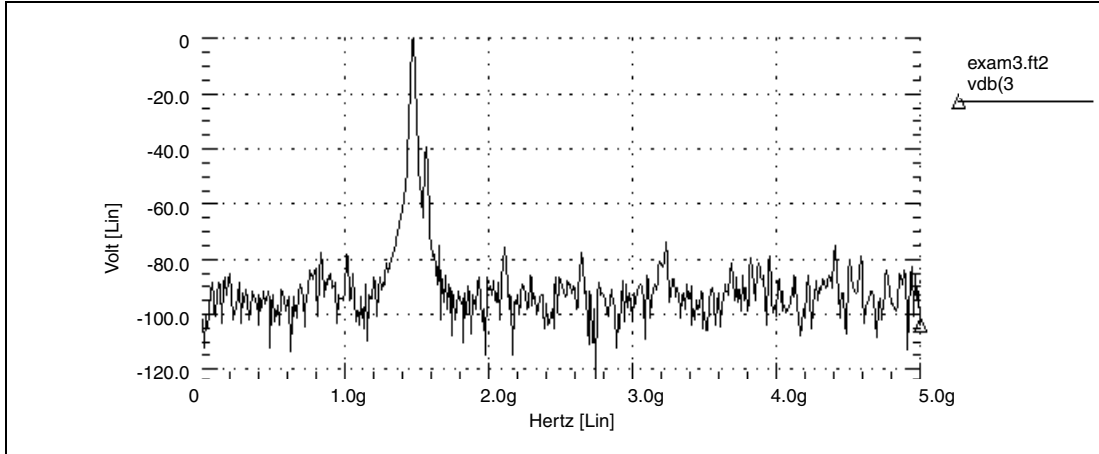


Figure 76 Mixer Output Spectrum, Hanning Window

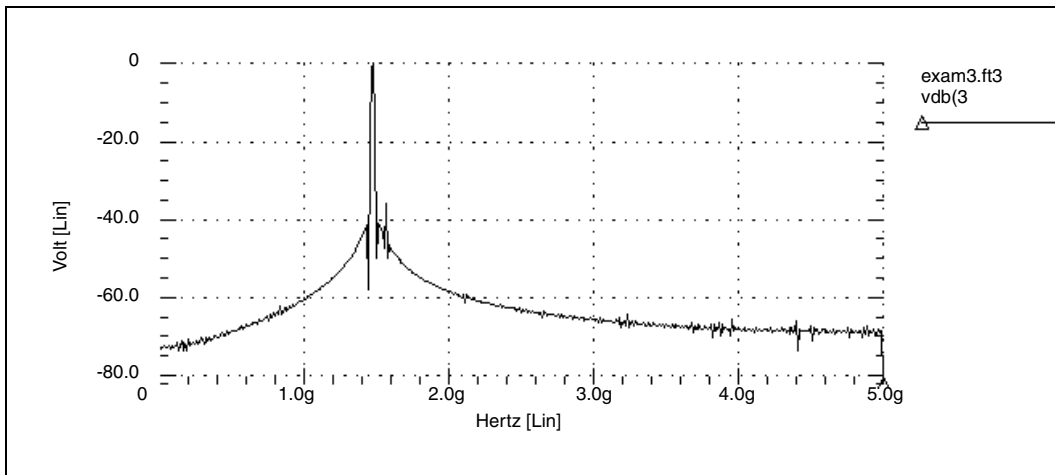


Figure 77 Mixer Output Spectrum, Hamming Window

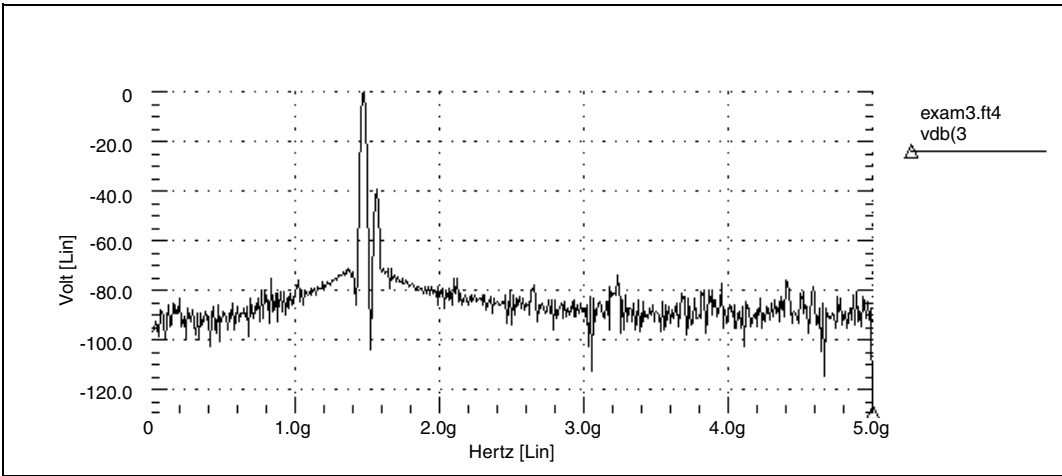


Figure 78 Mixer Output Spectrum, Blackman Window

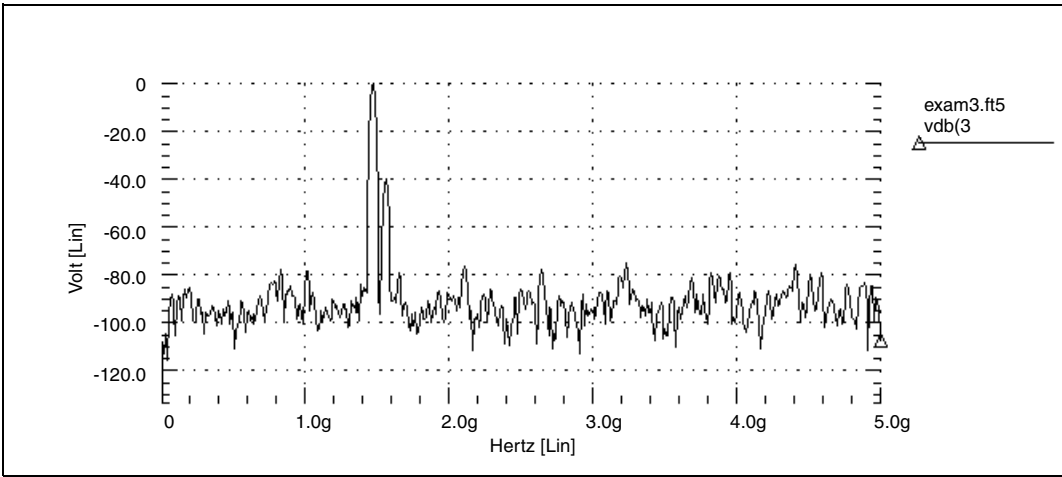


Figure 79 Mixer Output Spectrum, Blackman-Harris Window

Chapter 16: Spectrum Analysis
Signal Detection Test Circuit

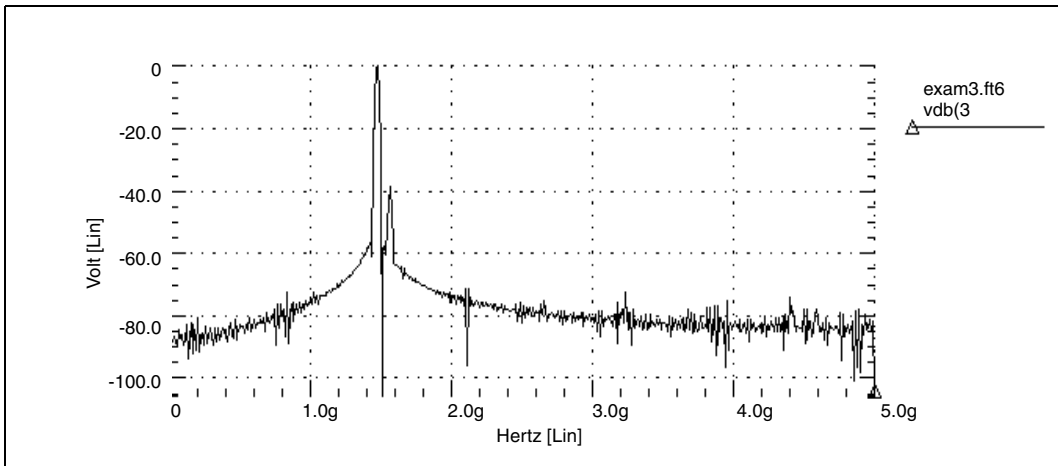


Figure 80 Mixer Output Spectrum, Gaussian Window

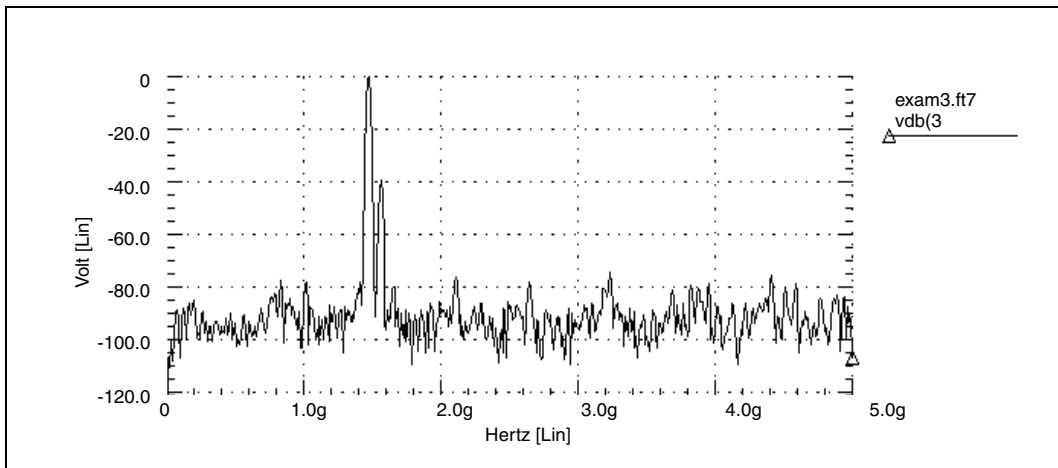


Figure 81 Mixer Output Spectrum, Kaiser-Bessel Window

Pole-Zero Analysis

Describes how to use pole/zero analysis in HSPICE.

HSPICE ships numerous examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files and [Filters Examples](#).

You can use pole/zero analysis in HSPICE to study the behavior of linear, time-invariant networks. You can apply the results to the design of analog circuits, such as amplifiers and filters. Use pole/zero analysis to determine the stability of a design, or to calculate the poles and zeroes to specify in a POLE function (see [Using Pole-Zero Analysis on page 576](#)).

Pole/zero analysis uses the .PZ (Pole/Zero) statement, instead of pole/zero (POLE function) and Laplace (LAPLACE function) transfer function modeling. See [Using Pole-Zero Analysis on page 576](#) for discussion of this technology.

Overview of Pole-Zero Analysis

In pole/zero analysis, a network transfer function describes a network. For any linear time-invariant network, you can use this general form to write the function:

Equation 62

$$H(s) = \frac{N(s)}{D(s)} = \frac{a_0 s^m + a_1 \cdot s^{(m-1)} + \dots + a_m}{b_0 s^n + b_1 \cdot s^{(n-1)} + \dots + b_n}$$

In the factorized form, the general function is:

Chapter 17: Pole-Zero Analysis

Using Pole-Zero Analysis

Equation 63

$$H(s) = \frac{a_0}{b_0} \cdot \frac{(s + z_1)(s + z_2) \dots (s + z_i) \dots (s + z_m)}{(s + p_1)(s + p_2) \dots (s + p_j) \dots (s + p_m)}$$

- The roots of the numerator $N(s)$ (that is, z_i) are the zeros of the network function.
- The roots of the denominator $D(s)$ (that is, p_j) are the poles of the network function.
- S is a complex frequency.

The dynamic behavior of the network depends on the location of the poles and zeros, on the network function curve (complex plane). The (real) poles are the natural frequencies of the network. You can graphically deduce the magnitude and phase curve of most network functions from the location of its poles and zeros (reference 2).

[References on page 594](#) lists a variety of source materials that address:

- Transfer functions of physical systems.
- Design of systems and physical modeling.
- Interconnect transfer function modeling.

Using Pole-Zero Analysis

HSPICE RF uses the exact matrix approach and the Muller method, while HSPICE uses only the Muller method to calculate the roots of the $N(s)$ and $D(s)$ polynomials.

Matrix Approach

The matrix approach uses the singular-value matrix decomposition algorithm. It applies primarily to a network that has no frequency-dependent elements. In this case, HSPICE RF writes the $D(s)$ function as the determinant of the network matrices, $D(s) = \det(G + sC)$, where G is the frequency-independent conductance matrix and C is the capacitance matrix. The poles can be the *eigen* values of the matrix equation $(G + sC) X = 0$, where X is the *eigen* vector.

Similarly, following Cramer's rule, the roots of the $N(s)$ function can also be the *eigen* values of a matrix. HSPICE RF supports two different kinds of singular values deposition (*svd*):

- `OPTION=HQR` requires that the G matrix is non-singular.
- `OPTION=SVD` requires only that G and C are real matrices.

The later *SVD* requires twice as much memory as the first one, and is approximately three times slower.

Muller Method

You can apply the Muller method if the network contains frequency-dependent elements (such as S - or W -elements).

The Muller method approximates the polynomial, using a quadratic equation that fits through three points in the vicinity of a root. To obtain successive iterations toward a particular root, HSPICE finds the nearer root of a quadratic, whose curve passes through the last three points.

Selection of the three initial points affects both the convergence of the process, and the accuracy of the roots obtained:

1. If the poles or zeros occupy a wide frequency range, then choose ($X0R$, $X0I$) close to the origin, to find poles or zeros at the zero frequency first.
2. Find the remaining poles or zeros, in increasing order.

The ($X1R$, $X1I$) and ($X2R$, $X2I$) values can be orders of magnitude larger than ($X0R$, $X0I$). If any poles or zeros occur at high frequencies, adjust $X1I$ and $X2I$ accordingly.

Pole/zero analysis results use the circuit's DC operating point, so the operating point solution must be accurate. Use the `.NODESET` statement (not `.IC`) for initialization, to avoid DC convergence problems.

For the syntax, see [.PZ](#) in the *HSPICE Reference Manual: Commands and Control Options*.

How HSPICE Calculates Poles and Zeros

HSPICE calculates poles and zeros independently from the denominator and numerator polynomials of the transfer function respectively. It is possible that there exist common multipliers, such as the pole and zero appearing at exactly

the same location. In this case, the pole and zero factor out. HSPICE only presents the roots of the two polynomials without doing the factorization.

For example, if you use the netlist `fkerwin.sp` from the demo folder, from the transfer function there should be two poles and zeros:

```
*      = (s**2 + 2) / (s**2 + 0.1*s + 1)
*      poles --- (-0.05004,+0.9987) , (-0.05004,-0.9987)
*      zeros --- ( 0.0      ,+1.4142) , ( 0.0      , -1.4142)
```

But HSPICE reports three poles and zeros:

```
poles (rad/sec)                poles (hertz)
*****
 real          imag          real          imag
zeros (rad/sec)                zeros (hertz)
-50.0394m      998.7214m      -7.9640m      158.9515m
-50.0394m      -998.7214m     -7.9640m      -158.9515m
*****
 real          imag          real          imag
 0.            -1.4142         0.            -225.0812m
 0.            1.4142 0.         0.            225.0812m
-1.4142        0.            -225.0812m     0.
-1.4142        0.            -225.0812m     0.
```

The transfer function cancels the common pole and zero at -1.4142 0. Since HSPICE solves the denominator and numerator separately, the results reports these separately.

Pole/Zero Analysis Examples

The following are examples of different types of pole/zero analysis.

- [Example 1 – Low-Pass Filter](#)
- [Example 2 – Kerwin’s Circuit](#)
- [Example 3 – High-Pass Butterworth Filter](#)
- [Example 4 – CMOS Differential Amplifier](#)
- [Example 5 – Simple Amplifier](#)
- [Example 6— Active Low-Pass Filter](#)

Example 1 – Low-Pass Filter

This example uses HSPICE demonstration netlist `flp5th.sp`, which is available in directory `$installdir/demo/hspice/filters`:

```
file:flp5th.sp lowpass 5th order filter
*****
* reference: gabor c. temes and sanjit k. mitra, "modern filter
theory
* and design", j. wiley, 1973, page 74.
*  $t = v(3) / iin$ 
*  $= 0.113 * (s^{**2} + 1.6543) * (s^{**2} + 0.2632) /$ 
*  $(s^{**5} + 0.9206 * s^{**4} + 1.26123 * s^{**3} + 0.74556 * s^{**2}$ 
*  $+ 0.2705 * s + 0.09836)$ 
*****
* pole zero, ac(.001hz-10hz) analysis
*
.options post
.pz v(3) iin
.ac dec 100 .001hz 10hz
.probe ac vdb(3) vp(3)
*
iin 1 0 1.00 ac 1
r1 1 0 1.0
c3 1 0 1.52
c4 2 0 1.50
c5 3 0 0.83
c1 1 2 0.93
l1 1 2 0.65
c2 2 3 3.80
l2 2 3 1.00
r2 3 0 1.0
.end
```

The following is an equivalent example in HSPICE RF:

Chapter 17: Pole-Zero Analysis

Pole/Zero Analysis Examples

```

* HSPICE RF example:
5TH-ORDER LOW_PASS FILTER
*
.OPTION POST
.PZ I(R2) IN
.AC DEC 100 .001HZ 10HZ
IN 0 1 1.00 AC 1
R1 1 0 1.0
C3 1 0 1.52
C4 2 0 1.50
C5 3 0 0.83
C1 1 2 0.93
L1 1 2 0.65
C2 2 3 3.80
L2 2 3 1.00
R2 3 0 1.00
.END

```

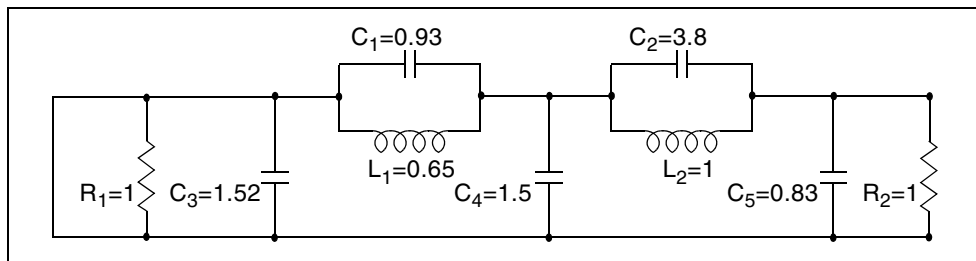


Figure 82 Low-Pass Prototype Filter

Table 64 shows the magnitude and phase variation of the current output, resulting from AC analysis. These results are consistent with pole/zero analysis. The pole/zero unit is radians per second, or hertz. The X-axis unit, in the plot, is in hertz.

Table 64 Pole/Zero Analysis Results for Low-Pass Filter

Poles (rad/sec)		Poles (Hertz)	
Real	Imaginary	Real	Imaginary
-6.948473e-02	-4.671778e-01	-1.105884e-02	-7.435365e-02
-6.948473e-02	4.671778e-01	-1.105884e-02	7.435365e-02
-1.182742e-01	-8.914907e-01	-1.882392e-02	-1.418852e-01

Table 64 Pole/Zero Analysis Results for Low-Pass Filter (Continued)

Poles (rad/sec)		Poles (Hertz)	
Real	Imaginary	Real	Imaginary
-1.182742e-01	8.914907e-01	-1.882392e-02	1.418852e-01
-5.450890e-01	0.000000e+00	-8.675361e-02	0.000000e+00
0.000000e+00	-1.286180e+00	0.000000e+00	-2.047019e-01
0.000000e+00	-5.129892e-01	0.000000e+00	-8.164476e-02
0.000000e+00	5.129892e-01	0.000000e+00	8.164476e-02
0.000000e+00	1.286180e+00	0.000000e+00	2.047019e-01

Constant Factor = 1.129524e-01

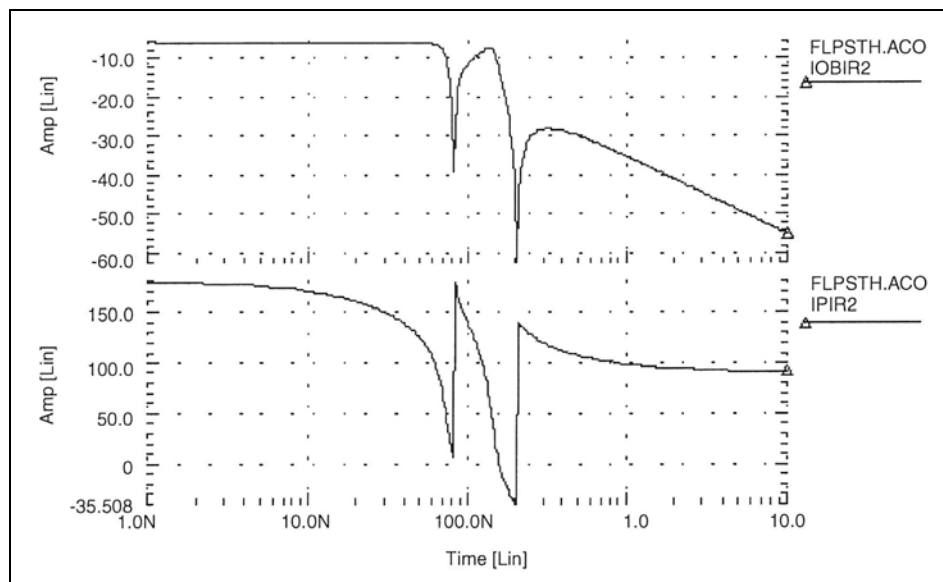


Figure 83 Fifth-Order Low-Pass Filter Response

Example 2 – Kerwin’s Circuit

This example is an HSPICE input file for pole/zero analysis of Kerwin’s circuit, which is in the following directory: `$installdir/demo/hspice/filters/fkerwin.sp`.

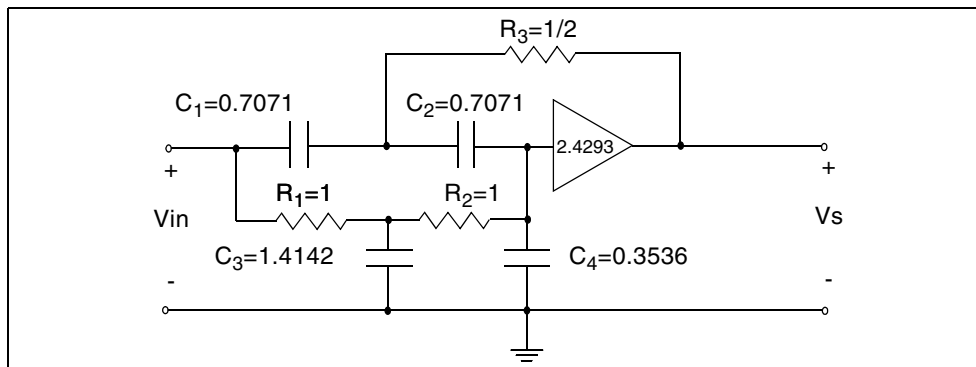


Figure 84 Design Example for Kerwin’s Circuit

Table 65 on page 582 lists analysis results.

Table 65 Pole/Zero Analysis Results for Kerwin’s Circuit

Poles (rad/sec)		Poles (Hz)	
Real	Imaginary	Real	Imaginary
-5.003939e-02	9.987214e-01	-7.964016e-03	1.589515e-01
-5.003939e-02	-9.987214e-01	-7.964016e-03	-1.589515e-01
-1.414227e+00	0.000000e+00	-2.250812e-01	0.000000e+00
0.000000e+00	-1.414227e+00	0.000000e+00	-2.250812e-01
0.000000e+00	1.414227e+00	0.000000e+00	2.250812e-01
-1.414227e+00	0.000000e+00	-2.250812e-01	0.000000e+00
Constant Factor = 1.214564e+00			

Example 3 – High-Pass Butterworth Filter

This example is a HSPICE input file, for pole/zero analysis of a fourth-order high-pass Butterworth filter. This file can be found at `$installdir/demo/hspice/filters/fhp4th.sp`. Table 66 on page 583 shows the analysis results.

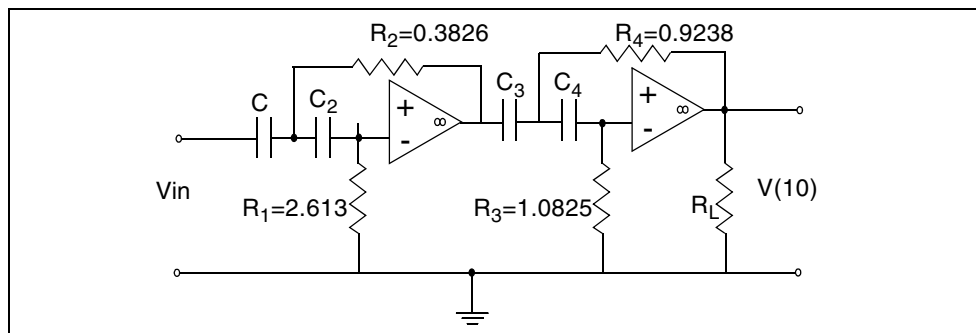


Figure 85 Fourth-Order High-Pass Butterworth Filter

Table 66 Pole/Zero Analysis Results for High-Pass Butterworth Filter

Poles (rad/sec)		Poles (Hz)	
Real	Imaginary	Real	Imaginary
-3.827019e-01	-9.240160e-01	-6.090889e-02	1.470617e-01
-3.827019e-01	9.240160e-01	-6.090890e-02	-1.470617e-01
-9.237875e-01	3.828878e-01	-1.470254e-01	6.093849e-02
-9.237875e-01	-3.828878e-01	-1.470254e-01	-6.093849e-02
0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
Constant Factor = 1.000000e+00			

Example 4 – CMOS Differential Amplifier

This example uses HSPICE demonstration netlist `mcdiff.sp`, which is available in directory `$installdir/demo/hspice/apps`. [Table 67 on page 585](#) shows the analysis results.

```
*file: mcdiff.sp cmos differential amplifier
* analysis : ac(20khz-500mhz), pole-zero
* mos level=5
*
.options scale=1e-6 scaln=1e-6 wl opts post
.pz v(5) vin
vin 7 0 0 ac 1
.ac dec 10 20k 500meg
.probe ac vdb(5) vp(5)
m1 4 0 6 6 mn 100 10 2 2
m2 5 7 6 6 mn 100 10 2 2
m3 4 4 1 1 mp 60 10 1.5 1.5
m4 5 4 1 1 mp 60 10 1.5 1.5
m5 6 3 2 2 mn 50 10 1.0 1.0
vdd 1 0 5
vss 2 0 -5
vgg 3 0 -3
rin 7 0 1
.model mn nmos level=5 vt=1 ub=700 frc=0.05 tox=800 dnb=1.6e16
+ xj=1.2 latd=0.7 cj=0.13 phi=1.2 tcv=0.003
.model mp pmos level=5 vt=-1 ub=245 frc=0.25 tox=800 dnb=1.3e15
+ xj=1.2 latd=0.9 cj=0.09 phi=0.5 tcv=0.002
.end
```

The following is an equivalent example in HSPICE RF:

```
* HSPICE RF example
CMOS DIFFERENTIAL AMPLIFIER
.OPTION PIVOT SCALE=1.e-6 SCALM=1.e-6 HQR
.PZ V(5) VIN
VIN 7 0 0 AC 1
.AC DEC 10 20K 500MEG
M1 4 0 6 6 MN 100 10 2 2
M2 5 7 6 6 MN 100 10 2 2
M3 4 4 1 1 MP 60 10 1.5 1.5
M4 5 4 1 1 MP 60 10 1.5 1.5
M5 6 3 2 2 MN 50 10 1.0 1.0
VDD 1 0 5
VSS 2 0 -5
VGG 3 0 -3
RIN 7 0 1
.MODEL MN NMOS LEVEL=5 VT=1 UB=700 FRC=0.05 DNB=1.6E16
+ XJ=1.2 LATD=0.7 CJ=0.13 PHI=1.2 TCV=0.003 TOX=800
.MODEL MP PMOS LEVEL=5 VT=-1 UB=245 FRC=0.25 TOX=800
+ DNB=1.3E15 XJ=1.2 LATD=0.9 CJ=0.09 PHI=0.5 TCV=0.002
.END
```

Table 67 Pole/Zero Analysis Results for CMOS Differential Amplifier

Poles (rad/sec)		Poles (Hz)	
Real	Imaginary	Real	Imaginary
-1.798766e+06	0.000000e+00	-2.862825e+05	0.000000e+00
-1.126313e+08	-6.822910e+07	-1.792583e+07	-1.085900e+07
-1.126313e+08	6.822910e+07	-1.792583e+07	1.085900e+07
-1.315386e+08	7.679633e+07	-2.093502e+07	1.222251e+07
-1.315386e+08	-7.679633e+07	-2.093502e+07	-1.222251e+07
7.999613e+08	0.000000e+00	1.273178e+08	0.000000e+00
Constant Factor = 3.103553e-01			

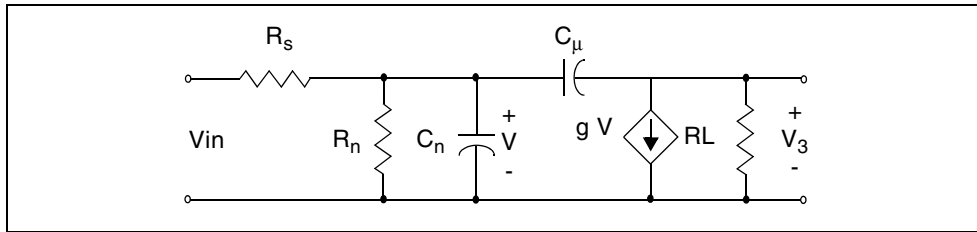


Figure 87 Simple Amplifier

Table 68 Pole/Zero Analysis Results for Amplifier

Poles (rad/sec)		Poles (Hz)	
Real	Imaginary	Real	Imaginary
-1.412555+06	0.000000e+00	-2.248151e+05	0.000000e+00
-1.415874+08	0.000000e+00	-2.253434e+07	0.000000e+00
4.000000e+09	0.000000e+00	6.366198e+08	0.000000e+00
Constant Factor = 1.000000e+06			

Example 6— Active Low-Pass Filter

This example uses demonstration netlist `flp9th.sp`, which is available in directory `$installdir/demo/hspice/filters`. It is for a pole/zero analysis of an active ninth-order low-pass filter by using the ideal operational amplifier element. This example performs an AC analysis. [Table 69 on page 591](#) shows the analysis results.

Chapter 17: Pole-Zero Analysis

Pole/Zero Analysis Examples

```
* file flp9th.sp----9th order low-pass filter
*
* reference: jiri vlach and kishore singhal, 'computer
* methods for circuit analysis and design',
* van nostrand reinhold co., 1983, pages 142
* and 494 to 496.
*
* pole/zero analysis and using vcvs as an ideal op-amp.
* for just pole/zero analysis .ac statement is not required.
vin in 0 ac 1
.ac dec 100 1 100k
.print vm(out) vm(in) vp(out)
.probe ac vdb(out,in) par('db(vm(out)/vm(in))')
.pz v(out) vin
.options post dcstep=1e3
+ x0r=-1.23456e+3 x1r=-1.23456e+2 x2r=1.23456e+3
+ fscal=1e-6 gscal=1e3 cscal=1e9 lscal=1e3
.subckt fdnr 1 r1=2k c1=12n r4=4.5k
r1 1 2 r1
c1 2 3 c1
r2 3 4 3.3k
r3 4 5 3.3k
r4 5 6 r4
c2 6 0 10n
eop1 5 0 2 4 level=1
eop2 3 0 6 4 level=1
.ends
*
rs in 1 5.4779k
r12 1 2 4.44k
r23 2 3 3.2201k
r34 3 4 3.63678k
r45 4 out 1.2201k
c5 out 0 10n
x1 1 fdnr r1=2.0076k c1=12n r4=4.5898k
x2 2 fdnr r1=5.9999k c1=6.8n r4=4.25725k
x3 3 fdnr r1=5.88327k c1=4.7n r4=5.62599k
x4 4 fdnr r1=1.0301k c1=6.8n r4=5.808498k
.end
```

The following is an equivalent example in HSPICE RF:

```

* HSPICE RF example
VIN IN 0 AC 1

.PZ V(OUT) VIN
.AC DEC 50 .1K 100K
.OPTION PLST DCSTEP=1E3 XOR=-1.23456E+3 X1R=-1.23456E+2
+ X2R=1.23456E+3 FSCAL=1E-6 GSCAL=1E3 CSCAL=1E9 LSCAL=1E3
.PRINT AC VDB(OUT)

.SUBCKT OPAMP IN+ IN- OUT GM1=1 RI=1K CI=26.6U GM2=1.33333 RL=75
R11 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

```

Chapter 17: Pole-Zero Analysis
Pole/Zero Analysis Examples

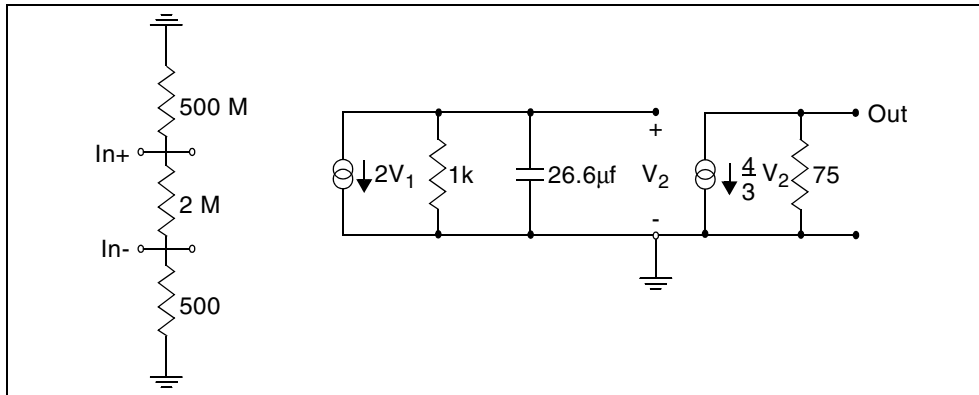


Figure 88 Linear Model of the 741C Operational Amplifier

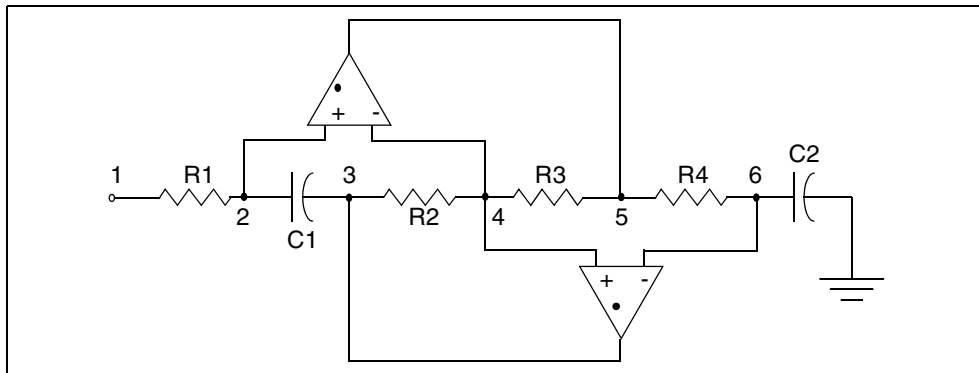


Figure 89 FDNR Subcircuit

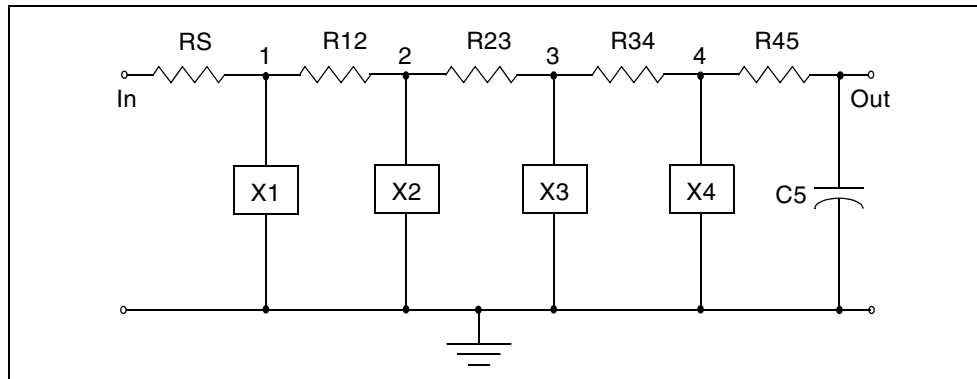


Figure 90 Active Realization of the Low-Pass Filter

Table 69 Pole/Zero Analysis Results for the Active Low-Pass Filter

Poles (rad/sec)		Poles (Hz)	
Real	Imaginary	Real	Imaginary
-4.505616e+02	-2.210451e+04	-7.170911e+01	-3.518042e+03
-4.505616e+02	2.210451e+04	-7.170911e+01	3.518042e+03
-1.835284e+03	2.148369e+04	-2.920944e+02	3.419236e+03
-1.835284e+03	-2.148369e+04	-2.920944e+02	-3.419236e+03
-4.580172e+03	1.944579e+04	-7.289571e+02	3.094894e+03
-4.580172e+03	-1.944579e+04	-7.289571e+02	-3.094894e+03
-9.701962e+03	1.304893e+04	-1.544115e+03	2.076802e+03
-9.701962e+03	-1.304893e+04	-1.544115e+03	-2.076802e+03
-1.353908e+04	0.000000e+00	-2.154811e+03	0.000000e+00
-3.668995e+06	-3.669793e+06	-5.839386e+05	-5.840657e+05
-3.668995e+06	3.669793e+06	-5.839386e+05	5.840657e+05
-3.676439e+06	-3.676184e+06	-5.851234e+05	-5.850828e+05

Chapter 17: Pole-Zero Analysis
Pole/Zero Analysis Examples

Table 69 Pole/Zero Analysis Results for the Active Low-Pass Filter (Continued)

Poles (rad/sec)		Poles (Hz)	
Real	Imaginary	Real	Imaginary
-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
-3.220467e-02	-2.516970e+04	-5.125532e-03	-4.005882e+03
-3.220467e-02	2.516970e+04	-5.125533e-03	4.005882e+03
2.524420e-01	-2.383956e+04	4.017739e-02	-3.794184e+03
2.524420e-01	2.383956e+04	4.017739e-02	3.794184e+03
1.637164e+00	2.981593e+04	2.605627e-01	4.745353e+03
1.637164e+00	-2.981593e+04	2.605627e-01	-4.745353e+03
4.888484e+00	4.852376e+04	7.780265e-01	7.722796e+03
4.888484e+00	-4.852376e+04	7.780265e-01	-7.722796e+03
-3.641366e+06	-3.642634e+06	-5.795413e+05	-5.797432e+05
-3.641366e+06	3.642634e+06	-5.795413e+05	5.797432e+05
-3.649508e+06	-3.649610e+06	-5.808372e+05	-5.808535e+05
-3.649508e+06	3.649610e+06	-5.808372e+05	5.808535e+05
-3.683700e+06	3.683412e+06	-5.862790e+05	5.862333e+05
-3.683700e+06	-3.683412e+06	-5.862790e+05	-5.862333e+05
-3.693882e+06	3.693739e+06	5.878995e+05	5.878768e+05

Table 69 Pole/Zero Analysis Results for the Active Low-Pass Filter (Continued)

Poles (rad/sec)		Poles (Hz)	
Real	Imaginary	Real	Imaginary
-3.693882e+06	-3.693739e+06	-5.878995e+05	-5.878768e+05
Constant Factor = 4.451586e+02			

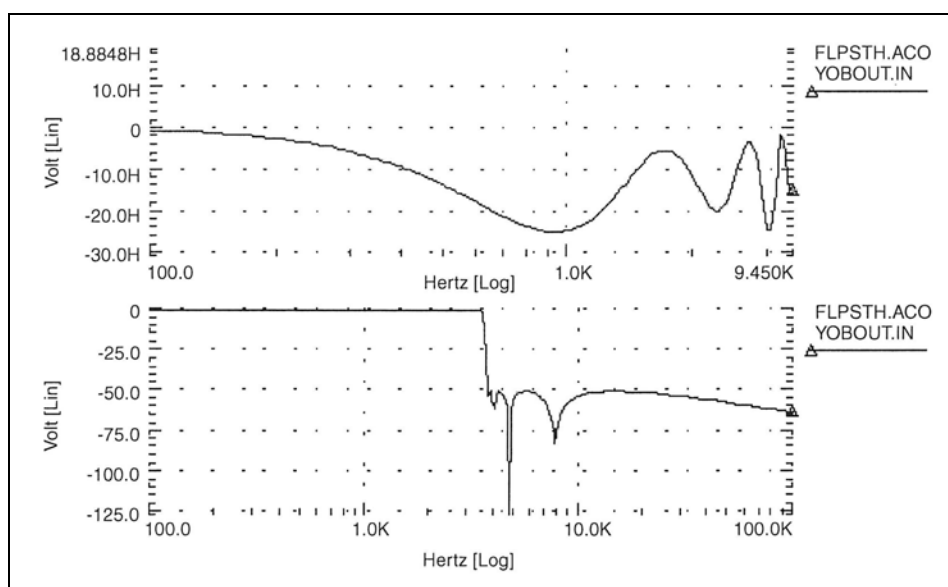


Figure 91 9th Order Low-Pass Filter Response

The graph in [Figure 91](#) shows overall response of the low-pass filter.

References

- [1] Desoer, Charles A. and Kuh, Ernest S. *Basic Circuit Theory*. New York: McGraw-Hill. 1969. Chapter 15.
- [2] Van Valkenburg, M. E. *Network Analysis*. Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1974, Chapters 10 & 13.
- [3] R.H. Canon, Jr. *Dynamics of Physical Systems*. New York: McGraw-Hill, 1967. This text describes electrical, mechanical, pneumatic, hydraulic, and mixed systems.
- [4] B.C. Kuo. *Automatic Control Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, 1975. This source discusses control system design, and provides background material about physical modeling.
- [5] L.T. Pillage, and R.A. Rohrer. "Asymptotic Waveform Evaluation for Timing Analysis", *IEEE Trans CAD*. Apr. 1990, pp. 352 - 366. This paper is a good references about interconnect transfer function modeling, and discusses extracting transfer functions for timing analysis.
- [6] S. Lin, and E.S. Kuh. "Transient Simulation of Lossy Interconnects Based on the Recursive Convolution Formulation", *IEEE Trans CAS* Nov. 1992, pp. 879 - 892. This paper is another source for how to model interconnect transfer functions.
- [7] Muller, D. E., *A Method for Solving Algebraic Equations Using a Computer, Mathematical Tables, and Other Aids to Computation (MTAC)*. 1956, Vol. 10,. pp. 208-215.
- [8] Temes, Gabor C. and Mitra, Sanjit K. *Modern Filter Theory And Design*. J. Wiley, 1973, page 74.
- [9] Temes, Gabor C. and Lapatra, Jack W. *Circuit Synthesis And Design* McGraw-Hill. 1977, page 301, example 7-6.
- [10] Temes, Gabor C. and Mitra, Sanjit K., *Modern Filter Theory And Design*. J. Wiley, 1973, page 348, example 8-3.
- [11] Desoer, Charles A. and Kuh, Ernest S. *Basic Circuit Theory*. McGraw-Hill, 1969, page 613, example 3.
- [12] Vlach, Jiri and Singhal, Kishore. *Computer Methods For Circuit Analysis and Design*. Van Nostrand Reinhold Co., 1983, pages 142, 494-496.

Performing Digital Cell Characterization

Describes how to characterize cells in data-driven analysis and shows some typical data sheet parameters.

HSPICE ships numerous examples for your use; see [Cell Characterization Examples](#) for paths to demo files.

Most ASIC vendors use the basic capabilities of the `.MEASURE` statement in Synopsys HSPICE to characterize standard cell libraries, and to prepare data sheets.

HSPICE stores input sweep parameters and measure output parameter, in measure output data files (*design.mt0*, *design.sw0*, and *design.ac0*). These files store multiple sweep data. You can use Custom WaveView to plot this data; for example, to generate fanout plots of delay versus load. You can also use the slope and intercept of the loading curves to calibrate VHDL, Verilog, Lsim, and Synopsys models. The following sections discuss these topics:

- [Performing Basic Cell Measurements](#)
- [Performing Advanced Cell Characterization](#)
- [Cell Examples](#)

This chapter shows you some typical data sheet parameters. A series of typical data sheet examples, demonstrates the flexibility of the `.MEASURE` statement.

This chapter also shows you how to characterize cells in data-driven analysis. Data-driven analysis automates cell characterization, including calculating the delay coefficient for the timing-simulator polynomial. You can simultaneously vary an unlimited number of parameters, or the number of analyses to perform. Cell characterization uses an ASCII file format for automated parameter input to HSPICE.

Performing Basic Cell Measurements

The following sections describe how to perform basic cell measurements.

- [Rise, Fall, and Delay Calculations](#)
- [Delay versus Fanout](#)
- [Pin Capacitance Measurement](#)
- [Op-amp Characterization of LM124](#)

Rise, Fall, and Delay Calculations

The following example does the following:

- Uses the MAX function to calculate v_{max} , over the time region of interest.
- Uses the MIN function to calculate v_{min} .
- Uses the measured parameters in subsequent calculations, for accurate 10 percent and 90 percent points, when determining the rise and fall time.
RISE=1 is relative to the time window that the TDval delay forms.
- Uses a fixed value for the measure threshold, to calculate the Tdelay delay.

```
.MEAS TRAN vmax MAX V(out) FROM=TDval TO=Tstop  
.MEAS TRAN vmin MIN V(out) FROM=TDval TO=Tstop  
.MEAS TRAN Trise TRIG V(out) val='vmin+0.1*vmax'  
+ TD=TDval RISE=1 TARG V(out) val='0.9*vmax' RISE=1  
.MEAS TRAN Tfall TRIG V(out) val='0.9*vmax' TD=TDval  
+ FALL=2 TARG V(out) val='vmin+0.1*vmax' FALL=2  
.MEAS TRAN Tdelay TRIG V(in) val=2.5 TD=TDval FALL=1  
+ TARG V(out) val=2.5 FALL=2
```

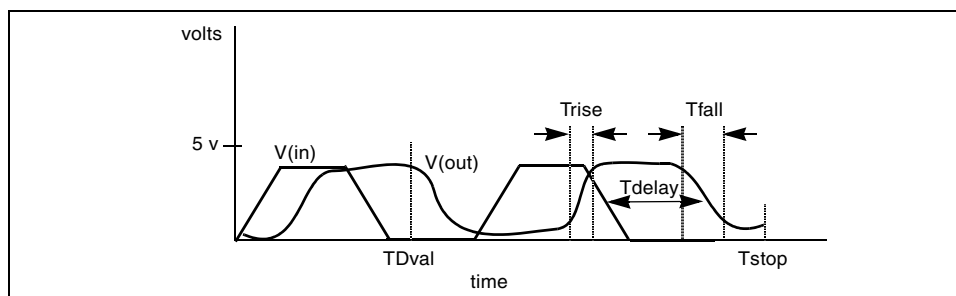


Figure 92 Rise, Fall, and Delay Time Demonstration

Ripple calculation performs the following:

- Delimits the wave at the 50 percent of VCC points
- Finds the T_{mid} midpoint
- Defines a bounded region by finding the pedestal voltage (V_{mid}) and then finding the first time that the signal crossed this value, T_{from}
- Measures the ripple in the defined region using the peak-to-peak (PP) measure function from T_{from} to T_{mid}

The following is an example:

```
.MEAS TRAN Th1 WHEN V(out)='0.5*vcc' CROSS=1
.MEAS TRAN Th2 WHEN V(out)='0.5*vcc' CROSS=2
.MEAS TRAN Tmid PARAM='(Th1+Th2)/2'
.MEAS TRAN Vmid FIND V(out) AT='Tmid'
.MEAS TRAN Tfrom WHEN V(out)='Vmid' RISE=1
.MEAS TRAN Ripple PP V(out) FROM='Tfrom' TO='Tmid'
```

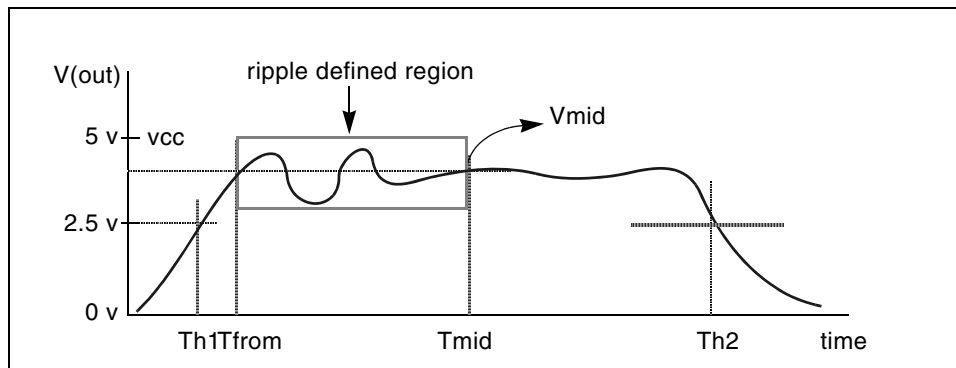


Figure 93 Waveform to Demonstrate Ripple Calculation

This file sweeps the sigma of the model parameter distribution, while it examines the delay. It shows you the delay derating curve, for the worst cases in the model. This example uses demonstration netlist `sigma.sp`, which is available in directory `$installdir/demo/hspice/cchar`.

Chapter 18: Performing Digital Cell Characterization

Performing Basic Cell Measurements

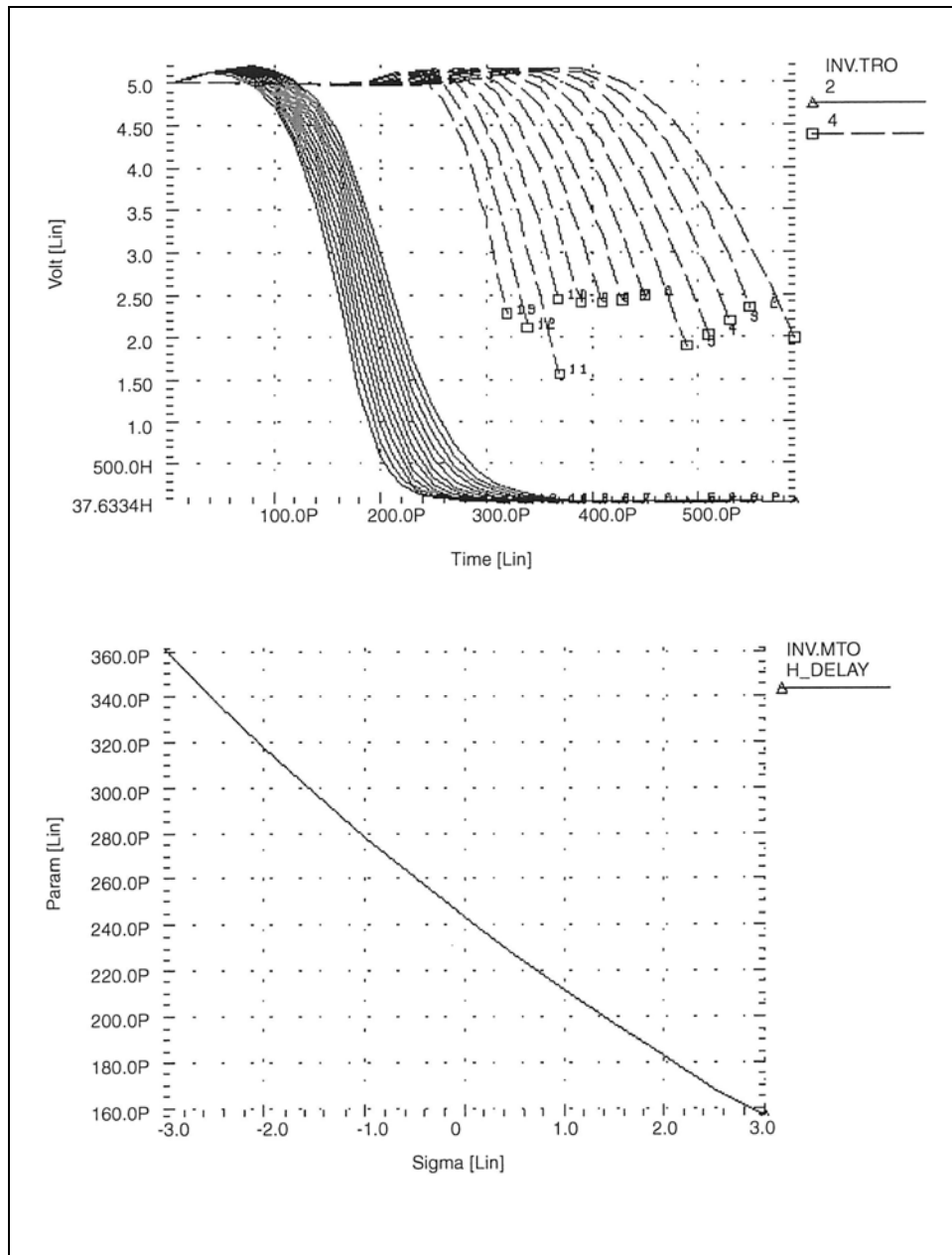


Figure 94 Inverter Pair Transfer Curves and Sigma Sweep vs. Delay

Delay versus Fanout

The example sweeps the subcircuit multiplier to quickly generate five load curves. To obtain more accurate results, buffer the input source with one stage.

For each second-sweep variable (`m_delay` and `rms_power`), the example calculates:

- mean
- variance
- sigma
- average deviance

This example uses the demonstration netlist `load1.sp`, which is available in directory `$installdir/demo/hspice/cchar`.

This example outputs the following results:

```
meas_variable = m_delay
mean = 273.8560p varian = 1.968e-20
sigma = 140.2711p avgdev = 106.5685p
```

```
meas_variable = rms_power
mean = 5.2544m varian = 8.7044u
sigma = 2.9503m avgdev = 2.2945m
```

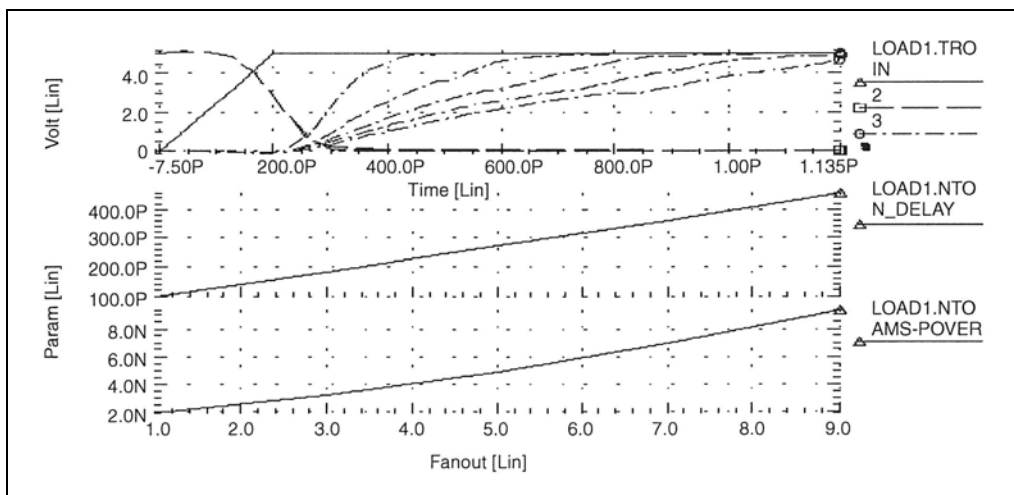


Figure 95 Inverter Delay and Power, versus Fanout

Pin Capacitance Measurement

This example does the following:

- Shows the effect of dynamic capacitance, at the switch point.
- Sweeps the DC input voltage (`pdcin`) to the inverter.
- Performs an AC analysis, at each 0.1 V increment.
- Calculates the `incap` measure parameter from the imaginary current through the voltage source at 10 kHz in the AC curve (not shown).

The peak capacitance (at the switch point) occurs when the voltage at the output side changes, in the direction opposite the input side of the Miller capacitor. This adds the Miller capacitance, times the inverter gain, to the effective capacitance.

```
mp out in 1 1 mp w=10u l=3u
mn out in 0 0 mn w=5u l=3u
vin in 0 DC= pdcin AC 1 0
.ac lin 2 10k 100k sweep pdcin 0 5 .1
.measure ac incap find par( '-1 * ii(vin)/
+ (hertz*twopi)' ) AT=10000hertz
```

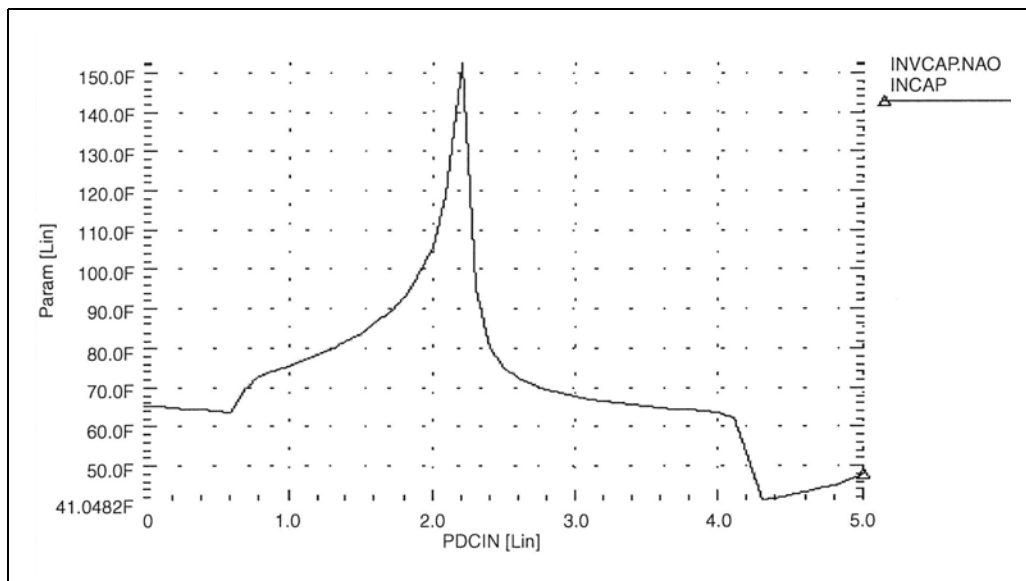


Figure 96 Graph of Pin Capacitance versus Inverter Input Voltage

Op-amp Characterization of LM124

This example analyzes op-amps. This example uses:

- .MEASURE statements to present a very complete data sheet.
- Four .MEASURE statements, to reference the `out0` output node of an op-amp circuit. These statements use output variable operators for parameters:
 - decibels `vdb(out0)`
 - voltage magnitude `vm(out0)`
 - phase `vp(out0)`

This example uses the demonstration file `alm124.sp`, in `$installdir/demo/hspice/apps`.

This example outputs the following results:

```
unitfreq = 9.0786E+05 targ= 9.0786E+05 trig= 1.0000E+00  
phasemargin = 6.6403E+01
```

```
gain(db) = 9.9663E+01 at= 1.0000E+00 from= 1.0000E+00  
+ to= 1.0000E+07
```

```
gain(mag)= 9.6192E+04 at= 1.0000E+00 from= 1.0000E+00  
+ to= 1.0000E+07
```

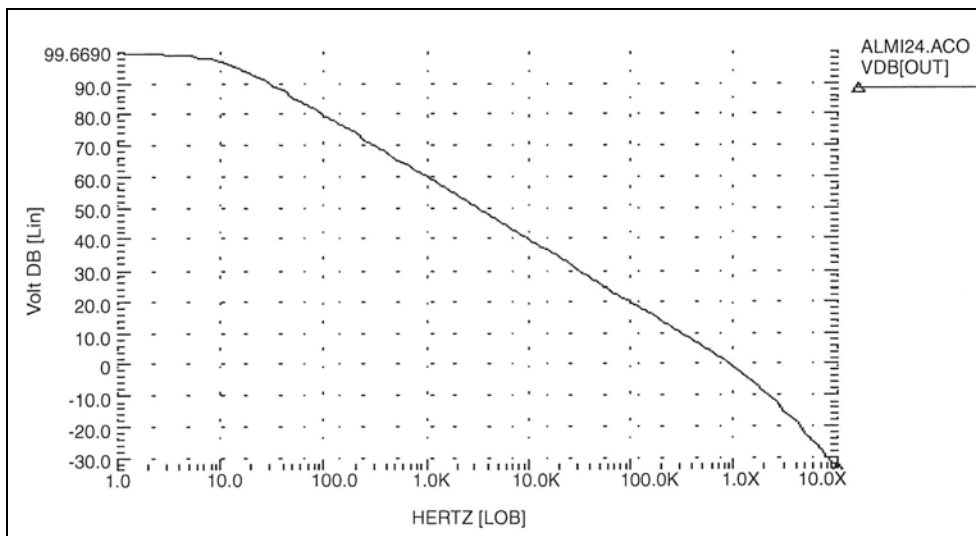


Figure 97 Magnitude Plot of Op-Amp Gain

Performing Advanced Cell Characterization

This section provides example input files, which characterize cells for an inverter, based on 3-micron MOSFET technology. HSPICE finds the best, worst, and typical cases for different fanouts. Use this library data for digital-based simulators, such as those used to simulate gate arrays and standard cells.

The example uses the demonstration file `cellchar.sp`, in `$installdir/demo/hspice/apps`. It demonstrates how to use the following to characterize a CMOS inverter:

- `.MEASURE` statement
- `.DATA` statement
- `AUTOSTOP` option
- `SUBCKT` definition
- `SUBCKT` call
- Models

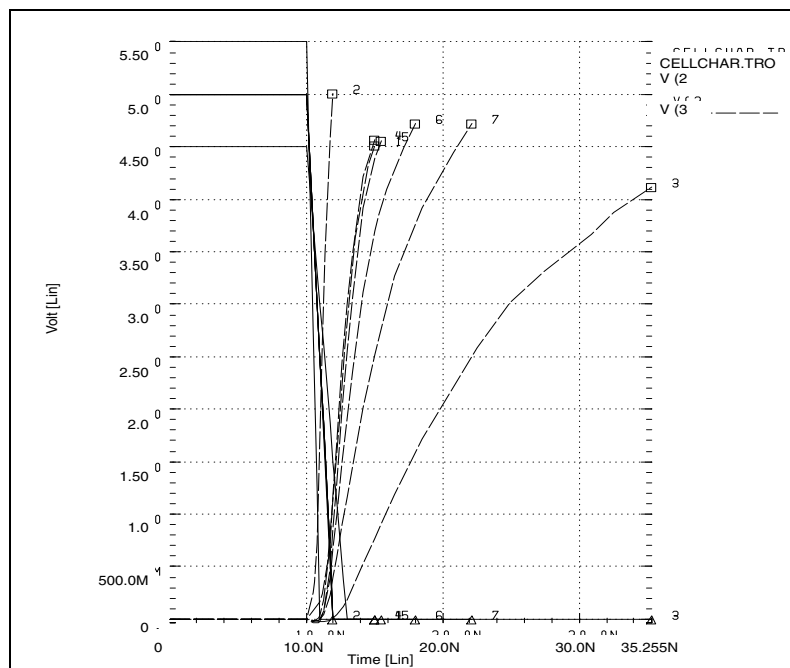


Figure 98 Plotting the Simulation Outputs

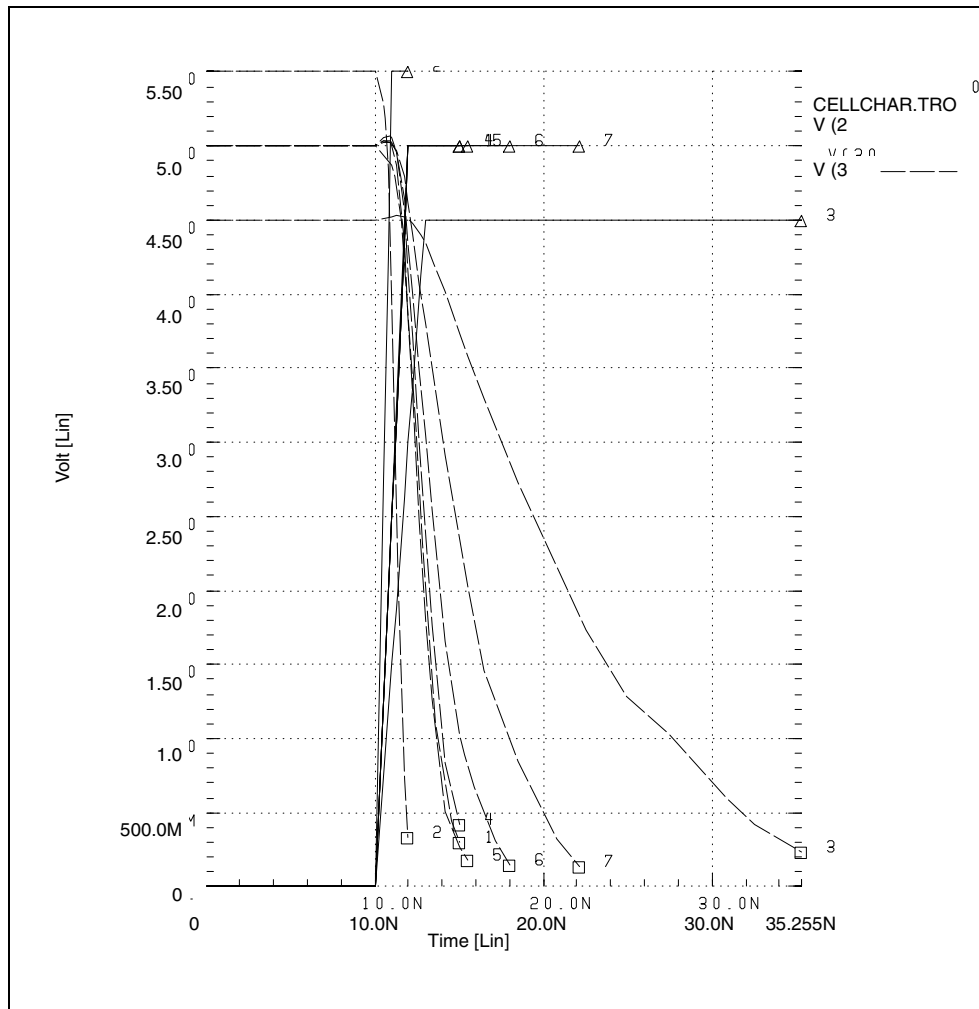


Figure 99 Verifying the Measure Statement Results by the Plots

Cell Examples

Figure 100 and Figure 101 are identical, except that their input signals are complementary.

- The circuit in Figure 100 calculates the rise time and the low-to-high propagation delay time.
- The circuit in Figure 101 calculates the fall time and the high-to-low propagation delay time.

Chapter 18: Performing Digital Cell Characterization

Performing Advanced Cell Characterization

If you use only one circuit, CPU time increases because analysis time increases when HSPICE calculates both rise and fall times.

The XOUTL or XOUTH subcircuit represents the fanout of the cell (inverter). To modify fanout, specify different multipliers (m) in the subcircuit calls.

You can also specify local and global temperatures. This example characterizes the cell at a global temperature of 27, but the temperature of the M1 and M2 devices is $(27+DTEMP)$. The .DATA statement specifies the DTEMP value.

The example uses a transient parameterized sweep, with .DATA and .MEASURE statements, to determine the inverter timing, for best, typical, and worst cases.

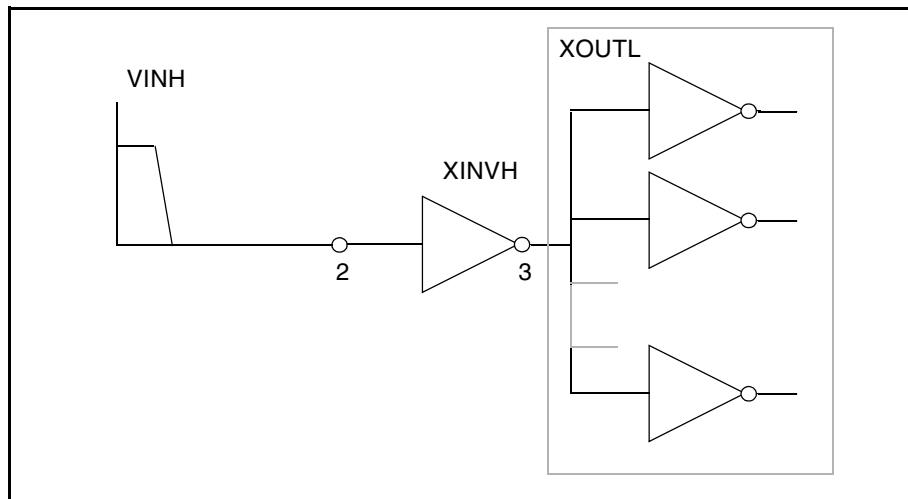


Figure 100 Cell Characterization Circuit 1

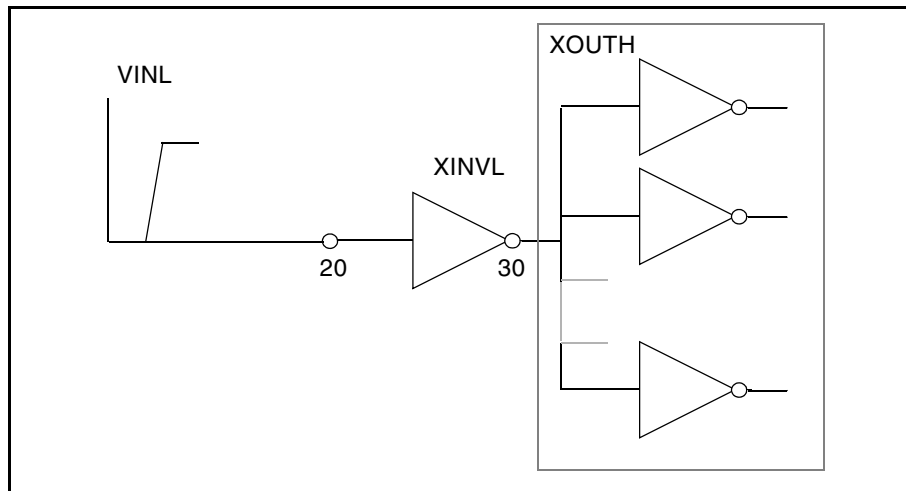


Figure 101 Cell Characterization Circuit 2

This example varies the following parameters:

- power supply
- input rise and fall time
- fanout
- MOSFET temperature
- n-channel and p-channel threshold
- drawn width and length of the MOSFET

Use the `.MEASURE` statement to specify a parameter to measure.

Use the `AUTOSTOP` option, to speed simulation time. The `AUTOSTOP` option terminates the transient sweep, although it has not completely swept the specified transient sweep range.

The `.MEASURE` statement uses quoted string parameter variables to measure the rise time, fall time, and propagation delays.

Note: Do not use character strings as parameter values in HSPICE RF.

Rise time starts when the voltage at node 3 (the output of the inverter) is equal to $0.1 \cdot VDD$ (that is, $V(3) = 0.1VDD$). Rise time ends when the voltage at node 3 is equal to $0.9 \cdot VDD$ (that is, $V(3) = 0.9VDD$).

For more accurate results, start the `.MEASURE` calculation after either:

Chapter 18: Performing Digital Cell Characterization

Performing Advanced Cell Characterization

- A time delay, or
- A simulation cycle, specifying delay time in the `.MEASURE` statement, or
- An input pulse statement.

The following example features:

- `AUTOSTOP` option and `.MEASURE` statements.
- Mean, variance, sigma, and avgdev calculations.
- Circuit and element temperature.
- Algebraic equation handling.
- `PAR()` as an output variable, in the `.MEASURE` statement.
- Subcircuit parameter-passing, and subcircuit multiplier.

- `.DATA` statement.

MOSFET Model Reliability Analysis (MOSRA)

Describes the procedures for HSPICE MOSFET reliability analysis (MOSRA).

The following sections cover the these topics:

- [MOSRA Overview](#)
- [MOSRA Commands and Control Options](#)
- [Level 1 MOSRA BTI and HCI Model Parameters](#)

MOSRA Overview

As the industry scales down CMOS technology, reliability requirements to maintain the long-term device become both more challenging and more important. Two of the most critical reliability issues, the hot carrier injection (HCI) and the bias temperature instability (BTI) can change the characteristics of MOS devices. HSPICE reliability analysis allows circuit designers to predict the reliability of their design to allow enough margin for their circuits to function correctly over their lifetime.

A unified custom reliability-modeling API is available or custom reliability model development. Contact your Synopsys technical support team for more information about the MOSRA API.

HSPICE MOSRA analysis currently supports Level 49, Level 53, Level 54, Level 57, Level 66, Level 69, Level 70, Level 71, and Level 73, and external CMI MOSFET models.

The following sections discuss these topics:

- [Reliability Analysis Use Model](#)
- [Example Setup](#)

Reliability Analysis Use Model

HSPICE reliability analysis (or HCI and BTI analysis), is a two-phase simulation: the fresh simulation phase and the post-stress simulation phase. The two-phase simulation can run separately or together.

- Fresh simulation phase: HSPICE computes the electron age/stress of selected MOS transistors in the circuit based on circuit behavior and on the HSPICE built-in stress model including HCI and/or BTI effect.
- Post-stress simulation phase: HSPICE simulates the degradation effect on circuit performance, based on the stress information that the fresh simulation phase produces.

Figure 102 presents the HSPICE reliability analysis process.

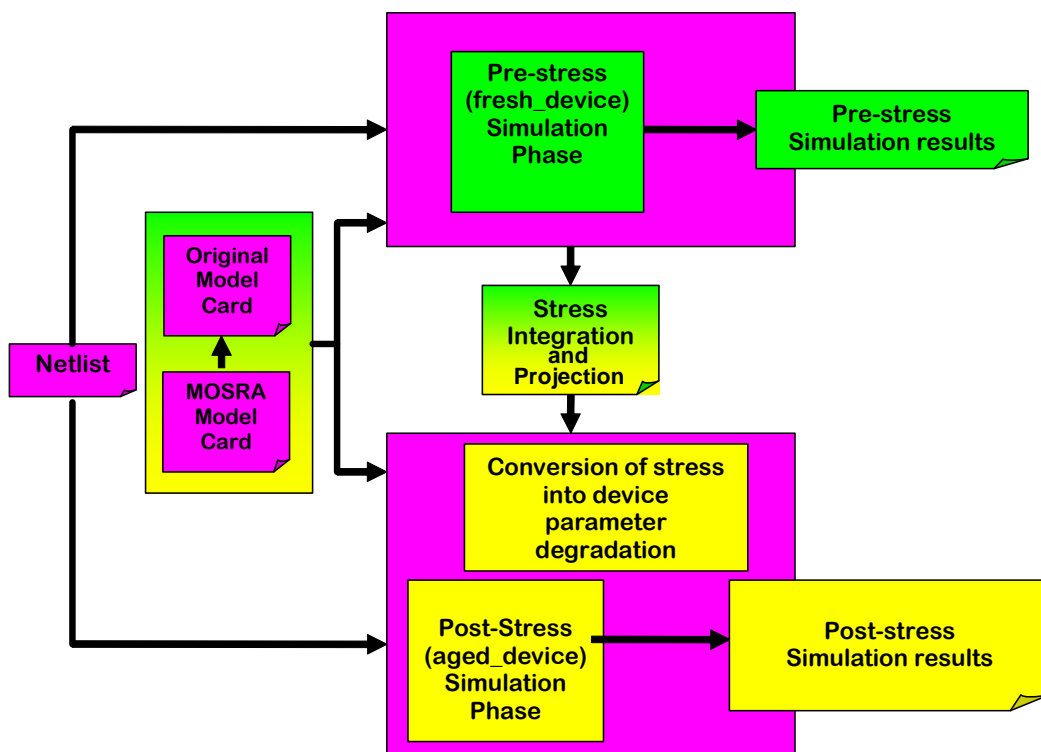


Figure 102 HSPICE Reliability Simulation Flow

Example Setup

The following example file demonstrates how to set up a HSPICE reliability analysis.

```
* MOSRA TEST
vdd 1 0 2
mp1 3 2 1 1 p1 l=0.1u w=10u ad=5p pd=6u as=5p ps=6u
mn1 3 2 0 0 n1 l=0.1u w=5u ad=5p pd=6u as=5p ps=6u
mp2 4 3 1 1 p1 l=0.1u w=10u ad=5p pd=6u as=5p ps=6u
mn2 4 3 0 0 n1 l=0.1u w=5u ad=5p pd=6u as=5p ps=6u
mp3 2 4 1 1 p1 l=0.1u w=10u ad=5p pd=6u as=5p ps=6u
mn3 2 4 0 0 n1 l=0.1u w=5u ad=5p pd=6u as=5p ps=6u
c1 2 0 .1p

.model p1 pmos level=54 version=4.5
.model n1 nmos level=54 version=4.5

.model p1_ra mosra level=1
+tit0 = 5e-8 titfd = 7.5e-10 tittd = 1.45e-20
+tn = 0.25

.appendmodel p1_ra mosra p1 pmos
.mosra reltotaltime=1e8

.ic v(2)=2
.tran .1ps 5ns
.options post
.end
```

MOSRA Commands and Control Options

The following sections discuss these topics:

- [.MOSRA](#)
- [Getting Measurements in a MOSRA Analysis](#)
- [.MOSRAPRINT](#)
- [.MOSRA_SUBCKT_PIN_VOLT](#)
- [.MODEL](#)
- [.APPENDMODEL](#)
- [Simulation Output Files](#)

- [RADEG Output Sorting \(.OPTION MOSRASORT\)](#)
- [CSV Format Degradation Information \(.OPTION RADEGOUTPUT\)](#)

.MOSRA

Starts HSPICE HCI and/or BTI reliability analysis.

Syntax

```
.MOSRA RelTotalTime=time_value
+ [RelStartTime=time_value] [DEC=value] [LIN=value]
+ [RelStep=time_value] [RelMode=1|2|3] SimMode=[0|1|2|3]
+ [AgingStart=time_value] [AgingStop=time_value]
+ [AgingPeriod=time_value] [AgingWidth=time_value]
+ [AgingInst="inst_name"]
+ [Integmod=0|1|2] [Xpolatemod=0|1|2]
+ [Tsample1=value] [Tsample2=value]
+ [Agethreshold=value] [DegradationTime=value]
+ [MosraLlife=degradation_type_keyword] [DegF=value]
+ [DegFN=value] [DegFP=value]
+ [Frequency=value]
```

Argument	Description
RelTotalTime	Final reliability test time to use in post-stress simulation phase. Required argument where <i>time_value</i> can be in units of: <ul style="list-style-type: none"> ▪ sec (default with no unit entry required) ▪ min ▪ hr ▪ day ▪ yr
RelStartTime	Time point of the first post-stress simulation. Default is 0.
DEC	Specifies number of post-stress time points simulated per decade.
LIN	Linear post-stress time points from RelStartTme to RelTotalTime.
RelStep	Post-stress simulation phase on time= RelStep, 2* RelStep, 3* RelStep, ... until it achieves the RelTotalTime; the default is equal to RelTotalTime. If you set values for DEC or LIN HSPICE ignores them.

Argument	Description
RelMode	<p>HSPICE reliability model mode selects whether a simulation accounts for both HCI and BTI effects or either one of them. If you define the RelMode in the .MOSRA command as 1 or 2, it takes higher priority and applies to all MOSRA models. If you do not set its value or enter 0, RelMode in the .MOSRA command, then the RelMode inside individual MOSRA models takes precedence for that MOSRA model only; the rest of the MOSRA models take the RelMode value from the .MOSRA command. If you set any other value, except 0, 1, or 2, HSPICE issues a warning, and RelMode automatically resets to the default value 0.</p> <ul style="list-style-type: none"> ▪ 0: both HCI and BTI, Default ▪ 1: HCI only ▪ 2: BTI only
SimMode	<ul style="list-style-type: none"> ▪ 0: Select pre-stress simulation only ▪ 1: Select post-stress simulation only ▪ 2: Select both pre- and post-stress simulation, Default ▪ 3: Select continual degradation integration through alters <p>See also, MOSRA Support for DC/AC/MC Analysis in Post-Stress Simulation.</p> <p>When SimMode=1:</p> <ul style="list-style-type: none"> ▪ HSPICE reads in the *.radeg0 file, updates the device model for reliability analysis; then generates a new transient-output in a *.tr1 waveform file. ▪ The *.radeg file and input netlist must be in the same directory. ▪ The netlist stimuli could be different from the SimMode=0 netlist that generated the *.radeg file. <p>When SimMode=3:</p> <ul style="list-style-type: none"> ▪ If you do not specify .option radegfile in the top level netlist, simulation starts from fresh device. ▪ If you specify .option radegfile in the top level netlist, HSPICE reads in the last suite degradation in the radeg file, and continues the degradation integration and extrapolation from the corresponding circuit time in the radeg file. ▪ In consecutive alters, HSPICE reads in the radeg generated from the previous alter run. <p>See also Usage Model: SimMode=3 (continual degradation integration through alters)</p> <p>Note: You can use the command-line option <code>-mrasim</code> to overwrite the value of SimMode in a .MOSRA command card. Possible values are:</p> <ul style="list-style-type: none"> ▪ 0: Selects pre-stress simulation only ▪ 1: Selects post-stress simulation only ▪ 2: Selects both pre- and post-stress simulation ▪ 3: Selects continual degradation integration through .ALTERS

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

MOSRA Commands and Control Options

Argument	Description
AgingStart	Optionally defines time when HSPICE starts stress effect calculation during transient simulation. Default is 0.0.
AgingStop	Optionally defines time when HSPICE stops stress effect calculation during transient simulation. Default is tstop in .TRAN command.
AgingPeriod	Stress period. Scales the total degradation over time.
AgingWidth	The AgingWidth (circuit time “on”) argument works with the AgingPeriod argument. For example: if you specify AgingPeriod=1.0s and AgingWidth=0.5s, then the circuit turns on for 0.5s, and turns off for 0.5s. (The period is 1.0s.)
AgingInst	Selects MOSFET devices to which HSPICE applies HCI and/or BTI analysis. The default is all MOSFET devices with reliability model appended. You must enclose the name in quotes. HSPICE allows Multiple names /and supports wildcards.
Integmod	The flag selects the integration method and function. <ul style="list-style-type: none">▪ 0 (default): User-defined integration function in MOSRA API▪ 1: True derivation and integration method▪ 2: Linearized integration method (support non-constant n coefficient)
Xplatemod	The flag selects the extrapolation method and function. <ul style="list-style-type: none">▪ 0 (default): User-defined extrapolation function in MOSRA API▪ 1: Linearization extrapolation method (support non-constant n coefficient)▪ 2: Two-point fitting extraction and extrapolation method
Tsample1	First simulation time point of stress_total sampling for Xplatemod=2
Tsample2	Second simulation time point of stress_total sampling for Xplatemod=2
Agethreshold	Only when the degradation value \geq Agethreshold, the MOSFET information is printed in the MOSRA output file *.radeg or *.cvs file. Default is 0.
DegradationTime	If you specify this argument, the MOSRA API calculates the degradation at the degradation time, and generates a .degradation output file.
MosraLife	Argument to compute device lifetime for the degradation type specified. This argument has the same function as .option mosralife.
DegF	Sets the MOSFET’s failure criteria for lifetime computation. This argument has the same function as .option degf. If you specify .option degf, it takes precedence over .mosra degf.
DegFN	Sets the NMOS's failure criteria for lifetime computation. This argument has the same function as .option degfn. If you specify .option degfn, it takes precedence over .mosra degfn.

Argument	Description
DegFP	Sets the PMOS's failure criteria for lifetime computation. This argument has the same function as <code>.option degfp</code> . If you specify <code>.option degfp</code> , it takes precedence over <code>.mosra degfp</code> .
Frequency	User-specified frequency of the signal for BTI frequency-dependent recovery effect calculus. If you do not specify a value, The API automatically calculates it.

Description

Use the `.MOSRA` command to initiate HCI and BTI analysis. This is a two-phase simulation, the fresh simulation phase and the post stress simulation phase. During the fresh simulation phase, HSPICE computes the electron age/stress of selected MOS transistors in the circuit based on circuit behavior and the HSPICE MOSFET reliability model. During the post stress simulation phase, HSPICE simulates the degradation effect on circuit performance, based on the stress information produced during the fresh simulation phase.

If you specify either DEC or LIN the API ignores the RelStep value. See [Figure 103 on page 614](#) for an illustration of the `.MOSRA` command/syntax.

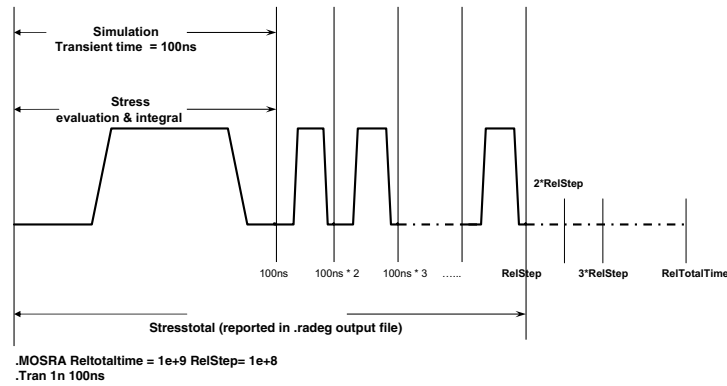
Example

```
.mosra reltotaltime=6.3e+8 relstep=6.3e+7
+ agingstart=5n agingstop=100n
+ aginginst="x1.*"
```

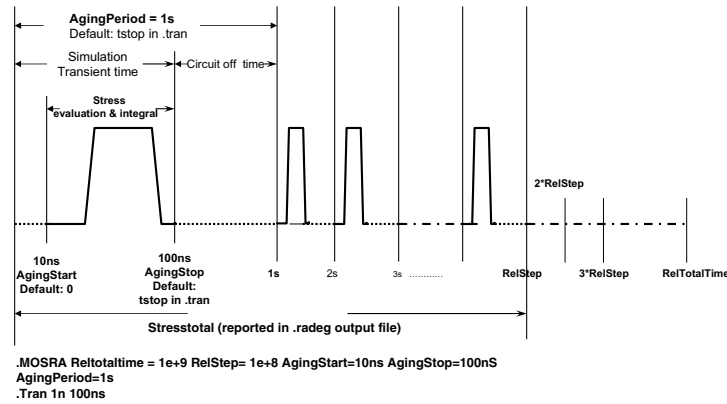
Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

MOSRA Commands and Control Options

Command/syntax for HSPICE reliability simulation (1/3)



Command/syntax for HSPICE reliability simulation (2/3)



Command/syntax for HSPICE reliability simulation (3/3)

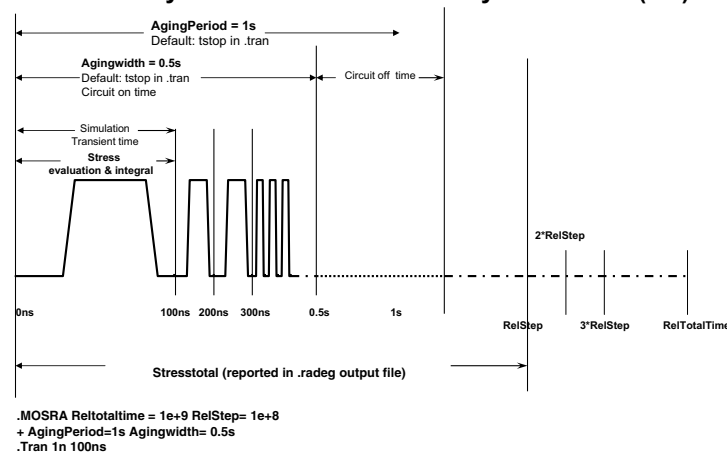


Figure 103 Graphic Illustration of MOSRA Command/Syntax

MOSRA Support for DC/AC/MC Analysis in Post-Stress Simulation

HSPICE MOSRA supports the DC/AC and Monte Carlo sweep in a `SimMode=1` Post-stress simulation. (HSPICE does not support `SimMode=0/2`.) A general MOSRA netlist must specify `SimMode = 1` in the `.MOSRA` command line to start the Post-stress simulation, and provide previously generated device degradation information through a user-specified `.radeg` input file. The syntax is as follows:

```
.mosra reltotaltime='10*365*24*60*60' lin=11 simmode=1  
.option radegfile = 'radeg file name'
```

Then you can specify the DC/AC analysis command. You can combine analysis commands with different sweeps (including Monte Carlo sweep).

```
.dc vdd 0 -1.2 -0.1 sweep monte=10  
.ac dec 1 1e5 1e9 sweep parm1 25 75 125  
.tran 1n 10n sweep monte=10
```

Examples

The following example netlist does a DC and AC simulation in MOSRA post-stress.

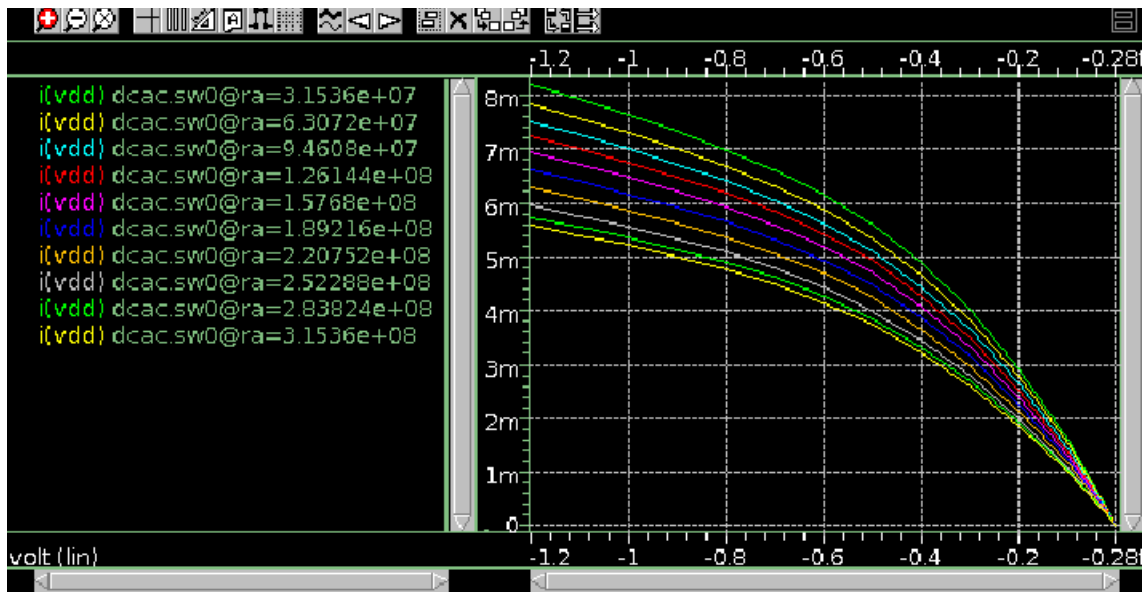
```
* MOSRA DC/AC TEST  
  
.option radegfile = 'simmode2.radeg0'  
  
vdd 1 0 -2 ac=1  
mp1 1 2 0 0 p1 l=0.1u w=10u ad=5p pd=6u as=5p ps=6u  
vgs 2 0 -2  
  
.model p1 pmos level=54 version=4.5  
  
.model p1_ra mosra level=1  
  
.appendmodel p1_ra mosra p1 pmos  
  
.mosra reltotaltime='10*365*24*60*60' lin=11 simmode=1  
.dc vdd 0 -1.2 -0.1  
.ac dec 1 1e5 1e9  
.options post  
.print dc i(vdd)  
.print ac i(vdd) ii(vdd)  
.end
```

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

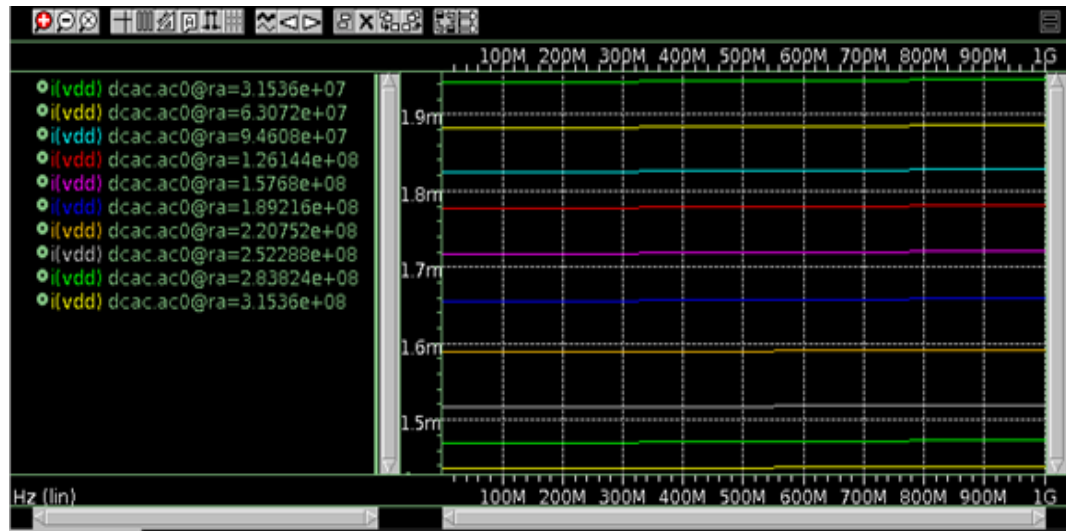
MOSRA Commands and Control Options

HSPICE parses the specified *radeg* file and do the post-stress simulation in the specified *relstep*, then generate the corresponding DC/AC output files:

```
dcac.sw0@ra=3.1536e+07
dcac.sw0@ra=6.3072e+07
dcac.sw0@ra=9.4608e+07
dcac.sw0@ra=1.26144e+08
dcac.sw0@ra=1.5768e+08
dcac.sw0@ra=1.89216e+08
dcac.sw0@ra=2.20752e+08
dcac.sw0@ra=2.52288e+08
dcac.sw0@ra=2.83824e+08
dcac.sw0@ra=3.1536e+08
```



```
dcac.ac0@ra=1.89216e+08
dcac.ac0@ra=2.20752e+08
dcac.ac0@ra=2.52288e+08
dcac.ac0@ra=2.83824e+08
dcac.ac0@ra=3.1536e+08
```



The following example netlist does a DC plus Monte Carlo simulation in MOSRA post-stress.

```
* MOSRA DC/MONTE TEST
.option radegfile = 'simmode2.radeg0'
vdd 1 0 -2
mp1 1 2 0 0 p1 l=0.1u w=10u ad=5p pd=6u as=5p ps=6u
vgs 2 0 -2
.param
+random1=AGAUSS(0,1,3)
+par1=random1
+A1_ms='par1*5.5225/2.0*3'
+toxn_ms_global = '(-9.462e-11)*A1_ms'
.model p1 pmos level=54 version=4.5
+ tox = '3e-9 + toxn_ms_global'
.model p1_ra mosra level=1
.appendmodel p1_ra mosra p1 pmos
.mosra reltotaltime='10*365*24*60*60' lin=11 simmode=1
.dc vdd 0 -1.2 -0.1 sweep monte=10
.options post
.print dc i(vdd)
.end
```

Getting Measurements in a MOSRA Analysis

See [Measurements in MOSRA Analysis](#) in [Chapter 12, Simulation Output](#) for instructions on generating multiple measurement files.

.MOSRAPRINT

Provides `.PRINT/ .PROBE` capability to access the electrical degradation of the specified element.

Syntax

```
.MOSRAPRINT output_name output_type(element_name, vds=exp1,  
+ vgs=exp2, vbs=exp3)
```

Argument	Description
<code>output_name</code>	User-defined output variable; this <code>output_name@element_name</code> is the as output variable name in the output file.
<code>output_type</code>	One of the following output variable types: <code>vth</code> , <code>gm</code> , <code>gds</code> , <code>ids</code> , <code>dids</code> or <code>dvth</code> .
<code>element_name</code>	The element name that the <code>.MOSRAPRINT</code> command applies.

Definition

The `.MOSRAPRINT` command supports the following models: B3SOI, B4SOI, PSP, BSIM3, BSIM4, HVMOS, HiSIM-HV, and Custom CMI MOSFETS.

This command provides access to device degradation information. The API uses the user-specified bias conditions, `vds`, `vgs` and `vbs` to characterize the device electrical property as specified by the output type. There is no order requirement for `vds`, `vgs`, and `vbs`. You can use wildcards '?' and '*' in `element_name`. The output variable `dids` reports the percent change of `ids` between post-stress simulation and fresh-simulation. `dvth` reports the change of `vth` between post-stress simulation and fresh-simulation. The output file format is the same as the measurement file format with file extension `*.ra`. You can use `.OPTION MEASFORM` with this command to produce `*.csv` files suitable for Microsoft Excel output.

Example

The following syntax prints the `ids` value of the MOSFET `m1`, when `vds = 5` `vgs=5`, `vbs=0`, at each `reltime` point.

```
.MOSRA reltotaltime=5e+7 relstep=1e+7  
.MOSRAPRINT ids(m1, vds=5, vgs=5, vbs=0)
```


.MOSRA_SUBCKT_PIN_VOLT

When a MOSFET is wrapped by a subckt-based macro model, this command specifies the subckt terminal voltages used by MOSRA model evaluation.

Syntax

```
.MOSRA_SUBCKT_PIN_VOLT subckt_name1, subckt_name2, ...
```

Description

Use this command to specify subckt-based macro terminal voltages HSPICE uses for MOSRA model evaluation.

subckt_name: The subcircuit name whose terminal voltages the API uses for MOSRA model evaluation.

Note: There is a limitation to this capability. The subckt-based macro model can contain only one MOSFET, and the number and definition of subckt terminals must be consistent with HSPICE MOSFET terminal number and definition.

Example

In this example, HSPICE uses subckt sub1's terminal voltages $v(d)/v(g)/v(s)/v(b)$, instead of the MOSFET M1's terminal voltages, $v(d1)/v(g1)/v(s1)/v(b1)$, for MOSRA model evaluation.

```
.subckt sub1 d g s b ...  
M1 d1 g1 s1 b ...  
Rd d d1 1k  
Rs s s1 1k  
Rg g g1 1k  
.model ...  
.ends  
.mosra_subckt_pin_volt sub1  
...  
.end
```

.MODEL

Syntax

```
.model mname mosra  
+ level=value  
+ [relmodelparam]
```

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)
MOSRA Commands and Control Options

Argument	Description
mname	User-defined MOSFET reliability model name
mosra	HSPICE model type name for MOSFET reliability model
level (alias: mosralevel)	To use the Synopsys MOSRA model, set LEVEL=1. For compatibility with HSIM, in the .MODEL statement, you can replace 'LEVEL=' with 'MOSRALEVEL='. HSPICE considers them equivalent. Example: HSPICE interprets the following two lines the same. <pre>.MODEL my_mod MOSRA LEVEL=1</pre> <pre>.MODEL my_mod MOSRA MOSRALEVEL=1</pre> To use custom MOSRA models and for discussion of LEVEL values, refer to the <i>HSPICE User Guide: Implementation of the MOSRA API</i> . Contact your Synopsys technical support team for more information.
relmodelparam	Reliability model parameter for HCI and BTI. See the HSPICE Application Note: Unified Custom Reliability Modeling API (MOSRA API) for sections on HCI and BTI parameters. Contact Synopsys Technical Support for access.

.APPENDMODEL

This command appends the parameter values from the source model card (SrcModel) to the destination model card (DestModel). The command requires all arguments. The .APPENDMODEL command supports wildcards. In addition, the .OPTION APPENDALL enables the top hierarchical level to use the .APPENDMODEL command even if the MOSFET model is embedded in a subcircuit. See [.OPTION APPENDALL](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Syntax

```
.appendmodel SrcModel ModelKeyword1 DestModel ModelKeyword2
```

Argument	Description
SrcModel	Source model name, e.g., the name of the MOSRA model.
ModelKeyword1	Model type for SrcModel. For example, the keyword “mosra”.

Argument	Description
DestModel	Destination model name, e.g., the original model in the model library.
ModelKeyword2	Model type for DestModel. For example, 'nmos'.

Example

This example appends the content of the model card hci_1 to the b3_nch BSIM3 model card.

```
.appendmodel hci_1 mosra b3_nch nmos
```

Wildcard Examples

In this example, the mosra model p1_ra appends to all of the pmos models. Note that you need quotation marks if only a wildcard defines model name.

```
.appendmodel p1_ra mosra "*" pmos
```

In the following example, the mosra model p1_ra appends to all of the pmos models with name pch* (pch1, pch2, pch_tt, etc.).

```
.appendmodel p1_ra mosra pch* pmos
```

The following sections discuss these control options:

- [.OPTION APPENDALL](#)
- [.OPTION DEGF](#)
- [.OPTION DEGFN](#)
- [.OPTION DEGFP](#)
- [.OPTION MOSRALIFE](#)
- [.OPTION MOSRASORT](#)
- [.OPTION MRAAPI](#)
- [.OPTION MRAEXT](#)
- [.OPTION MRAPAGED](#)
- [.OPTION MRA00PATH, MRA01PATH, MRA02PATH, MRA03PATH](#)
- [.OPTION RADEGFILE](#)
- [.OPTION RADEGOUTPUT](#)

.OPTION APPENDALL

With this option, you can use `.APPENDMODEL` at the main (uppermost) circuit level hierarchy even if the MOSFET model is embedded in a subcircuit. If there are `.APPENDMODEL` commands both in the main circuit and in a subcircuit, the `.APPENDMODEL` in the subcircuit has higher priority. For example:

```
.option appendall
.appendmodel n_ra mosra nch nmos

.SUBCKT mosra_test 1 2 3 4
M1 1 2 3 4 nch L=PL W=PW
.model nch nmos level= ...
.ENDS
```

The `.APPENDMODEL` command in the main circuit takes precedence.

```
.option appendall
.appendmodel n_ra mosra nch nmos

.SUBCKT mosra_test 1 2 3 4
M1 1 2 3 4 nch L=PL W=PW
.model nch nmos level= ...
.appendmodel n_ra1 mosra nch nmos
.ENDS
```

The `.APPENDMODEL` command in the subcircuit takes precedence.

.OPTION DEGF

Sets the MOSFET's failure criteria for lifetime computation.

Syntax

```
.OPTION DEGF=val
```

Description

In conjunction with `.OPTION MOSRALIFE`, this option sets the MOSFET's degradation value at lifetime. The options apply to all MOSFETs. The `RADEG` file prints the lifetime values.

.OPTION DEGFN

Sets the NMOS's failure criteria for lifetime computation.

Syntax

```
.option DEGFN=val
```

Description

In conjunction with `.OPTION MOSRALIFE`, this option sets the NMOS's degradation value at lifetime. If you do not specify the option or the `MRAlifetimeDeg` function cannot identify the keyword, HSPICE does not do the lifetime computation. The options apply to all MOSFETs. The `RADEG` file prints the lifetime values.

.OPTION DEGFP

Sets the PMOS's failure criteria for lifetime computation.

Syntax

```
.option DEGFP= val
```

Description

In conjunction with `.OPTION MOSRALIFE`, this option sets the PMOS's degradation value at lifetime. If you do not specify the option or the `MRAlifetimeDeg` function cannot identify the keyword, HSPICE does not do the lifetime computation. The options apply to all MOSFETs. The `RADEG` file prints the lifetime values.

.OPTION MOSRALIFE

Invokes the MOSRA “lifetime” computation.

Syntax

```
.OPTION MOSRALIFE=degradation_type_keyword [degF=val]  
[degFN=val] [degFP=val]
```

Description

Use this option to compute device lifetime calculation for the degradation type specified.

This option works with three others:

- `.OPTION DEGFP=val`— the designated MOSFET’s failure criteria for lifetime computation.
- `.OPTION DegFN=val`— the designated NMOS’s failure criteria for lifetime computation.
- `.OPTION DegFP=val` — the designated PMOS's failure criteria for lifetime computation.

The options apply to all MOSFETs. The `RADEG` file prints the lifetime values. The Synopsys built-in MOSRA Model Level 1 and the MOSRA API models support

lifetime calculus. (For the implementation of the lifetime function in the API models see the *HSPICE User Guide: Implementing the MOSRA API*, available by contacting the HSPICE technical support team.)

.OPTION MOSRASORT

Enables the descending sort for reliability degradation (RADEG) output.

Syntax

```
.OPTION MOSRASORT=degradation_type_keyword
```

Default delvth0

Description

Use this option `mosrasort` to enable the descending sort for reliability degradation (RADEG) output.

If you do not specify the `mosrasort` option, or HSPICE does not recognize the degradation type keyword, HSPICE the simulation omits the sort. (For a listing of Degradation type keywords see the *HSPICE User Guide: Implementing the MOSRA API*, available only by contacting the HSPICE technical support team.)

If you only specify the option `mosrasort`, and do not specify the degradation type keyword, HSPICE sorts RADEG by the `delvth0` keyword.

HSPICE sorts the output in two separate lists, one for NMOS, another for PMOS. HSPICE prints the NMOS device list first, and then the PMOS device list.

Example

In the following usage, the option does a descending sort for RADEG output on `delvth0`'s value.

```
.option mosrasort=delvth0
```

.OPTION MRAAPI

Enables and links the dynamically linked MOSRA API library.

Syntax

```
.OPTION MRAAPI=0 | 1
```

Default 0

Description

Use this option to enable and link the compiled MOSRA API .so library file to HSPICE during simulation. If you set this option parameter with no value or to 1, then the MOSRA API .so file loads as a dynamically linked object file.

If this option parameter does not exist in the netlist, or you explicitly set it to 0, the MOSRA API ignores the .so library.

.OPTION MRAEXT

Enables access to enhanced MOSRA API functions.

```
.OPTION MRAEXT=0 | 1
```

Default 0

Description

Use this option to enable access to enhanced MOSRA API functions. When `MRAEXT = 1`, HSPICE can access the extension functions.

.OPTION MRAPAGED

This option enables two modes of model parameter degradation.

Syntax

```
.OPTION MRAPAGED=0 | 1
```

Default 0

Description

If this option parameter is not in the netlist, or you explicitly set it to 0, degradation from the MOSRAAPI model is the parameter value shift with regard to the fresh model, `delta_P`. If you set this option parameter to 1, then the degradation from the MOSRAAPI model is the degraded model parameter, `P+delta_P`.

- 0: `delta_P` mode
- 1: Degraded model parameter

.OPTION MRA00PATH, MRA01PATH, MRA02PATH, MRA03PATH

These options enable string type variables in MOSRAAPI functions.

Syntax

```
.OPTION MRA00PATH = 'file_path1'
```

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

MOSRA Commands and Control Options

```
.option MRA01PATH = 'file_path2'  
.option MRA02PATH = 'file_path3'  
.option MRA03PATH = 'file_path4'
```

Default

NULL

Definition

Use these options to specify global string variables such as user-specified file paths for API model developers to access the MOSRA API functions.

.OPTION RADEGFILE

Use to specify a MOSRA degradation file name for use with `SIMMODE=1`.

Syntax

```
.OPTION RADEGFILE=file_name
```

Description

Use this option to specify a MOSRA degradation file name for use with `SIMMODE=1` (post-stress simulation only). HSPICE reads in the degradation information in the specified file and does the MOSRA post-stress simulation.

Example

```
.option radegfile = '1.radeg0'
```

.OPTION RADEGOUTPUT

Outputs the MOSRA degradation information in Word Excel CSV format.

Syntax

```
.OPTION RADEGOUTPUT=CSV
```

Default off (generates no CSV file)

Description

Use this option to output the MOSRA degradation information in Microsoft Excel CSV format. Does not generate a CSV file if you do not specify the `CSV` value.

Simulation Output Files

For each post-stress circuit time point, HSPICE computes stress information for each selected transistor and prints the degradation data to the *.radeg file.

HSPICE also outputs analysis and measurement results to these files:

.#@ra=circuit_time_point for TRAN/DC/AC analysis. For example:

- *.tr#@ra=circuit_time_point for transient analysis
- *.sw#@ra=circuit_time_point for DC
- *.ac#@ra=circuit_time_point for AC
- *.mt#@ra=circuit_time_point for transient measure
- *.ms#@ra=circuit_time_point DC measure
- *.ma#@ra=circuit_time_point for AC measure

In addition, HSPICE generates a file named *.*#@ra.grp, a waveform group file for MOSRA. When you use the Synopsys waveview tool, SX, to open this file, SX can load all corresponding *.*#@ra=circuit_time_point files.

HSPICE back-annotates the degradations to these transistors (aged_device) and performs the post-stress simulation. For example:

```
Degraded_device_parameter_Vth0 = Fresh_vth0 + delvth (NMOS)
Degraded_device_parameter_Vth0 = Fresh_vth0 - delvth (PMOS)
Degraded_device_paramter_U0 = Fresh_u0 * mulu0
```

HSPICE combines reliability data for all post-stress points in the following output *.radeg file.

```
delvth0 = 0.154229E-03
mulu0   = 99.9985%
```

```
** $DATA1 SOURCE='HSPICE' VERSION='C-2008.09 32-BIT'
*****Result of Reliability Analysis*****
mosrasort: delvth0
```

```
Circuit time 0.100000E+05
```

```
mn1
Device Type: NMOS
L= 0.100000E-06
W= 0.500000E-05
M= 0.100000E+01
Bias Direction: forward
delvth0 = 0.000000E+00
```

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

MOSRA Commands and Control Options

```
mulu0    =    100.0000%
mulua    =    100.0000%
mulub    =    100.0000%
muluc    =    100.0000%
delnfactor= 0.000000E+00
```

```
mn2
Device Type: NMOS
L= 0.100000E-06
W= 0.500000E-05
M= 0.100000E+01
Bias Direction: forward
delvth0  = 0.000000E+00
mulu0    =    100.0000%
mulua    =    100.0000%
mulub    =    100.0000%
muluc    =    100.0000%
delnfactor= 0.000000E+00
```

```
mp2
Device Type: PMOS
L= 0.100000E-06
W= 0.100000E-04
M= 0.100000E+01
Bias Direction: forward
delvth0  = 0.169531E-01
mulu0    =    100.0000%
mulua    =    100.0000%
mulub    =    100.0000%
muluc    =    100.0000%
delnfactor= 0.000000E+00
```

```
mp1
Device Type: PMOS
L= 0.100000E-06
W= 0.100000E-04
M= 0.100000E+01
Bias Direction: forward
delvth0  = 0.167418E-01
mulu0    =    100.0000%
mulua    =    100.0000%
mulub    =    100.0000%
muluc    =    100.0000%
delnfactor= 0.000000E+00
```

RADEG Output Sorting (.OPTION MOSRASORT)

HSPICE uses the option `mosrasort` to enable the descending sort for reliability degradation (RADEG) output.

```
.option mosrasort=degradation_type_keyword
```

The degradation type could be any degradation keyword in the RADEG output. For example, `.option mosrasort=delvth0`

HSPICE does a descending sort for RADEG output on `delvth0`'s value.

If you do not specify the `mosrasort` option, or HSPICE does not recognize the degradation type keyword, HSPICE omits the sort. (See the *HSPICE User Guide: Implementation of the MOSRA API*, available only by contacting HSPICE technical support, for a list degradation type keywords.) If you specify the option `mosrasort` without the degradation type keyword, HSPICE sorts RADEG by the `delvth0` keyword. HSPICE sorts the output in two separate lists, one for NMOS device, another for PMOS device. HSPICE prints the NMOS list first, and then the PMOS list.

Specify a MOSRA degradation File Name to be Used with SIMMODE=1 (.OPTION RADEGFILE)

Use `.OPTION RADEGFILE` to specify a MOSRA degradation file name for `SIMMODE=1` (post-stress simulation). HSPICE reads in the degradation information in the specified file and does the MOSRA post-stress simulation.

Syntax

```
.OPTION RADEGFILE=file_name
```

Example

```
.option radegfile = '1.radeg0'
```

Usage Model: SimMode=3 (continual degradation integration through alters)

With the D-2010.03 release, HSPICE supports changing the stress stimulus condition, as well as the temperature, during the course of reliability analysis. Here is an example of stimulus update due to battery decline.

Example— 0~1 year: Vdd=4v, 1~2 year: Vdd=3.8v, ... , 9~10 year: Vdd=2v.

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

MOSRA Commands and Control Options

This functionality utilizes the HSPICE `.alter` construct to facilitate a stimulus update. The flow of `SimMode=3` in the `.MOSRA` command provides the mechanism of continual degradation calculation/integration/extrapolation.

Construct the netlist as follows for the battery decline example above:

```
.options post accurate mraapi=1
.param vdd=4
vdd 1 0 'vdd'
mn1 1 1 0 0 n1 l=0.1u w=5u ad=5p pd=6u as=5p ps=6u
.model n1 nmos level=54 version= 4.4 vth0 = 0.25 ...
* mos reliability model card, MRA demo models
.model n1_ra mosra level=101 rela=1e-4 relb=2 reln=0.25 ...
* appendmodel command
.appendmodel n1_ra mosra n1 nmos
* mosra command
.mosra reltotaltime=3.15e+7 simmode=3 /* 1st 1 year */
.tran 1n 100n
.alter
.mosra reltotaltime=3.15e+7 simmode=3 /* 2nd year, reltotaltime
specified here is time interval*/
.param vdd=3.8 /* declined supply in 2nd year */
....
.alter
.mosra reltotaltime=3.15e+7 simmode=3 /* 10th year, reltotaltime
specified here is time interval */
.param vdd=2 /* declined supply in the 10th year */
.end
```

Notes

Consider these factors:

1. When `SimMode=3`, if you do not specify `.option radegfile` in the top level netlist, the simulation starts from fresh device; if you specify the option `radegfile` in the top level netlist, HSPICE reads in the last suite degradation in the `radeg` file and continues the stress calculation. In consecutive `.alters`, HSPICE reads in the `radeg` generated from the previous `alter` run.
2. The degradation output-format changes when `SimMode=3`. In a case where the MOSRA model has two degradations of the same type, the two degradations of the same type print out separately to facilitate the continual stress integration and extrapolation feature.

For example, assume a MOSRA model has two degradations of `delvth0`: a 10 mV one due to BTI, and a 20mV one due to HCI. In the original `radeg` file, the two degradations will be summed and output as:

```
delvth0 = 3.0000000e-2
```

In the new *radeg* file the two degradations are output separately:

```
Delvth0 = 1.000000e-2  
Delvth0 = 2.000000e-2
```

3. The MOSRA model cannot be changed in `alters`.

CSV Format Degradation Information (.OPTION RADEGOUTPUT)

HSPICE can output the MOSRA degradation information into CSV format by if you specify

```
.option radegoutput=csv
```

The file is readable as a Microsoft Excel file.

CSV file example:

```
Circuit time 0.315360E+09  
inst,type,L,W,M,Bias Direction,Lifetime,dids  
mn1,NMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,reverse,0.000000E+00,0.969292E+02  
mn2,NMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,forward,0.000000E+00,0.100000E+03  
mn3,NMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,forward,0.000000E+00,0.983077E+02  
mn4,NMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,forward,0.000000E+00,0.987247E+02  
mn5,NMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,forward,0.000000E+00,0.995699E+02  
mp1,PMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,reverse,0.000000E+00,0.970560E+02  
mp2,PMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,forward,0.000000E+00,0.100000E+03  
mp3,PMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,forward,0.000000E+00,0.982892E+02  
mp4,PMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,forward,0.000000E+00,0.996675E+02  
mp5,PMOS,0.100000E-06,0.100000E-  
05,0.100000E+01,reverse,0.000000E+00,0.981738E+02
```

Level 1 MOSRA BTI and HCI Model Parameters

The following tables list parameters and their descriptions for hot carrier injection (HCI) and the bias temperature instability (BTI).

The parameter listing tables are:

- NBTI/PBTI for V_{th} degradation ([Table 70](#))
- NBTI/PBTI for Mobility degradation ([Table 71 on page 635](#))
- HCI for NMOS and PMOS ([Table 72 on page 637](#))

For details on the Synopsys LEVEL 1 MOSRA model, contact your Synopsys technical support team.

Synopsys LEVEL1 mosra model, BTI Vth degradation

Table 70 Vth Degradation BTI Parameters

Name	Default	L term available	Description	Notes
TIT0	0	Yes	First parameter for interface-trap-inducing threshold voltage degradation	
TITCE	0	Yes	Inversion charge exponent for interface-trap-inducing threshold voltage degradation	
TITGA	0	Yes	Vgs dependence offset	
TITFD	0	Yes	Oxide electric field dependence for interface trap inducing threshold voltage degradation	
TITTD	0		Temperature dependent component of interface-trap-inducing threshold voltage degradation	
TITWC	0		Channel width coefficient for interface-trap-inducing threshold voltage degradation	
TITWE	0		Channel width exponent for interface trap inducing threshold voltage degradation	
TITLC	0		Channel length coefficient for oxide trap inducing threshold voltage degradation	
TITLE	0		Channel length exponent for oxide-trap-inducing threshold voltage degradation	
TN	0.25		Stress time exponent for interface-trap-inducing threshold voltage degradation	
TOT0	0		First parameter for oxide-trap-inducing threshold voltage	
TOTFD	0		Oxide electric field dependent component for oxide-trap-inducing threshold voltage degradation	
TOTTD	0		Temperature dependent component for oxide-trap-inducing threshold voltage degradation	

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

Level 1 MOSRA BTI and HCI Model Parameters

Table 70 *Vth Degradation BTI Parameters (Continued)*

Name	Default	L term available	Description	Notes
TOTDD	0	Yes	Drain voltage dependent coefficient for oxide electric field in threshold voltage degradation	
TOTWC	0		Channel width coefficient of oxide-trap-inducing threshold voltage degradation	
TOTWE	0		Channel width exponent of oxide-trap-inducing threshold voltage degradation	
TOTLC	0		Channel length coefficient of oxide-trap-inducing threshold voltage degradation	
TOTLE	0		Channel length component of oxide-trap-inducing threshold voltage degradation	
TK	0.5		Stress time exponent for oxide-trap-inducing threshold voltage degradation	
TTD0	1		First parameter for transient degradation of threshold voltage	TTD0=0 disables the Vth recovery effect
TDCD	0		Duty cycle dependent exponent for transient degradation of threshold voltage	TDCD is expected to be 0 or negative
TFC	0		Frequency dependence coefficient for BTI recovery of threshold voltage	
TFE	0		Frequency dependence exponent for BTI recovery of threshold voltage.	
TOTDE	1	Yes	Drain voltage exponent for oxide electric field in threshold voltage degradation	
EOXMOD	0 (disabled)		Enables separate electric field equations for mobility, DIBL, and VSAT degradations	

Table 70 *Vth Degradation BTI Parameters (Continued)*

Name	Default	L term available	Description	Notes
DLBTI	0		Length dependence offset for BTI model	
DWBTI	0		Width dependence offset for BTI model	
LNOMRA	0		Nominal length for NBTI length dependence limiting	

Mobility Degradation BTI Parameters

Table 71 *Mobility Degradation BTI Parameters*

Name	Default	L term available	Description	Notes
UIT0	0	Yes	First parameter for interface-trap-inducing mobility degradation	
UITCE	0	Yes	Inversion charge exponent for interface-trap-inducing mobility degradation	
UITGA	0	Yes	Vgs dependence offset	
UITFD	0	Yes	Oxide electric field dependence for interface-trap-inducing mobility degradation	
UITTD	0		Temperature dependent coefficient of interface-trap-inducing mobility degradation	
UITWC	0		Channel width dependent coefficient for interface-trap-inducing mobility degradation	
UITWE	0		Channel width exponent for interface-trap-inducing mobility degradation	
UITLC	0		Channel length dependent coefficient for interface-trap-inducing mobility degradation	
UITLE	0		Channel length exponent for interface-trap-inducing mobility degradation	

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

Level 1 MOSRA BTI and HCI Model Parameters

Table 71 Mobility Degradation BTI Parameters (Continued)

Name	Default	L term available	Description	Notes
UN	0.25		Stress time exponent for interface-trap-inducing mobility degradation	
UOT0	0		First parameter for oxide-trap-inducing mobility degradation	
UOTFD	0		Oxide electric field dependence for oxide-trap-inducing mobility degradation	
UOTTD	0		Temperature dependence for oxide-trap-inducing mobility degradation	
UOTWC	0		Channel width coefficient for oxide-trap-inducing mobility degradation	
UOTWE	0		Channel width exponent for oxide-trap-inducing mobility degradation	
UOTLC	0		Channel length coefficient for oxide-trap-inducing mobility degradation	
UOTLE	0		Channel length exponent for oxide-trap-inducing mobility degradation	
UK	0.5		Stress time exponent for oxide-trap-inducing mobility degradation	
UTD0	1		First parameter transient mobility degradation	UTD0=0 disables the mobility recovery effect
UDCD	0		Duty cycle dependent coefficient for transient mobility degradation	UDCD is expected to be 0 or negative
UFC	0		Frequency dependence coefficient for BTI recovery of mobility.	

Table 71 Mobility Degradation BTI Parameters (Continued)

Name	Default	L term available	Description	Notes
UFE	0		Frequency dependence exponent for BTI recovery of mobility.	
UOTDD	TOTDD	Yes	Drain voltage dependent coefficient for oxide electric field in mobility degradation	Used if EOXMOD=1
UOTDE	TOTDE	Yes	Drain voltage exponent for oxide electric field in mobility degradation	Used if EOXMOD=1

Synopsys LEVEL1 mosra model, HCI for NMOS and PMOS

Table 72 Threshold Voltage and Mobility Degradation HCI Parameters

Name	Default	L term available	Description	Notes
THCI0	0	Yes	First parameter for threshold voltage degradation induced by HCI	
THCI1	0		Coefficient for Isub-dependent term of mobility HCI degradation	
UHCI0	0	Yes	First parameter for mobility degradation induced by HCI	
UHCI1	0		Coefficient for Isub-dependent term of mobility HCI degradation	
TDII	0	Yes	Impact ionization current exponent for threshold voltage degradation induced by HCI	
TDLE	1		Channel length exponent for threshold voltage degradation induced by HCI	
TDCE	0	Yes	Channel current exponent for threshold voltage degradation induced by HCI	
TEA	0		Equivalent activation energy of Vth HCI degradation	

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

Level 1 MOSRA BTI and HCI Model Parameters

Table 72 Threshold Voltage and Mobility Degradation HCI Parameters (Continued)

Name	Default	L term available	Description	Notes
TDLT	0		Channel length dependence term of Vth HCI degradation	
THCVD	0	Yes	Coefficient for Vds-dependent term of Vth HCI degradation	
TDVD	0	Yes	Vds term exponent of Vth HCI degradation	
TDID	1	Yes	Channel current exponent for Vds-dependent term of Vth HCI degradation	
THCVB	0		Coefficient for Vbs dependence of Vth HCI degradation	
TDVB	1		Exponent for Vbs dependence of Vth HCI degradation	
HN	0.5	Yes	Time exponent for threshold voltage degradation induced by HCI	
UDCE	0	Yes	Channel current exponent for mobility degradation induced by HCI	
UDII	0	Yes	Impact ionization current exponent for mobility degradation induced by HCI	
UDLE	1		Channel length exponent for mobility degradation induced by HCI	
UEA	0		Equivalent activation energy of mobility HCI degradation	
UDLT	0		Channel length dependence term of mobility HCI degradation	
UHCVD	0	Yes	Coefficient for Vds-dependent term of mobility HCI degradation	
UDVD	0	Yes	Vds term exponent of mobility HCI degradation	

Table 72 Threshold Voltage and Mobility Degradation HCI Parameters (Continued)

Name	Default	L term available	Description	Notes
UDID	1	Yes		Channel current exponent for Vds-dependent term of mobility HCI degradation
UHCVB	0		Coefficient for Vbs dependence of mobility HCI degradation	
UDVB	1		Exponent for Vbs dependence of mobility HCI degradation	
0.5	HK	Yes	Time exponent for mobility degradation induced by HCI	
HIIL	0		First parameter for impact ionization	
HIIVD0	0		First vds dependent parameter of the impact ionization current	
HIIVD1	0		Second vds dependent parameter of the impact ionization current	
HIIVD2		0	Third vds dependent parameter of the impact ionization current	
VDSAT0	0		Nominal drain saturation voltage of the impact ionization current	
HIIT	0		Temperature dependent parameter for the impact ionization current	
HIIL	0		Channel length dependent parameter for the impact ionization current	
HIIE	0		Saturation channel electric field for the impact ionization current	

Chapter 19: MOSFET Model Reliability Analysis (MOSRA)

Level 1 MOSRA BTI and HCI Model Parameters

Table 72 Threshold Voltage and Mobility Degradation HCI Parameters (Continued)

Name	Default	L term available	Description	Notes
HIIVG0	0		First vgs dependent parameter for the impact ionization current	
HIIVG1	0		Second vgs dependent parameter for the impact ionization current	
HIIVG2	0		Third vgs dependent parameter for the impact ionization current	
HIIVGD	0		vds dependent parameter for the impact ionization current	
HIIVOFF	-1000 disabled		Offset voltage for Vgstep limiting	
ISUBMODE	1		lii vs. lsub model flag	
DLHCI	0		Length dependence offset for HCI model	
DWHCI	0		Width dependence offset for HCI model	
VBSMAX	1e6		Positive VBS limiting value for HCI model	

Post-Layout Simulation: RC Network Reduction and Back-Annotation

Describes post layout simulation including RC network reduction, simulations that contain a large number of parasitic elements, and back-annotation in HSPICE.

HSPICE back annotation supports Full Back-Annotation and Selective Net Back-Annotation for the DSPF and SPEF formats.

In HSPICE, the post-layout simulation is similar to pre-layout simulation. You can do the post-layout simulation with DSPF only if it is a fully extracted netlist with instances. You can include the DSPF file in the pre-layout netlist. You need to replace the ideal `.SUBCKT` blocks from your original netlist with `.SUBCKT` blocks containing the extracted parasitics. Remember to verify that the port order in the extracted `.SUBCKT` blocks match the port order in the ideal netlist. RC reduction ignores nodes with initial conditions set. If your circuit contains many initial condition statements, it is possible to see little or no change in resistor and/or capacitor counts after reduction.

If your extracted netlist is not too large (approximately 100,000 elements or fewer), then HSPICE can give you very good results. Otherwise, you can also employ an RC reduction.

HSPICE ships numerous of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

The following section cover these topics:

- [Linear Acceleration](#)
- [Linear Acceleration Control Options Summary](#)
- [Supporting Parasitic L- and K-elements](#)

- [Pruning Parasitics from a Post-Layout Flat Netlist](#)
- [Post-Layout Back-Annotation](#)

Linear Acceleration

By use of the `SIM_LA` option, you can accelerate the simulation of circuits that include large linear RC networks. To achieve this acceleration, HSPICE linearly reduces all matrices that represent RC networks. The result is a smaller matrix that maintains the original port behavior, yet achieves significant savings in memory and computation. Thus, the `SIM_LA` option is ideal for circuits with large numbers of resistors and capacitors, such as clock trees, power lines, or substrate networks.

In general, the RC elements separate into their own network. The shared nodes of both main circuit elements (including `.PRINT`, `.PROBE`, and `.MEASURE` statements) and RC elements are the port nodes of the RC network. All other RC nodes are internal nodes. The currents that flow into the port nodes are a frequency-dependent function of the voltages at those nodes. The multi-port admittance of a network represents this relationship.

- The `SIM_LA` option formulates matrices to represent multi-port admittance.
- Then, to eliminate as many internal nodes as possible, it reduces the size of these matrices, while preserving the admittance, otherwise known as port node behavior.
- The amount of reduction depends on the f_0 upper frequency, the threshold frequency where `SIM_LA` preserves the admittance ([Figure 104](#)).

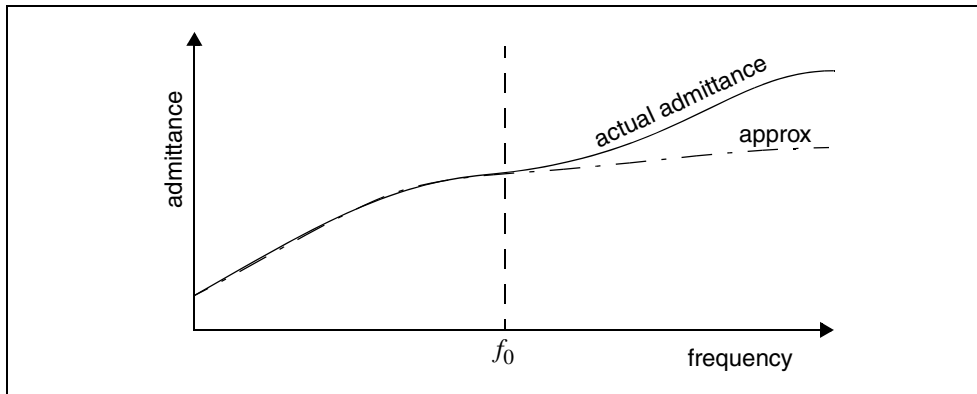


Figure 104 Multiport Admittance vs. Frequency

The `SIM_LA` option is very effective for post-layout simulation because of the volume of parasitics. For frequencies below f_0 , the *approx* signal matches that of the original admittance. Above f_0 , the two waveforms diverge, but the higher frequencies are not of interest. The lower the f_0 frequency, the greater the amount of reduction.

For the syntax and description of this control option, see [.OPTION SIM_LA](#) in the *HSPICE Reference Manual: Commands and Control Options*.

You can also choose one of three algorithms, explained in the following sections:

- [PACT Algorithm](#)
- [PI Algorithm](#)
- [LNE Algorithm](#)

PACT Algorithm

The `PACT` (Pole Analysis via Congruence Transforms) algorithm reduces the RC networks in a well-conditioned manner, while preserving network stability.

- The transform preserves the first two moments of admittance at DC (slope and offset), so that DC behavior is correct (see [Figure 105](#)).
- The algorithm preserves enough low-frequency poles from the original network to maintain the circuit behavior up to a specified maximum frequency f_0 , within the specified tolerance.

This approach is more accurate between these two algorithms, and is the default.

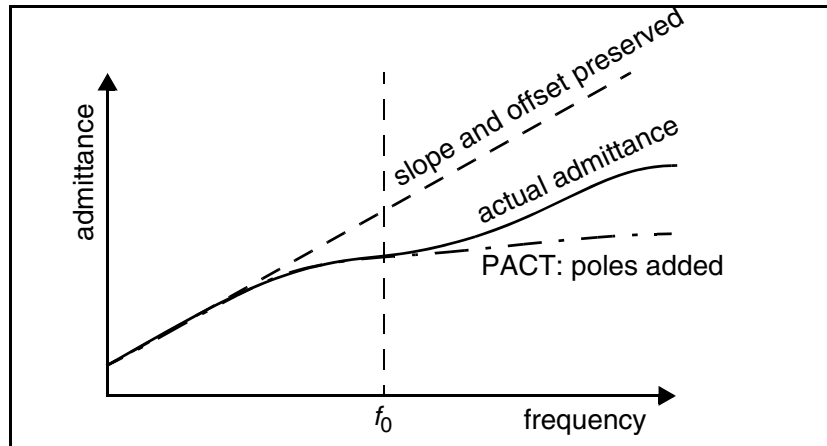


Figure 105 PACT Algorithm

PI Algorithm

This algorithm creates a *pi* model of the RC network.

- For a two-port, the *pi* model reduced network consists of:
 - a resistor that connects the two ports, and
 - a capacitor that connects each port to groundThe result resembles the Greek letter pi.
- For a general multi-port, `SIM_LA` preserves the DC admittance between the ports, and the total capacitance that connects the ports to ground. All floating capacitances lump to ground.

LNE Algorithm

The Linear Node Elimination (LNE) algorithm first checks the order of the nodes for low-to-high R-degree. If the R-degree is less than `MAXDEG` and the time constant ($=\text{sum}(C)/\text{sum}(G)$) of this node is small enough, the removal of the node takes place. Currently, the `MAXDEG` value is 7.

Linear Acceleration Control Options Summary

In addition to `SIM_LA`, other options are available to control the maximum resistance and minimum capacitance values to preserve, and to limit the operating parameters of the PACT algorithm. [Table 73](#) contains a summary of these control options. For their syntax and descriptions, see the respective option descriptions in Chapter 3, [HSPICE Netlist Simulation Control Options](#), in the *HSPICE Reference Manual: Command and Control Options*.

Table 73 PACT Options

Syntax	Description
<code>.OPTION SIM_LA=PACT PI LNE</code>	<p>Activates linear matrix reduction and selects any of the methods, where:</p> <ul style="list-style-type: none"> ▪ PACT invokes the Pole Analysis via Congruence Transforms algorithm ▪ PI creates a <i>pi</i> model to analyze the small signal behavior of bipolar junction and field effect transistors ▪ LNE invokes the Linear Node Elimination algorithm
<code>.OPTION LA_FREQ=<i>value</i></code>	<p>Upper frequency where you need accuracy preserved. <i>value</i> is the upper frequency for which the PACT algorithm preserves accuracy. If <i>value</i> is 0, PACT drops all capacitors because only DC is of interest. The maximum frequency required for accurate reduction depends on both the technology of the circuit and the time scale of interest. In general, the faster the circuit, the higher the maximum frequency. The default is 1GHz.</p>
<code>.OPTION LA_MAXR=<i>value</i></code>	<p>Maximum resistance for linear matrix reduction. <i>value</i> is the maximum resistance that the reduction preserves. <code>SIM_LA</code> assumes that any resistor greater than <i>value</i> has an infinite resistance, and drops the resistor after reduction finishes. The default is 1e15 ohms.</p>
<code>.OPTION LA_MINC=<i>value</i></code>	<p>Minimum capacitance for linear matrix reduction. <i>value</i> is the minimum capacitance preserved in the reduction. After reduction completes, <code>SIM_LA</code> lumps any capacitor smaller than <i>value</i> to ground. The default is 1e-16 farads.</p>

Table 73 PACT Options (Continued)

Syntax	Description
<code>.OPTION LA_TIME=<i>value</i></code>	Minimum time for preservation of accuracy. <i>value</i> is the minimum switching time for which the PACT and LNE algorithms preserve accuracy. HSPICE does not accurately represent waveforms that occur more rapidly than this time. LA_TIME is simply the dual of LA_FREQ. The default is 1ns, equivalent to setting LA_FREQ=1GHz.
<code>.OPTION LA_TOL=<i>value</i></code>	Error tolerance for the PACT and LNE algorithms. <i>value</i> is the error tolerance for the PACT and LNE algorithms, between 0.0 and 1.0. The default is 0.05.

Example

In this example, the circuit has a typical risetime of 1ns. Set the maximum frequency to 1 GHz, or set the minimum switching time to 1ns.

```
.OPTION LA_FREQ = 1GHz  
-or-  
.OPTION LA_TIME = 1ns
```

However, if spikes occur in 0.1ns, HSPICE does not accurately simulate them. To capture the behavior of the spikes, use:

```
.OPTION LA_FREQ = 10GHz  
-or-  
.OPTION LA_TIME = 0.1ns
```

Note: Higher frequencies (smaller times) increase accuracy, but only up to the minimum time step used in HSPICE.

Supporting Parasitic L- and K-elements

HSPICE supports simulation with parasitic L- and K-elements. You need to set the minimum value of mutual inductance by using the KLIM option. The default value of KLIM is 10mH. HSPICE does not calculate the second-order mutual inductance for values less than KLIM, but parasitic mutual inductance values can be many orders smaller than the default value.

Also, note that RC reduction is not very effective with respect to L- and K-elements. If you increase the simulation speed of your netlist having a huge

number of parasitic elements, you need to properly understand the accuracy versus speed trade-off. For more information about the KLIM option, see [.OPTION KLIM](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Pruning Parasitics from a Post-Layout Flat Netlist

To prune parasitics from a post-layout flat netlist, you can use the `.PRUNE` command to create an active-file netlist from a flat (`*.DSPF` file only) post-layout file. Use the following syntax to remove resistors and capacitors from a post-layout `*.spf` file by creating an active-net file to help speed cell characterization when simulating, for example, large macro blocks. Using the `.PRUNE` command helps you avoid back annotation steps.

```
.PRUNE "post-layout flat file" "active-net file"
```

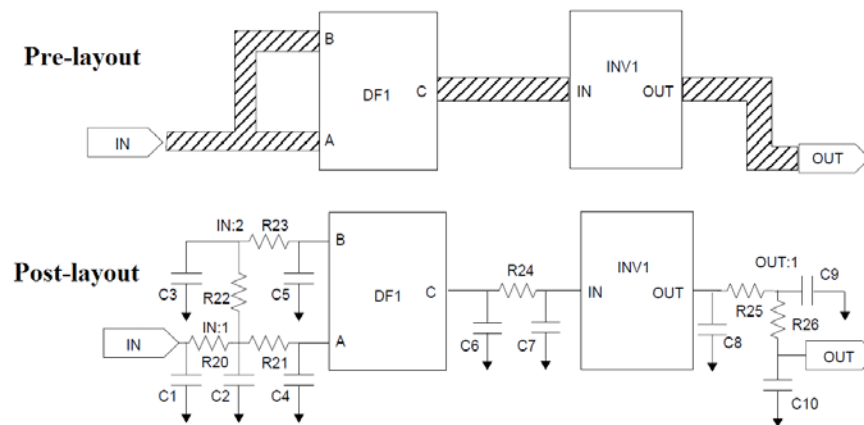


Figure 106 Simplifying cell characterization by pruning R/C parasitics

The process is as follows:

1. Obtain an active net file from a pre-layout netlist.
2. Use this active net file to create a pruned netlist on the flat post-layout netlist.
3. Re-use this pruned netlist in subsequent characterization runs.

Note: This flow is similar to the HSPICE selective net Back-Annotation flow. However, HSPICE performs the R/C de-coupling on the flat netlist instead of during back-annotation. See [Selective Net](#)

[Back-Annotation on page 653.](#)

Example

The following illustrates pruning capacitor and resistor parasitics from a flat post-processed *.spf netlist to create an active net input file in *.rcxt format.

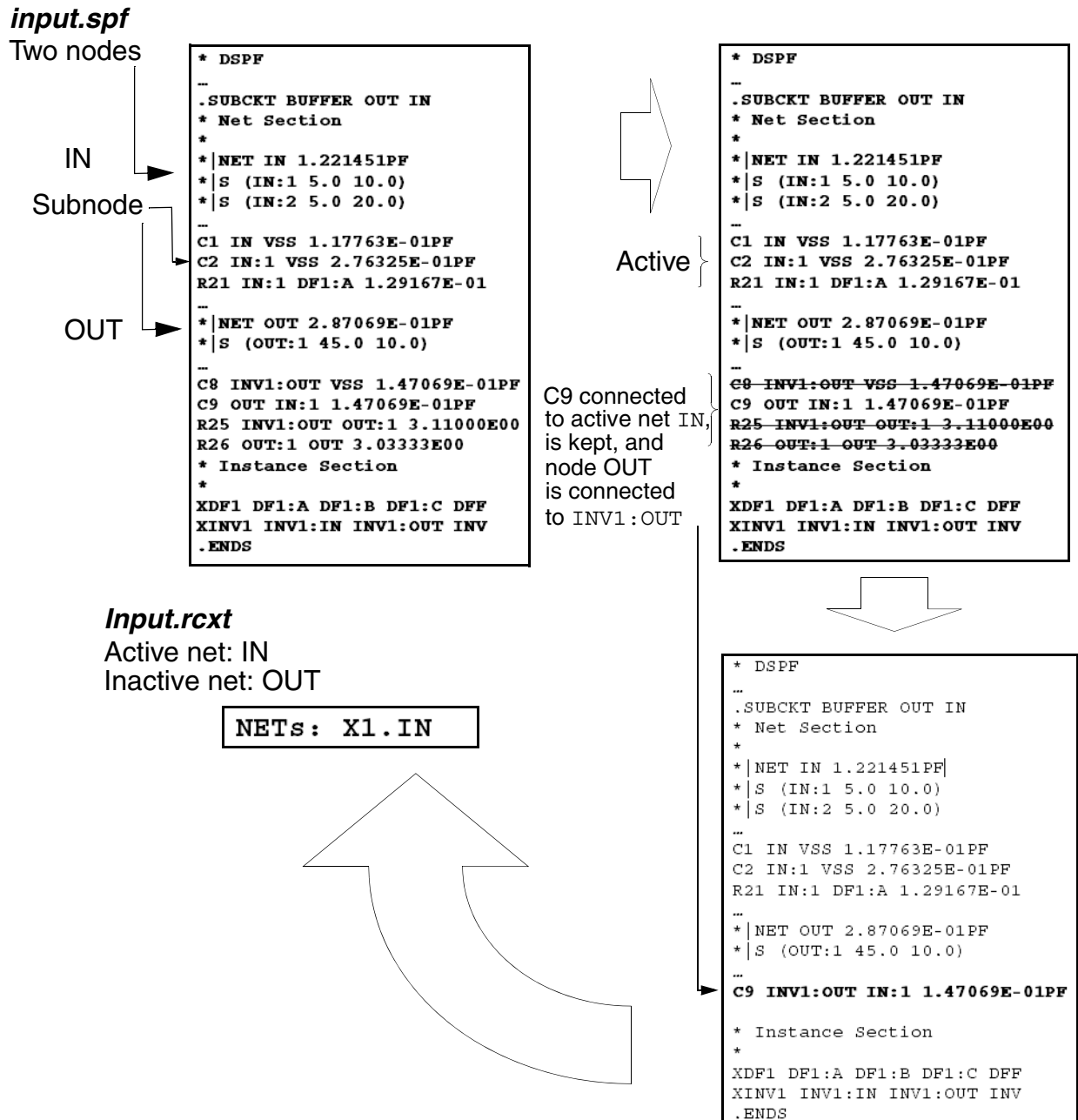


Figure 107 Converting a flat post-layout *.spf file to an active-net *.rcxt file.

Limitation

The active net name cannot contain a wildcard (*) character. For example, 'net*33' is illegal.

Post-Layout Back-Annotation

The traditional and straightforward way for post-layout analysis by HSPICE is to include the parasitic netlist containing parasitic RCs and devices as an ordinary netlist for simulation.

The problem with this approach is that the parasitic netlist is flat; hence you do not gain many advantages of hierarchical netlist, such as answers to these questions:

- How can different options apply to different blocks for better trade-off between accuracy and performance?
- How do you perform power analysis on a flat netlist to check the power consumption?
- The traditional flow flattens all nodes after extraction so it is more difficult to compare the delay before and after extraction.
- The traditional flow can also stress the limits of an extraction tool so reliability also becomes an issue.

To address these problems, HSPICE

- Needs a hierarchical (or flat) Layout Versus Schematic (LVS) ideal netlist for use by tools such as Star-RC or Star-RCXT, and a flat parasitic netlist in DSPF or SPEF format.
- Runs these two netlists to with back-annotation analysis. HSPICE can annotate the parasitic RCs (and devices in the instance section of DSPF file) to the ideal netlist and use the annotated ideal netlist for post-layout-like simulation.

This hybrid flat-hierarchical approach provides full control and advantages of simulating a hierarchical netlist. For example:

- You can back-annotate a part of a design for better accuracy and performance trade-off.
- You can perform power analysis based on the hierarchical ideal netlist to determine the power consumption of each block. Besides, you can reuse all post-processing statements for the pre-layout simulation for post-layout back-annotation simulation.

HSPICE enables parsing and annotating of two types of parasitic netlist formats:

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

The parasitic netlist describes the interconnect delay and load due to parasitic resistance and capacitance. You can represent parasitics on a net-by-net basis from a simple lumped capacitance to a fully distributed resistance capacitance tree.

For a useful application note see

The following sections cover these topics:

- [Full Back-Annotation](#)
- [Selective Net Back-Annotation](#)
- [Warnings/Error Messages](#)
- [Listing of Back-Annotation Commands and Options](#)
- [Application Note: Back-Annotation](#)
- [DSPF and SPEF File Structures](#)

Full Back-Annotation

The Full Back-Annotation flow annotates all nets from the parasitic netlist, and thus can produce the most accurate simulation result. For a large case with enormous number of parasitic RCs the full back-annotation flow could take much time and memory for the simulation, in which case [Selective Net Back-Annotation](#) might be a better choice. To invoke full back-annotation, you must supply two types of input files: an ideal netlist and a parasitic netlist in the format of DSPF/SPEF.

Flow for Full Back-Annotation

The option for invoking full back-annotation flow is `.OPTION BA_FILE`. For several examples of usage, see the demo cases and files shipped with HSPICE. Go to [Back-Annotation Demo Cases](#), and follow the path to `$installdir/demo/hspice/back_annotation/....`

The following cases illustrate the flow for full back-annotation:

<code>./option_ba_file/dspf/multiba</code>	Demonstrates use of the <code>.OPTION BA_FILE</code> command to launch multiple DSPF files for parasitic back-annotation.
<code>./option_ba_file/dspf</code>	Demonstrates use of the <code>.OPTION BA_FILE</code> command for single DSPF file.
<code>./option_ba_file/spef</code>	Demonstrates use of the <code>.OPTION BA_FILE</code> command for single SPEF file.

You need not specify which format (DSPF/SPEF) the parasitic netlist uses. HSPICE determines it automatically by analyzing the header of the parasitic netlist, so the file header must be clean to avoid adding confusing comments.

You can specify multiple parasitic netlists, each delimited by a semicolon. These parasitic netlists must be independent and *cannot* cross-reference each other.

Note: HSPICE back annotation supports encryption of `.DSPF` and `.SPF` files.

For descriptions and usage examples, see [.OPTION BA_FILE](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Example

Sample case for Full Back-Annotation Flow

```
*$ Full Back Annotation example for Inverter Circuit for DSPF
post layout netlist $*
Vsupply Vdd33 0 3.3
Vground Vss33 0 0.0
.temp 25
*****HSPICE BA implementation*****
.option ba_file='rc.spi' $$ DSPF post layout netlist
*****
.inc 'sch.spi' $$ schematic netlist
vin in 0 pulse (0 3.3 0 100p 100p 2n 4n)
.lib 'model.1' TT_3V
.option nomod post
.tran 1p 300n
.probe tran v(out)
**measurement to check period of the clock
.measure tran t_PERIOD TRIG v(out) val=0.5 RISE=15 TARG v(out)
+ val=0.5 RISE=16
.end
```

Selective Net Back-Annotation

Enable selective net back-annotation to improve the performance in post-layout simulations that contain a large number of parasitic RCs and focus on the sensitive blocks. A selective net back-annotation only annotates the nets that are active during an initial simulation run. This reduces both back-annotation and simulation time and improves the overall performance with the cost of some loss of accuracy.

With the F-2011.09 release, Selective BA supports the HSIMBA format.

You need three types of input files to invoke selective net back-annotation.

- An ideal netlist
- A parasitic netlist as used by full back-annotation
- An active net file, which can be in either of two formats: StarRC or HSIMBA.

Examples, Active Net Files

The content of active net file is case-insensitive for HSPICE.

Example 1 Active Net File sample in format of StarRC

```
NETS: A0
NETS: B0
NETS: A1
NETS: B1
NETS: X1/N6
```

Example 2 Active Net File sample in HSPICE format

```
NETS = {
    A0
    B0
    A1
    B1
    X1/N6
}
```

Flow for Selective Net Back-Annotation

To invoke selective net back-annotation, specify both `.OPTION BA_FILE` and `BA_ACTIVE`.

The option for invoking the selective net back-annotation flow is `.OPTION BA_ACTIVE`. For examples of usage, see the demo cases and files shipped with HSPICE. Go to [Back-Annotation Demo Cases](#), and follow the path to `$installdir/demo/hspice/back_annotation/...`

The following case illustrates the flow for selective net back-annotation:

<code>selective_ba/option_ba_active</code>	Demonstrates use of <code>.OPTION BA_ACTIVE</code> to specify the active net file name(s) for selective net back-annotation.
--	--

You do not need to specify the format of the active net file. HSPICE determines it automatically by analyzing its header. You can specify multiple active net files, each delimited by a semicolon.

Note: For net names such as `module.xi1/xi2/net_name`, by default, HSPICE truncates this name from the last period and identifies the net name as `xi1/xi2/net_name`. To use the full net name, i.e., `module.xi1/xi2/net_name`, use the HSPICE control option `.OPTION BA_ACTIVEHIER`.

Example

The following case illustrates a Selective Net Back-Annotation Flow.

```
*$ Selective Net Back Annotation example for Inverter Circuit for
DSPF post layout netlist $*
Vsupply Vdd33 0 3.3
Vground Vss33 0 0.0
.temp 25
*****HSPICE BA implementation*****
.option ba_file='rc.spi' $$ DSPF post layout netlist
.option BA_ACTIVE='selective.rcxt' $$active net file
*****
.inc 'sch.spi' $$ schematic netlist
.lib 'model.l' TT_3V
cload out 0 10f
.option nomod converge=100
.tran lp 300n
.probe tran v(out)
**measurement to check period of the clock
.measure tran t_PERIOD TRIG v(out) val=0.5 RISE=15
+ TARG v(out) val=0.5 RISE=16
.end
```

Active Net Generation

An active net file is needed to run Selective Net Back-Annotation. This active net file used to be generated by other tools, such as Synopsys HSPICE. Now, HSPICE can generate active net files (since the F-2011.09 release).

Flow for Active Net Generation

Use the `.BA_ACHECK` command to invoke Active Net Generation. The demo case below shows how to use this command. Go to [Appendix 31, Back-Annotation Demo Cases](#), and follow the path `$installdir/demo/hspice/back_annotation/...`

The following case illustrates the flow for active net back-annotation:

<code>selective_ba/ba_acheck/</code>	Demonstrates the use of the <code>.BA_ACHECK</code> command to specify the rule for detecting node activity.
--------------------------------------	--

The default format for the active net file is HSPICE format; the file name is `design.hsimba0`. Another available format available is the StarRC format; the file name is `design.rcxt0`. To generate the StarRC format, use the HSPICE control option `.OPTION BA_NETFMT=1`.

For descriptions and usage examples, see [.BA_ACHECK](#) and [.OPTION BA_NETFMT](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Example

The following case illustrates an Active Net Generation Flow:

```
*$ Selective Back Annotation example for TG BASED XOR Circuit for  
DSPF post layout netlist $*
```

```
Vsupply vdd 0 1.2  
Vground vss 0 0.0  
.option runlvl  
.temp 25  
*****HSPICE BA Active Net implimentation*****  
.ba_acheck dv=1 tstart=1u tstop=20u exclude='x*'  
.option ba_netfmt=1 $ 0: HSIM 1: StarRC  
*****  
.inc 'sch.spi' $$ schematic netlist  
*xadder sum vss carry vdd in2 in1 adder  
vin1 a 0 pulse (0 1.2 100p 100p 100p 2u 4u)  
vin2 b 0 pulse (0 1.2 100p 100p 100p 3.5u 7u)  
vclk clk 0 pwl (0 1.2 23u 1.2 23.1u 0)  
vclkb clkb 0 pwl (0 0 23u 0 23.1u 1.2)  
.option ba_activehier=1  
.lib 'model.1' TT  
.option nomod post  
.tran 1p 30u  
.probe tran v(out) v(a) v(b) v(clk) v(clkb)  
.end
```

Warnings/Error Messages

HSPICE may issue warnings when doing back-annotation. The following are workarounds and solutions when appropriate.

- Warnings for cutoff coupling capacitors—Coupling capacitors across two nets are very common in parasitic netlists. For example, assume one coupling capacitor (CC) with terminals connected to two nodes on nets A and B, respectively. When HSPICE launches selective net back-annotation and net A is active while net B is inactive, it cuts off CC from the node under net B and the terminal becomes a dangling node. HSPICE issues a low-level warning message as such as “undefined node; might be defined in inactive netnet.” By default, HSPICE gives such a warning and processes the cutoff terminal of CC as an ordinary dangling node. Additional

modes are available in HSPICE to change the default behavior (see [.OPTION BA_COUPLING](#)). When you apply `.OPTION BA_COUPLING`, the warning changes to `undefined node; reset to GROUND_NET` or `undefined node; reset to pre_layout node: N1` according to the option setting.

- Warnings for terminal name mismatch—Cases where the terminal name used in the parasitic netlist is not consistent with the one used in the ideal netlist might generate a warning. For example: The terminal name for MOSFET in parasitic netlist is `M1:UDRN`, while the default terminal name for MOSFET recognized by HSPICE is `D[R][A][I][N]`. Another case is the terminal name in parasitic netlist `XM1:D`, while in the ideal netlist the corresponding subckt definition has a node list of `N1, N2, N3, N4`. Generally, HSPICE is able to correct this kind of mismatch automatically. If HSPICE cannot achieve this auto-correction, the tool gives a warning message such as: `cannot find the node pin`. To eliminate such warning messages, specify [.OPTION BA_TERMINAL](#) to explicitly map these terminals from the parasitic netlist to the ideal netlist.
- Warning for undefined PIN node—You must declare all pin nodes in a parasitic netlist. If you violate this rule, HSPICE warns `undefined node`, and all cuts off all connected element terminals to that node.
- Warning for missing instance—When the element instance referenced in the parasitic netlist cannot be found in the ideal netlist, HSPICE gives the warning message: `cannot find the instance`.
- Warning for invalid connectivity—If the node connectivity in the parasitic netlist conflicts with the connectivity in the ideal netlist, HSPICE corrects this error automatically and then gives the warning message `invalid connectivity; corrected now`.

Listing of Back-Annotation Commands and Options

The following is an alphabetical listing of all the back annotation command and options available in HSPICE. See the *HSPICE Reference Manual: Commands and Control Options* for details on each of these modes. For sample usage of these commands see [Application Note: Back-Annotation on page 658](#).

- [.BA_ACHECK](#): Specifies the rule for detecting node activity.
- [.OPTION BA_ACTIVE](#): Specifies the active net file name(s) for selective net back-annotation.

- **.OPTION BA_ACTIVEHIER**: Annotates full hierarchical net names for BA_ACTIVE files.
- **.OPTION BA_ADDPARAM**: Specifies extra parameters for scale by .OPTIONS BA_SCALE/BA_GEOSHRINK.
- **.OPTION BA_COUPLING**: Controls how to treat cutoff coupling capacitors when invoking selective net back-annotation.
- **.OPTION BA_DPFPFX**: Prepends an extra prefix when searching the ideal netlist for referenced instances in the parasitic file (DSPF/SPEF/DPF).
- **.OPTION BA_ERROR**: Handles errors on nets.
- **.OPTION BA_FILE**: Launches parasitic back-annotation.
- **.OPTION BA_FINGERDELIM**: Explicitly specifies the delimiter character used for finger devices.
- **.OPTION BA_GEOSHRINK**: Acts as element scaling factor for .OPTION BA_SCALE.
- **.OPTION BA_HIERDELIM**: Specifies the hierarchical separator in the DPF file.
- **.OPTION BA_IDEALPFX**: Specifies the extra prefix to prepend when searching the ideal netlist for referenced instances in the parasitic file (DSPF/SPEF/DPF). For an example, see **.OPTION BA_IDEALPFX** in the *HSPICE Reference Manual: Commands and Control Options*.
- **.OPTION BA_MERGEPORT**: Controls whether to merge net ports into one node.
- **.OPTION BA_NETFMT**: Specifies the format of the active net file.
- **.OPTION BA_PRINT**: Controls whether to output nodes and resistors/capacitors introduced by back-annotation.
- **.OPTION BA_SCALE**: Sets the element scaling factor for instances in the DPF file separately.
- **.OPTION BA_TERMINAL**: Specifies the terminal name mapping between the parasitic netlist and the terminal names that the simulator recognizes.

Application Note: Back-Annotation

HSPICE provides for both full back-annotation and selective back annotation.

The following sections present these topics:

- [Full Back-Annotation](#)
- [Selective Back-Annotation](#)
- [Recommended Settings for STAR-RC-XT Command File](#)

Full Back-Annotation

For back-annotation in HSPICE, you need the following files:

- Schematic netlist
- Post-layout netlist

To perform back-annotation in HSPICE:

1. Specify the post-layout netlist file with:

```
.option ba_file="full_path_of_post_layout_netlist_file"
```

2. Include the schematic netlist as follows:

```
.include "schematic_netlist"
```

Result: HSPICE returns a back-annotation report in the *.lis file.

Sample report:

In the following report, the number within the parentheses is the number parsed and the number outside the parentheses is the actual annotated figure.

```
***** begin BA *****  
BA File="postlayout_netlist.spi"  
*** BA Statistics ***  
BA Nets back annotated (parsed): 6 (6)  
BA Resistors back annotated (parsed): 204 (204)  
BA Capacitors back annotated (parsed, coupled): 251 (251, 222)  
BA Instances back annotated (parsed): 6 (6)  
BA Time = 0.01 sec  
***** end BA *****
```

The following sections present the syntaxes to handle these situations:

- [Multi Post-Layout Back-Annotation](#)
- [Back-Annotation for Stem Ports](#)
- [Output Control for Back-Annotation](#)
- [Scaling During Back-Annotation](#)
- [Terminal Mapping in Back-Annotation](#)
- [Prefix Handling for Devices of Post-layout Netlist](#)

- [Prefix Handling for Devices of a Pre-layout Netlist](#)
- [Delimiter Handling for Fingered Devices](#)
- [Handling the Hierarchy Separator](#)

Multi Post-Layout Back-Annotation

HSPICE back-annotation supports multi post-layout netlists at all levels of the hierarchy. You can specify multiple file by using `.option ba_file`, separated by a semicolon (;). This syntax is:

```
.option ba_file = "file1 [; file2; file3; ...]"
```

HSPICE modifies the reporting structure to issue a back-annotation report for an individual post-layout netlist, and then issues a separate back-annotation report.

```
***** begin BA *****
BA File="rc1.spi"

*** BA Statistics ***
BA Nets back annotated (parsed): 6 (6)
BA Resistors back annotated (parsed): 204 (204)
BA Capacitors back annotated (parsed, coupled): 251 (251, 222)
BA Instances back annotated (parsed): 6 (6)
BA File="rc2.spi"

*** BA Statistics ***
BA Nets back annotated (parsed): 4 (4)
BA Resistors back annotated (parsed): 78 (78)
BA Capacitors back annotated (parsed, coupled): 30 (30, 0)
BA Instances back annotated (parsed): 2 (2)

*** Multi-BA Summary ***
BA File="rc1.spi"
BA File="rc2.spi"
BA Nets back annotated (parsed): 10 (10)
BA Resistors back annotated (parsed): 282 (282)
BA Capacitors back annotated (parsed, coupled): 281 (281, 222)
BA Instances back annotated (parsed): 8 (8)
BA Time = 0.01 sec
***** end BA *****
```

Back-Annotation for Stem Ports

In a schematic, a pin or net which appears as a single connection between two points can actually be multiple parallel connections of metals in the layout.

This extraction is controlled by the **SHORT_PINS** command of STAR-RCXT. When you set this option to **NO**, the stem connects to a pin in the form of a bus.

Users employ this extraction feature for measuring stem currents. HSPICE supports the bus type pin and provides it with an option:

```
.option ba_mergeport= [1|0]
```

By default, the value is 1 which shorts the ports. To incorporate stems, set `.option ba_mergeport=0`.

Output Control for Back-Annotation

During back-annotation, HSPICE creates its own node names for parasitics. Because of the large number of parasitics, your search for a particular node or operating point in the waveform may become tedious. For ease of use, HSPICE supports storing only pre-layout nodes names by using:

```
.option ba_print = ideal
```

The default value for this option is `all` which includes parasitic nodes.

Scaling During Back-Annotation

HSPICE provides scaling for back-annotation using two options for back-annotated post layout netlists:

- `.option ba_scale`
- `.option ba_geoshrink`

In back-annotation (for a post layout netlist):

$$\text{final_dimension} = \text{original_dimension} \cdot \text{ba_scale} \cdot \text{ba_geoshrink}$$

These options, by default, are equal to `.option scale` and `.option geoshrink`, respectively.

There is a standard practice in PDKs for parameters that scale. However there is possibility of different parameter names, for example, instead of 'W' use of 'Wr', etc. To account for such situations, HSPICE provides:

```
.option ba_scaleparam="LINEAR: [param11 param12]...;  
+ QUAD: [param21 param22] ..."
```

For example:

The following commands HSPICE to scale WR, LR parameters linearly and ASR, AREAR parameters quadratic- ally.

```
.OPTION BA_SCALEPARAM = "LINEAR: WR LR; QUAD: ASR AREAR"
```

Terminal Mapping in Back-Annotation

The terminal name is a set of strings found in device parasitic information for the RC netlist, following the netlist delimiter. For example:

In this example Here DRN, GATE, SRC, and BULK are terminal names for the MOSFET.

```
Mmn1 mn1#DRN mn1#GATE mn1#SRC mn1#BULK nch3 ad=0.2016p
+ as=0.2016p l=0.35u nrd=0 nrs=0 pd=1.8u ps=1.8u sa=0.48u
+ sb=0.48u sca=8.27147 scb=0.004406 scc=3.6e-05 w=0.42u
...
```

HSPICE provides terminal names for other elements, as well.

Note: The letters in the square brackets are optional.

Device	Terminal 1	Terminal 2	Terminal 3	Terminal 4
M(MOSFET)	D [R] [A] [I] [N]	G [A] [T] [E]	S [O] [U] [R] [C] [E]	B [U] [L] [K]
Q(BJT)	C [O] [L] [L] [E] [C] [T] [O] [R]	B [A] [S] [E]	E [M] [I] [T] [T] [E] [R]	S [U] [B] [S] [T] [R] [A] [T] [E]
R(Resistor)	A [N] [O] [D] [E]	C [A] [T] [H] [O] [D] [E]	S [U] [B] [S] [T] [R] [A] [T] [E]	N/A
C(Capacitor)	P [L] [U] [S]	M [I] [N] [U] [S]	S [U] [B] [S] [T] [R] [A] [T] [E]	N/A
D(Diode)	P [O] [S] [I] [T] [I] [V] [E]	N [E] [G] [A] [T] [I] [V] [E]	S [U] [B] [S] [T] [R] [A] [T] [E]	N/A

When a terminal name may be different from those listed above for elements, HSPICE provides an option to map different user terminal names to what the simulator recognizes.

```
.option ba_terminal = "terminal_index1 alias1
+ [; terminal_index2 alias2; terminal_index3 alias3; ...]"
```

The following example maps user-defined terminals (UDRN, UGATE) in the parasitic netlist to default terminal characters (D, G).

TBD

Prefix Handling for Devices of Post-layout Netlist

Sometimes, a post-layout netlist has device names with prefixes. For example, for a pre-layout device name such as M1, HSPICE may change it to M_mM1 in post-layout netlist.

The HSPICE back-annotation engine tries to match such devices. However, you should explicitly include the prefix by using:

```
.option ba_dpfpfx = "prefix_string"
```

For the example above, you need to specify:

```
ba_dpfpfx = "M_"
```

Prefix Handling for Devices of a Pre-layout Netlist

Sometimes, a pre-layout netlist has device names with prefixes. For example, a pre-layout device name such as xmM1 may change to mM1 in a post-layout netlist.

The HSPICE back annotation engine tries to match such devices. However, you should explicitly include the prefix using:

```
.option ba_idealpfx = "prefix_string"
```

For the example above, you must specify:

```
.option ba_idealpfx = "xm"
```

Delimiter Handling for Fingered Devices

In a post-layout netlist, multi-fingered device names have delimiters followed by a number. For a pre-layout netlist device listed as:

```
MP1 d g s b w=1u l=1u M=3
```

In the post-layout, it may appear as:

```
MMP1 d g s b w=1u l=1u ...
```

```
MMP1#1 d g s b w=1u l=1u ...
```

```
MMP1#2 d g s b w=1u l=1u ...
```

where: '#' is the finger delimiter.

The HSPICE back annotation engine, by default, recognizes '@'. However, for other delimiters, you must explicitly include the same using:

```
.option ba_fingerdelim = "delimiter_string"
```

So, for above example, you have to specify:

```
.option ba_fingerdelim = "#"
```

Handling the Hierarchy Separator

Usually, a STARC DSPF file displays Hierarchy divider information as:

```
*|DIVIDER :
```

This notation plays a role in understanding hierarchy. However, in cases when a divider is not present, explicitly define the separator by using:

```
.option ba_hierdelim = "delimiter_string"
```

For example, for the colon (:) hierarchy separator (DIVIDER), specify:

```
.option ba_hierdelim = ":"
```

Selective Back-Annotation

Selective back-annotation allows you to selectively back-annotate the post-layout netlist per *active net* criteria. HSPICE uses active net, a set of nets / pins specified in *.rcxt or *.rc format for back-annotating an RC netlist.

For selective back-annotation in HSPICE, you need the following files:

- Schematic netlist
- Post-layout netlist
- Active net file (in .rcxt or .rc format)

To perform selective back-annotation in HSPICE:

1. Specify the post-layout netlist file with

```
option ba_file="full_path_of_post_layout_netlist_file"
```

2. Include the schematic netlist by using:

```
.include "schematic_netlist"
```

3. Specify the path of an active net file by using:

```
.option ba_active="active_net_file"
```

Result: HSPICE writes the back-annotation report to the *.lis file as in the following sample report:

```
***** begin BA *****
Active Nets File="selective.rcxt"

BA File="rc.spi"

*** BA Statistics ***
BA Nets back annotated (parsed): 11 (11)
BA Resistors back annotated (parsed): 255 (255)
BA Capacitors back annotated (parsed, coupled): 100 (100, 0)
BA Instances back annotated (parsed): 22 (22)
BA Time = 0.01 sec
***** end BA *****
```

When you define tolerance change in voltage value of nets, you create the active net files by using the following syntax:

```
.BA ACHECK [include=node_pattern] [exclude=node_pattern]  
+ [dv=val1] [level=val2] [tstart=start_time] [tstop=stop_time]  
where:
```

- `include` and `exclude` are useful options for specifying nets with wildcards.
- `dv` specifies the threshold of 'change of voltage'. So, if you specify `dv=0.1`, HSPICE considers all nets which vary more than 0.1 V as active nets.
- `level` causes the simulator to pick nets from the TOP level to the hierarchy level specified. For `level=2`, HSPICE considers active nets from the TOP level and immediate hierarchy level. By default, `level=0`, which means that HSPICE considers the full hierarchy.
- `tstart` and `tstop` define start and stop times, for which you want to perform an active net check. By default, these are the same as transient start and stop times.

HSPICE displays the report on created active net file as:

```
***** Active Net Information *****  
Active Net file:hsp.rcxt  
Active net threshold:1.00E+00v  
#Active nets:68(7.83% of total nets)  
*****
```

Active net files include embedded subckt instance names. By default, HSPICE creates an active net file in `.hsimba#` format, which is the HSPICE active net format.

To write the active net file in `.rcxt#` format, use:

```
.option ba_netfmt = 1
```

By default, HSPICE does not recognize module names and results in incorrect back-annotation. To correct such a situation, use:

```
.option ba_activehier = [0|1]
```

The default is 0; to correct problems due to use of a module instead of a subckt instance name, set:

```
ba_activehier=1
```

Coupling Capacitance Handling during Selective Back-Annotation

During selective back-annotation a coupling capacitance between active and inactive net may occur. By default, HSPICE does not connect the coupling cap of an active node to an inactive node and so leaves the node dangling.

HSPICE provides flexibility for the user to choose the connection by using:

```
.option ba_coupling = [0|1|2]
```

where:

- `.option ba_coupling = 0` (default) - Leaves the active net terminal of coupling capacitor under process as an ordinary dangling node.
- `.option ba_coupling = 1` - Connects the active net terminal coupling capacitance to the node which STAR-RC defines by * | GROUND_NET in the command file.
- `.option ba_coupling = 2` - Connects the active net terminal coupling capacitance to the unexpanded inactive node.

To achieve the most accurate results, set:

```
.option ba_coupling=2
```

Recommended Settings for STAR-RC-XT Command File

Besides the simulator, back-annotation depends on a STAR-RC-XT command file. Recommended settings for this file are as follows:

- EXTRACTION: RC
- REDUCTION: NO
- COUPLE_TO_GROUND: YES
- XREF: YES
- POWER_EXTRACT: YES
- NETLIST_FORMAT: SPF
- NETLIST_FILE: *output_filename*
- NETLIST_INSTANCE_SECTION: YES
- NETLIST_CONNECT_SECTION: YES
- SHORT_PINS: NO
- NETLIST_GROUND_NODE_NAME: 0
- NETLIST_SUBCKT: YES

DSPF and SPEF File Structures

DSPF File Structure

Open Verilog International (OVI) publishes the DSPF standard.

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

SPEF File Structure

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

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

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

Chapter 20: Post-Layout Simulation: RC Network Reduction and Back-Annotation

Post-Layout Back-Annotation

Multi-Technology Simulation of 3D Integrated Circuit

Describes the HSPICE solution to simulate a modularized IC chip inside a 3D integrated circuit.

HSPICE provides multi-die interconnect analysis capability with its G-2012.06 release. A three-dimensional integrated circuit (3D-IC) is a single chip that integrates two or more layers of active electronic components both vertically and horizontally into a single circuit. All components on the layers communicate using on-chip signaling, whether vertically or horizontally.

The following sections discuss these topics:

- [Overview of HSPICE 3D-IC Simulation Netlist](#)
- [3D-IC Netlist Construct and Usage](#)
- [Transient Analysis and Alters Simulation Features](#)
- [Full Circuit Example](#)

Overview of HSPICE 3D-IC Simulation Netlist

The following are the general rules for analysis statements and simulation control statements:

- You must define all analysis statements and simulation control statements with the full 3D-IC instance hierarchy reference.
- Any analysis and simulation control statements referenced to subcircuit names require enhancements to identify the unique IC module with the `.module` construct label.

See [Full Circuit Example on page 690](#) for an example that references include and library files.

The HSPICE simulates a 3D-IC with existing methodology to simulate the modularized IC chip inside the full 3D-IC (single die) and uses enhanced commands and simulation controls specific to the process.

HSPICE keeps the single IC module netlist intact. It also allows customizing of the circuit properties for different instantiations of the same IC module such as `.TEMP`, `.OPTION TNOM`, `.OPTION SCALE`, `.OPTION GEOSHRINK`, and netlist parameters. This approach enables netlist interpretation and model interpretation controlling statements, such as `.IVTH`. This methodology maintains the instance full hierarchy to match the layout hierarchy for back-annotation.

The following sections discuss these topics:

- [HSPICE Netlist Definitions](#)
- [IC Module Usage](#)

HSPICE Netlist Definitions

The netlist scope includes:

- A device model that you can define by either the `.MODEL` command or `.SUBCKT` (macro-model) constructs.
- Device model referencing depends on the netlist static scope (the netlist definition inside the `.SUBCKT` construct).
- You reference netlist parameters based on dynamic scope (the full instance hierarchical path).

IC Module Usage

The HSPICE simulation allows you to define unique circuit properties as the default for each IC module in the following items:

- Circuit property definitions using commands and control options such as: `.OPTION SCALE`, `.OPTION GEOSHRINK`, `.TEMP` and so forth.
- Netlist parameter references using the `.PARAM` command.
- Model card definitions for both model card and macro-model forms.

- Verilog-A module definitions
- Subcircuit block definitions

3D-IC Netlist Construct and Usage

The 3D-IC construct employs two commands: `.MODULE` and `.ENDMODULE` to create a 3D-IC-specific netlist block. These commands enable you to define the unique IC module entities without name labels or circuit properties and to avoid collision between different IC modules. You can define the model reference static scope unique for the given IC module and define the unique IC module default entities and circuit properties. For definitions and details about all legal commands and options listed in the `.MODULE— .ENDMODULE` construct, see the [HSPICE Reference Manual: Commands and Control Options](#), Chapters 2 and 3.

Syntax

```
.MODULE label [BASE=base_module_label]  
...  
.ENDMODULE [label]
```

where: *label* consists of legal netlist commands and constructs, including circuit-topology definition statements and circuit-property definition statements such as:

- File inclusion commands `.LIB` and `.INCLUDE`.
- `.SUBCKT` constructs that contain legal netlist commands.
- `.HDL` (Verilog-A) commands.
- `.PARAM` commands.
- `.MODEL` commands.
- Scaling control options `.OPTION SCALE` and `.OPTION GEOSHRINK`. These options define the device scaling factor for each IC module such that all instances below the subckt carry these properties.
- Temperature controls `.TEMP` and `.OPTION TNOM`: These commands define the simulation temperature for each IC module such that all instances below the subckt carry these properties.

- `.GLOBAL` command: Defines the global node for each IC module. Thus, all nodes that reside below the subckts carry this node definition connect to this node within the IC module. For example, if `.GLOBAL` defines a node within the `.MODULE` construct, only the instances inside the subckts (defined within the same `.MODULE` construct) and subsequent nodes below the subckts can connect to the defined node without connecting through subckt ports.

Note: Even though HSPICE defines the `.GLOBAL` nodes for each IC module, the simulator only limits its reference the IC module. The nodes can be referenced from top level through the following syntax: `instance_name.global_node_label`.

- `.IVTH`: If you declare this command inside the `.MODULE` block, it applies to the model card defined within the same `.MODULE` construct only.

where: `base_module_label` argument allows you to define and inherit all of the content of the base module in the derived IC module without any IC module label. The derived IC module content can overwrite the base IC module content.

You can connect the module based global nodes explicitly at the top level such that, all instances instantiated with the IC module top subckt could have different top level connection.

For more information on accessing the global node inside a module from the top level, see `.MODULE` and `.CONNECT` commands in the *HSPICE® Reference Manual: Commands and Control Options*.

Illegal Netlist Commands, Controls and Constructs

The following may not be part of the contents of a `.MODULE` construct.

- Analysis feature statements.
- Output probing statements such as `.PRINT`, `.PROBE` and `.MEASURE`.
- Simulation feature statements, such as `.ALTER`, `.TRAN`, and Monte Carlo constructs.
- Simulation controlling options other than those explicitly stated in the Syntax section above.
- Conditional/logical statements such as `.IF` and `.ELSE`.

The following sections discuss these topics:

- [Scope Reference Rules](#)
- [Using Verilog-A Modules Within the .MODULE Scope](#)

- [Top-Level IC Module Reference](#)
- [Intrinsic Model Card Reference from a Top-Level Instance](#)
- [IC Module Instance-Specific Properties Scope](#)
- [Over-writing Circuit Properties](#)
- [Parameter Direct Access from Top-Level Netlist](#)

Scope Reference Rules

All entities and circuit properties can be applicable only to the netlist entities inside the same `.MODULE` and `.ENDMODULE` construct. In other words, the entities defined inside the `.MODULE` construct take higher precedence over those with the same name labels that are defined at the top level outside a `.MODULE` construct.

Examples

Example 1: This is a legal netlist in which the simulation nominal temperature for the `xtop1` block is 40. The nominal temperature for the rest of the circuit is 25. The device length for the `xtop1.m1` is $3.6e-008$. The device length for the `xtop2.m1` is $5.6e-008$.

```
.temp 25
.param ptop=5.6e-008
xtop1 ... top-module::top1
xtop2 ... top2
.module top-module
    .temp 40
    .param ptop=3.6e-008

    .subckt top1 ...
        m1 ... nmod l="ptop" w=2.7e-006 ...
    .ends top1
.endmodule top-module

.subckt top2 ...
    m1 ... nmod l="ptop" w=2.7e-006 ...
.ends top2
```

Example 2: Legal netlist with device references to the models defined within the same `.module` construct. The `xtop1.m1` references the model card with `vth0=0.38` while the `xtop2.m1` references the one with the `vth0=0.41`.

```
.model nmod nmos level=54 vth0=0.41 ...
```

Chapter 21: Multi-Technology Simulation of 3D Integrated Circuit

3D-IC Netlist Construct and Usage

```
xtop1 ... top::top1
xtop2 ... top2
.module top
    .model nmod nmos level=54 vth0=0.38
    ...
    .subckt top1
    m1 ... nmod l=3.6e-008 w=2.7e-006 ...
    .ends top1
.endmodule top
.subckt top2
    m1 ... nmod l=3.6e-008 w=2.7e-006 ...
.ends top2
```

Example 3: This *illegal* netlist shows how you must not reference undefined model cards, either in the same `.module` construct or at the top-level, even though the referenced model name is uniquely defined inside another `.module` construct.

```
xtop1 ... tmod1::top1
xtop2 ... tmod2::top2
.module tmod1
    .model nmod1 nmos level=54 vth0=0.45 ...
    .subckt top1
    m1 ... nmod2 l=3.6e-008 w=2.7e-006 ...
    .ends top1
.endmodule tmod1
.module tmod2
    .model nmod2 nmos level=54 vth0=0.41 ...
    .subckt top2
    m1 ... nmod1 l=3.6e-008 w=2.7e-006 ...
.ends top2
.endmodule tmod2
```

Example 4: This legal netlist shows the device reference to the models defined without the `.module` construct only because there is not a model defined inside the `.module` construct with the same name label. The `xtop1.m1` references the model card with `vth0=0.45` at the top level static scope. The

`xtop1.m2` references the model card with the `vth0=0.41`. The `xtop2.m1` references the model card with the `vth0=0.45` (same as the `xtop1.m1`).

```
.model nmod1 nmos level=54 vth0=0.45 ...
xtop1 ... top::top1
xtop2 ... top2
.module top
    .model nmod2 nmos level=54 vth0=0.41
    ...
    .subckt top1
    m1 ... nmod1 l=3.6e-008 w=2.7e-006
    ...
    m2 ... nmod2 l=3.6e-008 w=2.7e-006
    ...
    .ends top1
.endmodule top
    .subckt top2
        m1 ... nmod1 l=3.6e-008 w=2.7e-006 ...
    .ends top2
```

Using Verilog-A Modules Within the .MODULE Scope

You can use a Verilog-A (VA) module to uniquely define each IC module scope by using the `.HDL` statement inside the `.MODULE` construct. See [.HDL](#) in the *HSPICE Reference Manual: Commands and Control Options*. See [Using Verilog-A](#) in the *HSPICE User Guide: Advanced Analog Simulation and Analysis*.

The following rules apply for VA module references:

- Instances inside an IC module can reference the VA modules defined inside the same IC module and the VA modules defined without the IC-module scope (top level).
- Instances inside the IC module cannot cross-reference any VA module defined in a different IC module scope.
- The interposer level (top level) instance can reference the VA modules defined without the IC-module scope (top level) and VA modules defined inside any IC module with additional IC-module label references.

The VA module reference precedence is as follows:

1. VA module within the same IC module scope.
2. VA module without IC-module scope (top level).

VA Reference Usage Rules

Use the following HSPICE subcircuit instance syntax when you want to have a top level instance reference the VA module inside the IC module:

```
Xinstance_name interconnect VA_module_name parameters
```

Use the following Subcircuit Instance Syntax Extension:

```
Xinstance_name interconnect A_module_name[module_label::]  
VA_module_name
```

Examples, Reference Usage

Legal Netlist Example 1: VA module referenced with multiple VA module with the same name in different scopes. The `xtop1.x1` references the VA module, `my_inv`, defined inside the `top` IC module while the `xtop2.x1` references the one defined without IC module (top-level).

```
.HDL "my_va.va"      * "my_inv" VA module is defined.  
xtop1 ... top::top1  
xtop2 ... top2  
.module top  
.HDL "my_va2.va"    * "my_inv" VA module is defined.  
.subckt top1  
x1 ... my_inv ...  
.ends top1  
.endmodule top  
.subckt top2  
x1 ... my_inv ...  
.ends top2
```

Legal Netlist Example 2: VA module defined within the IC module and referenced from the top level instance. The `xr2` references the VA module, `resistor`, defined inside the "mod1" IC module while the `xr3` references to the one defined inside the `mod2` IC module.

```
// res.va  
`include "discipline.h"  
module resistor (a,b);  
inout a,b;  
electrical a,b;  
branch(a,b) res;  
parameter real R=1;  
analog begin  
I(res) <+ V(res)/R;  
end  
endmodule
```

```
// res_cap.va
`include "discipline.h"
module resistor (a,b);
inout a,b;
electrical a,b;
branch(a,b) res;
parameter real R=1;
parameter real C=1e-15;
analog begin
I(res) <+ V(res)/R + C*ddt(V(res));
end
endmodule

*** Top level netlist
.module mod1
.hdl './res.va'
...
.endmodule

.module mod2
.hdl './res_cap.va'
...
.endmodule

vsource 1 0 pwl 0n 0 10n 3
r1 1 2 10K
xr2 2 3 mod1::resistor R=10K
xr3 3 0 mod2::resistor R=10K C=1p
...
c1 2 0 1uF
.tran 1n 1u
.end
```

Example 3: *Illegal* Netlist—This example contains a VA module cross-reference to the VA module defined in a different IC module scope. It is illegal to have the `xtop.x1` in the IC module “`top1`” cross-references the VA module, `inv`, defined inside the `top2` IC module.

```
xtop ... top1::top1
.module top1
.subckt top1
x1 ... inv ...
.ends top1
.endmodule top
.module top2
.HDL "my_va.va" * "inv" VA module is defined.
.endmodule top
```

Top-Level IC Module Reference

Reference Rules for top-level IVC module references:

A top-level instance instantiates the IC module top subcircuit block that is defined inside the `.MODULE` construct and requires a reference to the module label.

HSPICE Subcircuit Instance Syntax:

Xinstance_name interconnect subcircuit_name parameters

HSPICE Subcircuit Instance Syntax Extension:

*Xinstance_name interconnect
[module_label::]subcircuit_name parameters*

Examples, Top-Level References

Example 1: A legal netlist in which the “x_{top1}” instance instantiates the “top” subcircuit defined inside the “t_{mod}” module (with the “m₁” instance) while the “x_{top2}” instance instantiates the “top” subcircuit defined without the “.module” construct (with the “m₂” instance).

```
xtop1 ... tmod::top
xtop2 ... top
.module tmod
.subckt top
m1 ...
.ends top
.endmodule tmod
.subckt top
m2 ...
.ends top
```

Example 2: An *illegal* netlist because you may not reference a subcircuit definition label with an incorrect “.module” label even though the subcircuit definition name is unique throughout the netlist.

```
xtop1 ... tmod1::top2
xtop2 ... tmod2:top1
xtop3 ... top1
xtop4 ... top2
.module tmod1
.subckt top1
m1 ...
.ends top1
.endmodule tmod1
.module tmod2
.subckt top2
m2 ...
.ends top2
.endmodule tmod2
```

Intrinsic Model Card Reference from a Top-Level Instance

Interpretation rules: When the intrinsic model card is referenced from the top-level instance, all interpretation references are based on the top-level rules, such as parameter passing and scaling.

HSPICE Transistor Instance Syntax

Minstance_name interconnect model_card_name parameters

HSPICE Transistor Instance Syntax Extension:

*Minstance_name interconnect
[module_label::]model_card_name parameters*

Example

```
m1 ... tmod::nmos ... .module tmod
.model nmos nmos ...
.endmodule tmod
```

IC Module Instance-Specific Properties Scope

The .modulevar and .endmodulevar block enables you to define the unique IC module entities for each top-level instance instantiation.

The following defines the HSPICE `.MODULEVAR` and `.ENDMODULEVAR` constructs.

Syntax `.MODULEVAR` Block

```
.MODULEVAR label  
...  
.ENDMODULEVAR [label]
```

where: `label` can be any legal netlist statements and constructs, such as:

- `.PARAM`
- `.OPTION`
- `.TEMP`
- `.LIB` and `.INCLUDE` to include files containing legal statements inside the `.MODULEVAR` construct

Illegal netlists statements and constructs include:

- Any circuit topology constructs and statements, such as `.SUBCKT` and instance statements
- Other illegal statements and constructs such as the `.MODULE` construct.

`.MODULEVAR` Construct Usage

The circuit entities allowed are parameters and properties such as those which these statements contain:

- Parameters - `.PARAM`
- Circuit properties such as: `.TEMP`, `.OPTION SCALE`, and `.OPTION GEOSHRINK`.

Note: It is *illegal* to define any circuit topology construct or statements, such as device element statements and the `.subckt` construct.

- The `.modulevar` label can only be referenced by the `modulevar=` parameter as part of the `Xinstance_name` statement.
- Circuit property precedence rules:

The overall circuit properties reference precedence is as follows:

1. Defined inside the `.modulevar` construct.
2. Defined inside the `.module` construct.
3. Defined at the top-level netlist (outside any `".module"` construct).

- Any circuit properties not defined inside the lower precedence scope, are treated as additional circuit properties for the referenced IC module.

Over-writing Circuit Properties

The top-level IC module instance can overwrite any circuit properties with predefined a `.modulevar` construct label.

Syntax

HSPICE Subcircuit Instance Syntax:

```
Xinstance_name interconnect subcircuit_name parameters
```

HSPICE Subcircuit Instance Syntax Extension:

```
Xinstance_name interconnect [module_label::]  
    subcircuit_name [modulevar=modulevar_label] parameters
```

Examples: Instance-Specific Properties Reference

Example 1: Legal Netlist specifying the properties as follows:

Instance	Nominal Temperature	Device Length
xtop1.m1	25	5e-008
xtop2.m1	40	3e-008
xtop3.m1	25	5e-008
xtop4.m1	10	1e-008

```
xtop1 ... tmod::top modulevar="top-inst"  
xtop2 ... tmod::top  
xtop3 ... top modulevar="top-inst"  
xtop4 ... top
```

```
.temp 10  
.param ptop=1e-008  
.module tmod  
    .temp 40  
    .param ptop=3e-008  
  
    .subckt top ...  
    m1 ... nmod l="ptop" w=2.7e-006 ...  
    .ends top  
.endmodule tmod  
  
.modulevar top-inst  
    .temp 25
```

Chapter 21: Multi-Technology Simulation of 3D Integrated Circuit
 3D-IC Netlist Construct and Usage

```
.param ptop=5e-008
.endmodulevar top-inst

.subckt top ...
m1 ... nmod l="ptop" w=3.7e-006 ...
.ends top
```

Example 2: This netlist shows top-down parameter passing of the following properties (by way of `.option parhier=global`):

Instance	Nominal Temperature	Device Length
xtop1.m1	25	3e-008
xtop2.m1	25	5e-008
xtop3.m1	40	5e-008
xtop4.m1	40	1e-008
xtop5.m1	25	1e-008
xtop6.m1	25	8e-008
xtop7.m1	10	4e-008
xtop8.m1	10	8e-008

And the bottom-up parameter passing of the following properties using `.OPTION PARHIER=local`.

Instance	Nominal Temperature	Device Length
xtop1.m1	25	8e-008
xtop2.m1	25	4e-008
xtop3.m1	40	8e-008
xtop4.m1	40	6e-008
xtop5.m1	25	7e-008
xtop6.m1	25	2e-008
xtop7.m1	10	7e-008
xtop8.m1	10	9e-008

```
xtop1 ... tmod::top modulevar="top-inst"
xtop2 ... tmod::top modulevar="top-inst"
+ ptop=4e-008
xtop3 ... tmod::top
xtop4 ... tmod::top ptop=6e-008

xtop5 ... top modulevar="top-inst"
```

```
xtop6 ... top modulevar="top-inst"  
+ ptop=2e-008  
xtop7 ... top  
xtop8 ... top ptop=9e-008  
  
.temp 10  
.param ptop=1e-008  
  
.module tmod  
  .temp 40  
  .param ptop=3e-008  
  
  .subckt top ...  
  .param ptop=8e-008  
  m1 ... nmod l="ptop" w=2.7e-006 ...  
  .ends top  
.endmodule tmod  
  
.modulevar top-inst  
  .temp 25  
  .param ptop=5e-008  
.endmodulevar top-inst  
  
.subckt top ...  
.param ptop=7e-008  
m1 ... nmod l="ptop" w=3.7e-006 ...  
.ends top
```

Example 3: *Illegal* netlist due to referencing undefined ".modulevar" construct label.

```
xtop1 ... tmod::top modulevar="top-inst"  
.temp 10  
.param ptop=1e-008  
.module tmod  
  .temp 40  
  .param ptop=3e-008  
  
  .subckt top ...  
  m1 ... nmod l="ptop" w=2.7e-006 ...  
  .ends top  
.endmodule tmod
```

Parameter Direct Access from Top-Level Netlist

The following syntax extension allows you to access the parameters defined inside the `.module` or the `.modulevar` construct.

Syntax

```
.param param_label="module_label::param_label"  
.param modulevar_label="modulevar_label::param_label"
```

Example

```
.module top  
.param top_p=5  
.endmodule  
.modulevar top_inst  
.param top_inst_p=8  
.endmodulevar  
.param p1="top::top_p" * p1 = 5  
.param p2="top_inst::top_inst_p" * p2 = 8
```

Transient Analysis and Alters Simulation Features

The current 3D-IC solution applies to transient analysis only.

The following sections cover these topics:

- [Sweeps for Temperature and Parameters](#)
- [.ALTER Features](#)

Sweeps for Temperature and Parameters

You can simulate 3D-IC modules using temperature and parameter sweeps in a transient analysis:

- [Temperature Sweep in a Transient Simulation](#)
- [Parameter Sweep in a Transient Simulation](#)

Temperature Sweep in a Transient Simulation

This reference extension allows you to reference the `.temp` command or `.option tnom` for a temperature SWEEP in a transient simulation.

Keyword temp Syntax Extension

```
module_label::temp  
modulevar_label::temp
```

Example 1: Single IC module temperature SWEEP.

```
.module top
```

```
.temp 25
.endmodule
* Sweep "top1::temp" from 40 to 100 with each step increase by 10.
* Total simulation: 6
.tran 0ns 10ns SWEEP top::temp 40 100 10
```

Example 2: Multiple IC module temperature SWEEP with "data."

```
.module top1
.temp 25
.endmodule
.module top2
.temp 40
.endmodule
.tran 0ns 1ns SWEEP data=tempdata
* Sweep top1::temp from 25 to 35 and top2::temp from 40 to 45
* Total simulation: 6
.data tempdata top1:temp top2::temp
25 40
30 40
35 40
25 45
30 45
35 45
.enddata
```

Parameter Sweep in a Transient Simulation

This reference extension allows you to sweep through the given parameters defined in either the `.module` or the `.modulevar` constructs.

Reference Syntax

```
module_label::param_label
modulevar_label::param_label
```

Example1: 1.Single IC module parameter SWEEP

```
.module top
.param p1= 25
.endmodule
* Sweep "top::p1" from 25 to 35 with each step increase by 5.
* Total simulation: 3
.tran 0ns 10ns SWEEP top::p1 25 35 5
```

Example 2: Multiple IC module parameter SWEEP with "data."

```
.module top1
.param p1= 25
.endmodule
.module top2
.param p2= 40
.endmodule
.tran 0ns 1ns SWEEP data=paramdata
* Sweep top1::p1 from 25 to 30 and top2::p2 from 40 to 45
* Total simulation: 4
.data paramdata top1:p1 top2::p2
25 40
30 40
25 45
30 45
.enddata
```

.ALTER Features

The 3D-IC usage for the HSPICE `.ALTER` instance-statement replacement scheme is the same as the existing replacement scheme for a standard device simulation. Two HSPICE commands aid in this process: `.DEL MODULE` and `.DEL MODELEVAR`.

You modify or add to existing `.module` or `.modulevar` constructs by redefining the `.module` or `.modulevar` constructs with previously-defined label names.

The following modification rules apply:

- The content defined with the new constructs for the existing label constructs only modifies the content to the same label construct content.
- If the statements exist in the original construct, the new contents replace the contents of the original construct.
- If the statements are undefined in the original construct, the content is added to the label construct.
- If any reference content becomes unavailable for instance reference, you need to modify the instance statements by redefining them as replacements to the existing top-level instance statements.

See the following topics:

- [.DEL MODULE Command for .ALTER Blocks](#)
- [.DEL MODULEVAR Command for .ALTER Blocks](#)

.DEL MODULE Command for .ALTER Blocks

The `.DEL MODULE` command undefines the previously defined `.module` construct and prepares it for redefinition. You can only define the `.DEL MODULE` construct inside `.ALTER` blocks and all the contents previously defined with the specified `.MODULE` label are no longer referenced.

Syntax

```
.del module existing_module_label
```

Example: Redefining the `top` label.

```
.module top
    .subckt inv
        m1...
        m2...
    .ends inv
.endmodule
xtop ... top::inv
.alter s1
    .del module top * Undefine the "top" IC module.
    .module top      * Redefine the "top" IC module
        .subckt inv
            xm1 ... nch
            xm2 ... pch
        .ends inv
        .subckt nch
            ...
        .ends
        .subckt pch
            ...
        .ends
    .endmodule
.end
```

.DEL MODULEVAR Command for .ALTER Blocks

The `.DEL MODULEVAR` command undefines the previously defined `.MODULEVAR` construct and prepares it for redefinition. You can only define the `.DEL MODULEVAR` construct inside `.ALTER` blocks. With the `.DEL MODULEVAR` statement defined, all the contents previously defined with the specified `.MODULEVAR` label can no longer be referenced.

Chapter 21: Multi-Technology Simulation of 3D Integrated Circuit

Full Circuit Example

Syntax

```
.del modulevar existing_modulevar_label

.module top
    .subckt inv
        m1... w=p l=0.02u
    .ends inv
.endmodule

.modulevar ic1
    .param p=0.05u
.endmodulevar
.param p=0.06u
xtop ... top::inv modulevar="ic1"

.alter s1
    .del modulevar ic1 * "xtop.m1" will have "0.06u" as
width.

.alter s2
    .del modulevar ic1
    .modulevar ic1
        .param p=0.07u * "xtop.m1" will have "0.07u" as width.
    .endmodulevar
.end
```

Full Circuit Example

The following full circuit example includes these characteristics:

- This 3D-IC consists of multiple single dies with the same design and technology node.
- This 3D-IC has die-based simulation corners or circuit properties.

The example shows how you can use a single IC module netlist for a single IC set of memory simulation properties from the file `memory.lib`.

```
.lib TT
    .param ... * parameters for the simulation corner TT.
    ...
.endlib TT
.lib FF
    .param... * parameters for the simulation corner FF
    ...
.endlib FF
```



```
.lib SS
    .param... * parameters for the simulation corner SS
    ...
.endlib SS
.lib models
    .models ...          * model cards for the memory IC.
    .subckt nch_mac ... * macro-models for the memory IC.
    ...
    .ends nch_mac
.endlib models
```

The netlist then draws on single IC memory circuit definitions from the file `memory.sp`:

```
* Top level circuit from single memory IC module
.subckt 1G_mem_top ...
    .temp 100
    xbank1 ... bank
    ...
.ends 1G_mem_top
* Other subcircuit definitions.
.subckt bank ...
    ...
.ends bank
```

The 3D-IC Memory netlist draws from the `3D_mem.sp` file:

```
* global control and parameters for 3D IC simulation.
.temp -40
* 1st IC memory module (fast corner)
xmem1 .... 1GMem::1G_mem_top
* 2nd IC memory module (slow corner)
xmem2 .... 1GMem::1G_mem_top
* 3rd IC memory module (typical - default)
xmem3 .... 1GMem::1G_mem_top
* 4th IC memory module
xmem4 .... 1G_mem_top

* top level control logic block.
x5 .... memory_control
.subckt memory_control
    ...
.ends memory_control
* Netlist definitions from the original single IC circuit.
.include "memory.sp"

.module 1GMem
    * Default control and parameters
    .lib "memory.lib" TT
```

Chapter 21: Multi-Technology Simulation of 3D Integrated Circuit

Full Circuit Example

```
* Default single IC memory properties
.temp 25
...
* Models for the circuit elaborations in the memory circuit.
.lib "memory.lib" models
* Netlist definitions from the original single IC circuit.
.include "memory.sp"
.endmodule 1GMem
```

Part 4: Variation and Statistical Analysis

This Part contains the following chapters/topics.

- [Chapter 22, Monte Carlo—Traditional Flow Statistical Analysis](#)
- [Chapter 23, Variability Analysis Using the Variation Block](#)
- [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#)
- [Chapter 25, Mismatch Analyses](#)
- [Chapter 26, Monte Carlo Data Mining](#)
- [Chapter 27, DC Sensitivity Analysis and Variation Block](#)

Monte Carlo—Traditional Flow Statistical Analysis

Describes the traditional statistical analysis features supported by HSPICE and HSPICE RF.

The D-2010.03 release enabled advanced sampling capabilities for use with the traditional Monte Carlo format. Earlier releases did not support advanced sampling capabilities. The advanced sampling capabilities are described in this chapter. See [Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo on page 819](#) for more information. The features described in this chapter differ from, and are generally a subset of, the enhanced statistical analysis features described in [Chapter 23, Variability Analysis Using the Variation Block](#), and [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#).

HSPICE ships hundreds of examples for your use. See [Listing of Demonstration Input Files](#) for paths to demo files. See [Variability Examples](#) to find traditional Monte Carlo demonstration files.

For information on bisection in conjunction with Monte Carlo, see [Chapter 30, Timing Analysis Using Bisection](#).

This chapter discusses the following topics:

- [Application of Statistical Analysis](#)
- [Analytical Model Types](#)
- [Circuit and Model Temperature Simulation](#)
- [Worst-Case Analysis](#)
- [Traditional Monte Carlo Simulations](#)
- [Traditional Monte Carlo Analysis Characteristics](#)
- [Advanced Sampling Methods](#)

- [Worst-Case and Monte Carlo Sweep Example](#)
- [Global and Local Variations with Monte Carlo](#)

Application of Statistical Analysis

When you design an electrical circuit, it must meet tolerances for the specific manufacturing process. The electrical yield is the number of parts that meet the electrical test specifications. Overall process efficiency requires maximum yield. To analyze and optimize the yield, HSPICE and HSPICE RF support statistical techniques and observe the effects of variations in element and model parameters.

The basic functionality of Monte Carlo analysis is to simulate the effects of variations on circuit performance. When measurements are executed, the results from all the samples form a distribution. This distribution has characteristics which can be described in statistical terms: mean, standard deviation, and so on. The statistical terms are calculated by the simulator and reported at the end of the run listing.

The main reason to run Monte Carlo analysis is to find out whether the circuit will still have acceptable yield if it is subject to variations. Comparing the statistical results reported by the simulator can basically answer this question; however, in many cases the shape of the distribution and how it relates to the specification are of interest. For visual inspection, it is useful to create a histogram. The histogram helps in determining whether the variations in the performance look *normal* by considering not only the bell shape of the Normal distribution, but also unexpected behavior like outliers and gaps.

The next question typically is whether the distribution is well centered with respect to the specification, and what the predicted yield is. Another question which needs to be asked in this context is whether the circuit is over-designed. Over-designed means that the margins are too big for the particular characteristic. This might be at the expense of secondary properties, like power and area, which affect cost ultimately.

These are the more traditional ways of looking at the results from Monte Carlo analysis. However, more information can be accessed with more sophisticated tools. (See [Chapter 23, Variability Analysis Using the Variation Block](#), [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#), and [Chapter 26, Monte Carlo Data Mining](#).)

Analytical Model Types

To model parametric and statistical variation in circuit behavior, use:

- `.PARAM` statement to investigate the performance of a circuit as you change circuit parameters. See the `.PARAM` statement in the *HSPICE Reference Manual: Commands and Control Options* for details on this command.
- Temperature variation analysis to vary the circuit and component temperatures, and compare the circuit responses. You can study the temperature-dependent effects of the circuit, in detail.
- Monte Carlo analysis when you know the statistical standard deviations of component values to center a design. This provides maximum process yield, and determines component tolerances.
- Worst-case corner analysis when you know the component value limit to automate quality assurance for:
 - Basic circuit function
 - Process extremes
 - Quick estimation of speed and power tradeoffs
 - Best-case and worst-case model selection
 - Parameter corners
 - Library files
- Data-driven analysis for cell characterization, response surface, or Taguchi analysis (see [Performing Digital Cell Characterization](#)), which automates characterization of cells and calculates the coefficient of polynomial delay for timing simulation. You can simultaneously vary any number of parameters and perform an unlimited number of analyses. This analysis uses an ASCII file format so that HSPICE can automatically generate parameter values. This analysis can replace hundreds or thousands of HSPICE simulation runs.
- Use yield analyses to modify:
 - DC operating points
 - DC sweeps
 - AC sweeps
 - Transient analysis

- Use Custom WaveView to generate scatter plots from the operating point analysis or a family of curve plots for DC, AC, and transient analysis.
- Use `.MEASURE` statements to save results for delay times, power, or any other characteristic extracted in a `.MEASURE` statement. HSPICE generates a table of results in an `.mt#` file in ASCII format. You can analyze the numbers directly or read this file into WaveView to view the distributions. Also, if you use `.MEASURE` statements in a Monte Carlo or data-driven analysis, then the HSPICE output file includes the following statistical results in the listing:

- Mean $\frac{x_1 + x_2 + \dots + x_n}{N}$

- Sigma $\sqrt{\text{Variance}}$

- Variance $\frac{(x_1 - \text{Mean})^2 + \dots + (x_n - \text{Mean})^2}{N - 1}$

- Average Deviation $\frac{|x_1 - \text{Mean}| + \dots + |x_n - \text{Mean}|}{N - 1}$

Circuit and Model Temperature Simulation

Temperature affects *all* electrical circuits. [Figure 108](#) shows the key temperature parameters associated with circuit simulation:

- Model reference temperature – You can model different models at different temperatures. Each model has a `TREF` (temperature reference) parameter.
- Element junction temperature – Each resistor, transistor, or other element generates heat so that an element is hotter than the ambient temperature.
- Part temperature – At the system level each part has its own temperature.
- System temperature – A collection of parts form a system, which has a local temperature.
- Ambient temperature – The ambient temperature is the air temperature of the system.

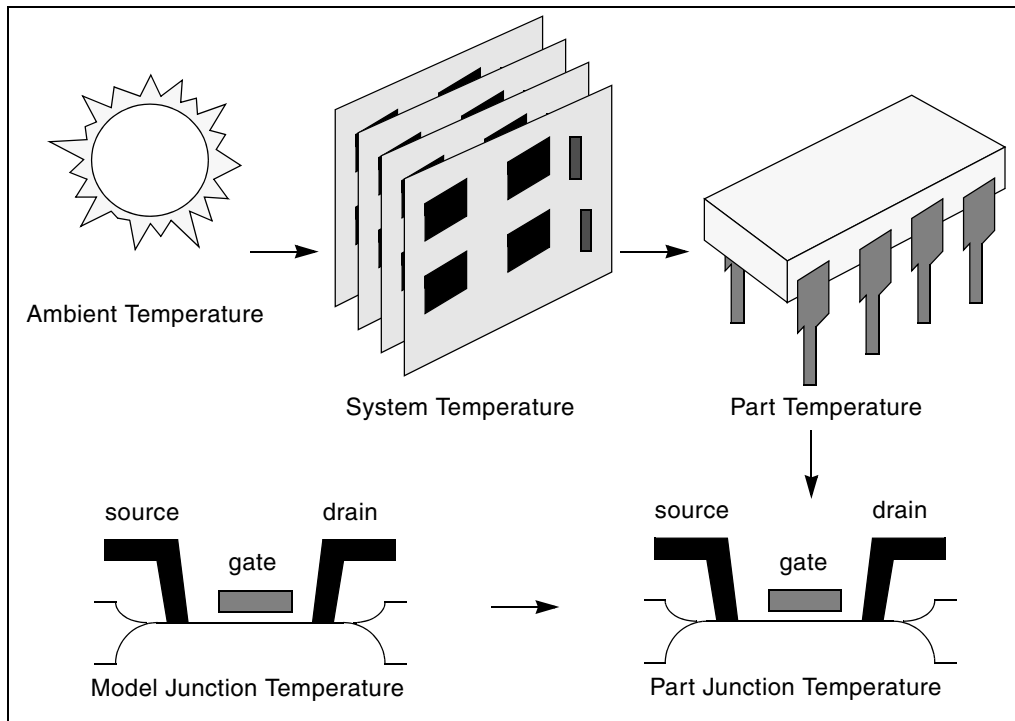


Figure 108 Part Junction Temperature Sets System Performance

HSPICE and HSPICE RF calculate temperatures as differences from the ambient temperature:

$$T_{\text{ambient}} + \Delta_{\text{system}} + \Delta_{\text{part}} + \Delta_{\text{junction}} = T_{\text{junction}}$$

$$I_{\text{ds}} = f(T_{\text{junction}}, T_{\text{model}})$$

Every element includes a DTEMP keyword, which defines the difference between junction and ambient temperature.

Example

The following example uses DTEMP in a MOSFET element statement:

```
M1 drain gate source bulk Model_name W=10u L=1u DTEMP=+20
```

Temperature Analysis

You can specify three temperatures:

- Model reference temperature specified in a `.MODEL` statement. The temperature parameter is usually `TREF`, but can be `TEMP` or `TNOM` in some models. This parameter specifies the temperature, in °C, at which HSPICE measures and extracts the model parameters. Set the value of `TNOM` in an `.OPTION` statement. Its default value is 25°C.
- Circuit temperature that you specify using a `.TEMP` statement or the `TEMP` parameter. This is the temperature, in °C, at which HSPICE simulates all elements. To modify the temperature for a particular element, use the `DTEMP` parameter. The default circuit temperature is the value of `TNOM`.
- Individual element temperature, which is the circuit temperature, plus an optional amount that you specify in the `DTEMP` parameter.

To specify the temperature of a circuit in a simulation run, use either the `.TEMP` statement, or the `TEMP` parameter in the `.DC`, `.AC`, or `.TRAN` statements. HSPICE compares the circuit simulation temperature that one of these statements sets against the reference temperature that the `TNOM` option sets. `TNOM` defaults to 25°C, unless you use the `SPICE` option, which defaults to 27°C. To calculate the derating of component values and model parameters, HSPICE uses the difference between the circuit simulation temperature, and the `TNOM` reference temperature.

Elements and models within a circuit can operate at different temperatures. For example, a high-speed input/output buffer that switches at 50 MHz is much hotter than a low-drive NAND gate that switches at 1 MHz. To simulate this temperature difference, specify both an element temperature parameter (`DTEMP`), and a model reference parameter (`TREF`). If you specify `DTEMP` in an element statement, the element temperature for the simulation is:

```
element temperature=circuit temperature + DTEMP
```

Specify the `DTEMP` value in the element statement (resistor, capacitor, inductor, diode, BJT, JFET, or MOSFET statement), or in a subcircuit element. Assign a parameter to `DTEMP`, then use the `.DC` statement to sweep the parameter. The `DTEMP` value defaults to zero.

If you specify `TREF` in the model statement, the model reference temperature changes (`TREF` overrides `TNOM`). Derating the model parameters is based on the difference between circuit simulator temperature and `TREF` (instead of `TNOM`).

.TEMP Statement

To specify the temperature of a circuit for a simulation, use the `.TEMP` statement.

Worst-Case Analysis

Circuit designers often use worst-case analysis when designing and analyzing MOS and BJT IC circuits. To simulate the worst case, set all variables to their 2- or 3-sigma worst-case values. Because several independent variables rarely attain their worst-case values simultaneously, this technique can lead to over-designing the circuit. However, this analysis is useful as a fast check.

Model Skew Parameters

The HSPICE device models include physically measurable model parameters. The circuit simulator uses parameter variations to predict how an actual circuit responds to extremes in the manufacturing process. Physically measurable model parameters are called *skew* parameters because they skew from a statistical mean to obtain predicted performance variations.

Examples of skew parameters are the difference between the drawn and physical dimension of metal, postillion, or active layers, on an integrated circuit.

Generally, you specify skew parameters independently of each other, so you can use combinations of skew parameters to represent worst cases. Typical skew parameters for CMOS technology include:

- X_L – Polysilicon CD (critical dimension of the poly layer, representing the difference between drawn and actual size).
- XW_n, XW_p – Active CD (critical dimension of the active layer, representing the difference between drawn and actual size).
- TOX – Thickness of the gate oxide.
- RSH_n, RSH_p – Resistivity of the active layer.
- $DELVTO_n, DELVTO_p$ – Variation in threshold voltage.

You can use these parameters in any level of MOS model, within the HSPICE device models. The $DELVTO$ parameter shifts the threshold value. HSPICE adds this value to VTO for the Level 3 model, and adds or subtracts it from $VFB0$ for the BSIM model. [Table 74](#) shows whether HSPICE adds or subtracts

deviations from the average.

Table 74 Sigma Deviations

Type	Parameter	Slow	Fast
NMOS	XL	+	-
	RSH	+	-
	DELVTO	+	-
	TOX	+	-
	XW	-	+
PMOS	XL	+	-
	RSH	+	-
	DELVTO	-	+
	TOX	+	-
	XW	-	+

HSPICE selects skew parameters based on the available historical data that it collects either during fabrication or electrical test. For example, HSPICE collects the *XL skew* parameter for poly CD during fabrication. This parameter is usually the most important skew parameter for a MOS process.

[Figure 109 on page 703](#) is an example of data that historical records produce.

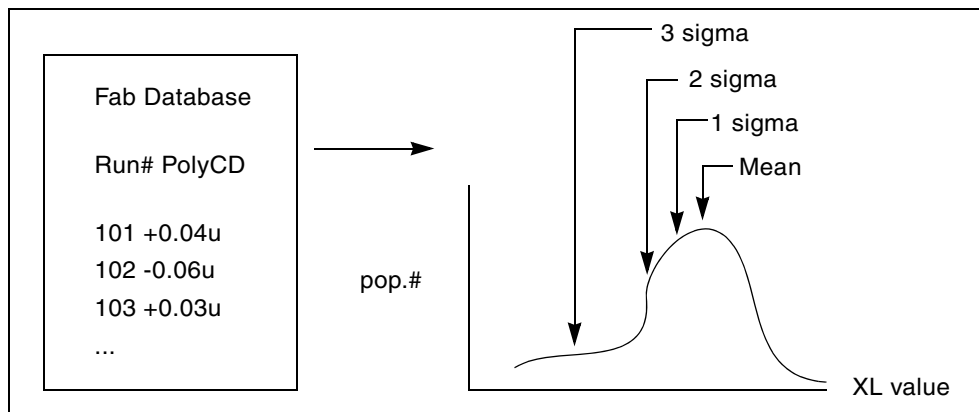


Figure 109 Historical Records for Skew Parameters in a MOS Process

Using Skew Parameters

Figure 110 shows how to create a worst-case corners library file for a CMOS process model. Specify the physically measured parameter variations so that their proper minimum and maximum values are consistent with measured current (I_{DS}) variations. For example, HSPICE can generate a 3-sigma variation in I_{DS} from a 2-sigma variation in physically measured parameters.

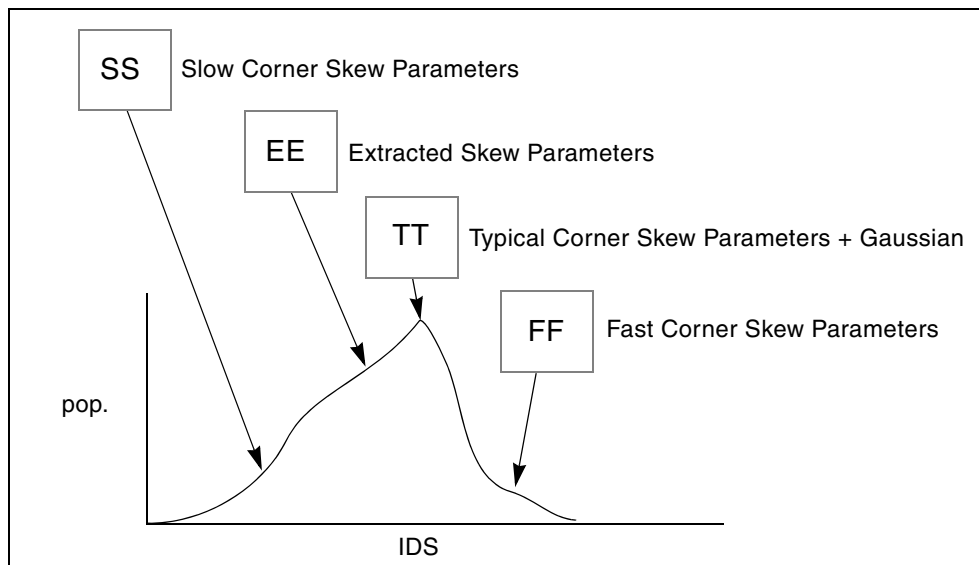


Figure 110 Worst-Case Corners Library File for a CMOS Process Model

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis Worst-Case Analysis

The `.LIB` (library) statement, and the `.INCLUDE` or `.INC` (include file) statement, access the models and skew. The library contains parameters that modify `.MODEL` statements. The following example of `.LIB` features both worst-case and statistical-distribution data by using model skew parameters. In statistical distribution, the median value is the default for all non-Monte Carlo analysis.

Example

```
.LIB TT
$TYPICAL P-CHANNEL AND N-CHANNEL CMOS LIBRARY DATE:3/4/91
$ PROCESS: 1.0U CMOS, FAB22, STATISTICS COLLECTED 3/90-2/91
$ following distributions are 3 sigma ABSOLUTE GAUSSIAN

.PARAM
$ polysilicon Critical Dimensions
+ polycd=AGAUSS(0,0.06u,1) xl='polycd-sigma*0.06u'
$ Active layer Critical Dimensions
+ nactcd=AGAUSS(0,0.3u,1) xwn='nactcd+sigma*0.3u'
+ pactcd=AGAUSS(0,0.3u,1) xwp='pactcd+sigma*0.3u'
$ Gate Oxide Critical Dimensions (200 angstrom +/- 10a at 1
$ sigma)
+ toxcd=AGAUSS(200,10,1) tox='toxcd-sigma*10'

$ Threshold voltage variation
+ vtoncd=AGAUSS(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=AGAUSS(0,0.05v,1) delvtop='vtopcd+sigma*0.05'

.INC '/usr/meta/lib/cmos1_mod.dat' $ model include file
.ENDL TT
.LIB FF
$HIGH GAIN P-CH AND N-CH CMOS LIBRARY 3SIGMA VALUES

.PARAM TOX=230 XL=-0.18u DELVTON=-.15V DELVTOP= 0.15V
.INC '/usr/meta/lib/cmos1_mod.dat' $ model include file

.ENDL FF
```

The `/usr/meta/lib/cmos1_mod.dat` include file contains the model.

```
.MODEL NCH NMOS LEVEL=2 XL=XL TOX=TOX DELVTO=DELVTON . .
.MODEL PCH PMOS LEVEL=2 XL=XL TOX=TOX DELVTO=DELVTOP . .
```

Note: The model keyname (left) equals the skew parameter (right).
Model keys and skew parameters can use the same names.

Skew File Interface to Device Models

Skew parameters are model parameters for transistor models or passive components. A typical device model set includes:

- MOSFET models for all device sizes by using an automatic model selector.
- RC wire models for polysilicon, metal1, and metal2 layers in the drawn dimension. Models include temperature coefficients and fringe capacitance.
- Single-diode and distributed-diode models for N+, P+, and well (includes temperature, leakage, and capacitance based on the drawn dimension).
- BJT models for parasitic bipolar transistors. You can also use these for any special BJTs, such as a BiCMOS for ECL BJT process (includes current and capacitance as a function of temperature).
- Metal1 and metal2 transmission line models for long metal lines.
- Models must accept elements. Sizes are based on a drawn dimension. If you draw a cell at 2μ dimension and shrink it to 1μ , the physical size is 0.9μ . The effective electrical size is 0.8μ . Account for the four dimension levels:
 - Drawn size
 - Shrunk size
 - Physical size
 - Electrical size

Most simulator models scale directly from *drawn* to *electrical* size. HSPICE MOS models support all four size levels ([Figure 111](#)).

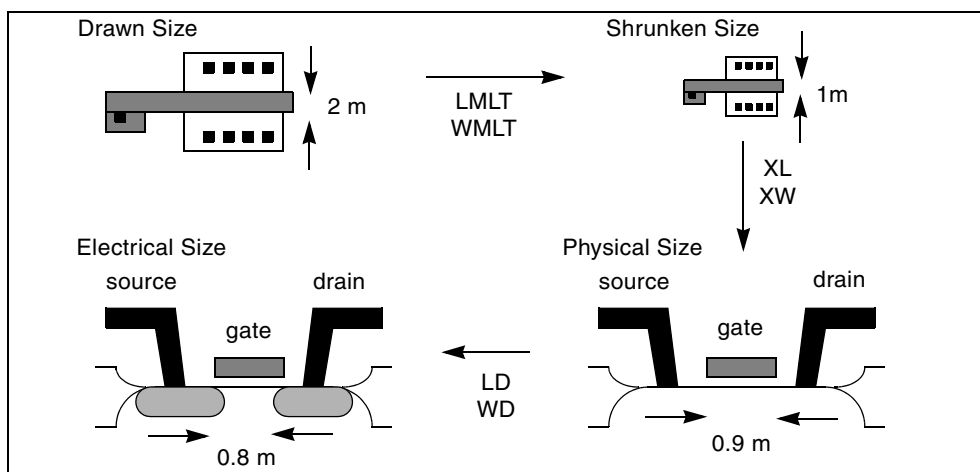


Figure 111 Device Model from Drawn to Electrical Size

Traditional Monte Carlo Simulations

A high-level overview of HSPICE traditional Monte Carlo analysis follows. Later sections provide more in-depth information.

The basic premise of a Monte Carlo analysis is that you are going to parameterize one or more circuit variables, vary those values by a randomized amount from the norm, and run HSPICE a pre-determined number of times. Each run is called a sweep and will generate tabular or plot data as specified by the user. Measurements are also typically used to look at circuit operating conditions from run to run.

You can randomize anything that can be set with a parameter or variable. Examples include things as diverse as a simple resistor value, a model parameter for a MOSFET, or the length of a transmission line.

Values can be varied using three basic statistical variations: uniform, limit, and Gaussian. Using those methods, you choose the nominal value and the absolute or relative variation. You can optionally supply the standard deviation and a multiplier.

Note that HSPICE does not run a nominal simulation with traditional Monte Carlo. For traditional Monte Carlo, the first sample is index 1; it is not the nominal value. For example: `monte=10`, you get 10 samples (1-10).

However, you can run Monte Carlo using the Variation Block features (where the first run is the nominal case) with AGAUSS (traditional) style definitions. See [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#).

The following sections discuss these topics:

- [Basic Syntax](#)
- [Local and Global Parameter Variation](#)
- [Exception for Model Parameters](#)
- [Starting Values and Seeds](#)
- [Other Monte Carlo Control Options](#)
- [Monte Carlo Analysis in HPP](#)

Basic Syntax

The basic syntax of a Monte Carlo analysis includes three elements:

1. Defining a parameter with one of the distribution keywords

2. Using the parameter in your netlist as the value for an element or model parameter
3. Including the SWEEP and MONTE keywords in the analysis statement

Syntax used for a Monte Carlo analysis

```
.MODEL mname ModelType ([level=val]  
+ [keyword1=val1] [keyword2=val2]  
+ [keyword3=val3] [LOT distribution value]  
+ [DEV distribution value]...)
```

Consider the following example. In this simple RC charging circuit, the value of `r_one` has a nominal value of 1K and is varied by 400 ohms for 10 iterations.

RC charging circuit:

```
.option post probe  
*define a parameter called "resval" with an absolute, uniform  
distribution  
.param resval=aunif(1000,400)  
vsrc_one 1 0 5v  
r_one 1 2 resval  
c_one 2 0 1u  
.ic 2=0  
*specify 10 Monte Carlo iterations  
.tran 1e-5 5e-3 sweep monte=10  
*measure to find when 1 time constant (.632*vdd) occurs  
.meas tran tc when v(2)='.632*5'  
*create plots of the charging curve and resistor values  
.probe v(2) par(resval)  
.end
```

The resulting waveforms are called *multi-member*. Plotting one signal displays the curves from all the runs.

Local and Global Parameter Variation

A common source of confusion is local and global parameter variation. The key is that each time you use a parameter, it gets assigned a new random value. In this case the resistors are said to be under local variation. Take the following examples:

```
.param resval=aunif(1000,400)  
r_one 1 2 resval  
r_two 2 3 resval  
r_three 3 4 resval
```

In this case, all three resistors will get unique, random values. (However, as discussed in the next section, there is an exception to this rule.) If you want to

set a group of components to the same random value, assign an intermediate parameter first:

```
.param resval=aunif(1000,400)
.param my_resval=resval
r_one 1 2 my_resval
r_two 2 3 my_resval
r_three 3 4 my_resval
```

In the second example, the assignment of a random value is only done once, then used three times. In this case, the resistors are said to be under global variation.

Exception for Model Parameters

Because a model definition is only done once, the behavior described in the previous section would assign the same parameter value to all devices referencing that model. To overcome this, `.OPTION MODMONTE` lets the user decide if all instances of a device should get the same or unique model parameters.

Starting Values and Seeds

Another source of confusion is the starting value. If you run the same Monte Carlo simulation twice, the results will be identical because HSPICE/HSPICE RF always uses the same “seed” value for the first run. If it randomized the seed by default, it would be difficult to determine whether changes you made to the circuit and topology were the result of your changes or the new random values. You can specify a seed or have HSPICE pick a random seed with `.OPTION SEED` if that behavior is desired.

Other Monte Carlo Control Options

- `.OPTION MONTECON` - Some random parameter assignments can cause HSPICE not to converge. This parameter is used to decide whether to terminate a simulation or move to the next run if convergence fails.
- `.OPTION RANDGEN` - Use this option to specify the type of random number generator used.
- `.OPTION MCBRIEF` - Controls how HSPICE outputs Monte Carlo parameters and generates or suppresses output files.

See [HSPICE Netlist Simulation Control Options](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Monte Carlo Analysis in HPP

It should be noted here that HSPICE Precision Parallel (HPP) supports *enhanced* SRS and *not* the traditional SRS. Hence, while comparing the results

between HSPICE without HPP and HSPICE with HPP Monte Carlo analysis, you need to keep in mind the differences explained in this chapter. See [Differences between Traditional SRS and VB SRS on page 724](#). (For details on HPP, see [HSPICE Precision Parallel \(-hpp\) on page 61](#).)

Traditional Monte Carlo Analysis Characteristics

Monte Carlo analysis uses a random number generator to create the following types of functions:

- Gaussian parameter distribution
 - Relative variation - Variation is a ratio of the average.
 - Absolute variation - Adds variation to the average.
 - Bimodal - Multiplies distribution to statistically reduce nominal parameters.
- Uniform parameter distribution
 - Relative variation - Variation is a ratio of the average.
 - Absolute variation - Adds variation to the average.
 - Bimodal - Multiplies distribution to statistically reduce nominal parameters.
- Random limit parameter distribution
 - Absolute variation - Adds variation to the average.
 - Monte Carlo analysis randomly selects the *min* or *max* variation.

The value of the MONTE analysis keyword determines how many times to perform operating point, DC sweep, AC sweep, or transient analysis.

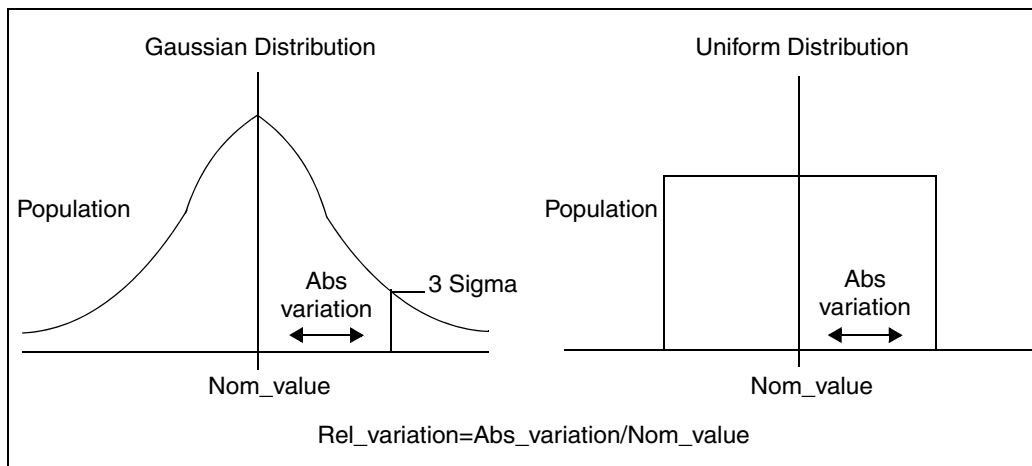


Figure 112 Monte Carlo Distribution

The following sections discuss these topics:

- [Monte Carlo Setup](#)
- [Monte Carlo Output](#)
- [.PARAM Distribution Function](#)
- [Monte Carlo Parameter Distribution](#)
- [Non-Gaussian Probability Distribution Functions](#)
- [Monte Carlo Examples](#)

Monte Carlo Setup

To set up a Monte Carlo analysis, use the following HSPICE statements:

- `.PARAM` statement - Sets a model or element parameter to a Gaussian, Uniform, or Limit function distribution.
- `.DC`, `.AC`, or `.TRAN` analysis—enables MONTE.
- `.MEASURE` statement - Calculates the output mean, variance, sigma, and standard deviation.
- `.MODEL` statement - Sets model parameters to a Gaussian, Uniform, or Limit function distribution.

Select the type of analysis to run, such as operating point, DC sweep, AC sweep, or TRAN sweep.

Operating Point

```
.PARAM dummy=1  
.DC dummy 1 1 1 MONTE=[num] firstrun=[num]
```

-or-

```
.DC MONTE=list[(] [num1:num2] [num3] [num5:num6] [num7] [)]
```

DC Sweep

```
.DC vin 1 5 0.25 sweep MONTE=val [firstrun=num1]
```

-or-

```
.DC vin 1 5 0.25 sweep MONTE=list[(] [num1:num2] [num3]  
+ [num5:num6] [num7] [)]
```

AC Sweep

```
.AC dec 10 100 1meg sweep MONTE=val [firstrun=num1]
```

-or-

```
.AC vin 1 5 0.25 sweep MONTE=list[(] [num1:num2] [num3]  
+ [num5:num6] [num7] [)]
```

TRAN Sweep

```
.TRAN 1n 10n sweep MONTE=val [firstrun=num1]
```

-or-

```
.TRAN vin 1 5 0.25 sweep MONTE=list[(] [num1:num2] [num3]  
+ [num5:num6] [num7] [)]
```

The *val* value specifies the number of Monte Carlo iterations to perform. A reasonable number is 30. The statistical significance of 30 iterations is quite high. If the circuit operates correctly for all 30 iterations, there is a 99 percent probability that over 80 percent of all possible component values operate correctly. The relative error of a quantity, determined through Monte Carlo analysis, is proportional to $val^{-1/2}$.

The *firstrun* values specify the desired number of iterations. HSPICE runs from *num1* to *num1+val-1*. The number after *firstrun* can be a parameter. You can write only one number after *list*. The colon represents “from ... to ...”. Specifying only one number makes HSPICE run only at the one specified point.

Examples

Example 1: HSPICE runs from the 90th to 99th Monte Carlo iteration:

```
.tran ln 10 sweep monte=10 firstrun=90
```

You can write more than one number after `list`. The colon represents “from ... to ...”. Specifying only one number makes HSPICE run only at that single point.

Example 2: HSPICE begins running at the 10th iteration, then continues from the 20th to the 30th, then jumps to the 40th, and finally, runs from the 46th to 72nd Monte Carlo iteration.

```
.tran ln 10n sweep monte=list(10 20:30 40 46:72)
```

Example 3: `Firstrun` option, where the number of samples is 1000.

```
.tran ... sweep monte =1000, firstrun = 10
```

Example 4: `List` option, where the number of samples is 100.

```
.tran ... sweep monte = list(10:100)
```

Monte Carlo Output

The following probing commands provide these benefits:

- `.MEASURE` statements are the most convenient way to summarize the results.
- `.PRINT` statements generate tabular results, and print the values of all Monte Carlo parameters.
- `.OPTION MCBRIEF` determines the output types of the random parameters during Monte Carlo analysis to improve output performance; select from several values for different results. See [.OPTION MCBRIEF](#) in the *HSPICE Reference Manual: Commands and Control Options*.
- If one iteration is out of specification, you can obtain the component values from the tabular listing. A detailed re-simulation of that iteration might help identify the problem.
- Custom WaveView superimposes all iterations as a single plot. So you can analyze each iteration individually.

.PARAM Distribution Function

This section describes how to assign a .PARAM parameter in Monte Carlo analysis. For a general description of the .PARAM statement, see the .PARAM command in the *HSPICE Reference Manual: Commands and Control Options*.

You can assign a .PARAM parameter to the keywords of elements and models, and assign a distribution function to each .PARAM parameter. HSPICE recalculates the distribution function each time that an element or model keyword uses a parameter. When you use this feature, Monte Carlo analysis can use a parameterized schematic netlist without additional modifications.

Syntax

```
.PARAM xx=UNIF(nominal_val, rel_variation
+ [, multiplier])
.PARAM xx=AUNIF(nominal_val, abs_variation
+ [, multiplier])
.PARAM xx=GAUSS(nominal_val, rel_variation, num_sigmas
+ [, multiplier])
.PARAM xx=AGAUSS(nominal_val, abs_variation, num_sigmas
+ [, multiplier])
.PARAM xx=LIMIT(nominal_val, abs_variation)
```

Argument	Description
xx	Distribution function calculates the value of this parameter.
UNIF	Uniform distribution function by using relative variation.
AUNIF	Uniform distribution function by using absolute variation.
GAUSS	Gaussian distribution function by using relative variation.
AGAUSS	Gaussian distribution function by using absolute variation
LIMIT	Random-limit distribution function by using absolute variation. Adds +/- <i>abs_variation</i> to <i>nominal_val</i> based on whether the random outcome of a -1 to 1 distribution is greater than or less than 0.
nominal_val	The nominal (mean, average, or center) value of the distribution function. Non-Monte Carlo analyses use this value as the default.

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis

Traditional Monte Carlo Analysis Characteristics

Argument	Description
abs_variation	<p>Specifies the absolute variation about the nominal value for the distribution function.</p> <ul style="list-style-type: none">▪ AUNIF distributions will vary about nominal_val by +/- abs_variation.▪ AGAUSS distributions will vary about the nominal_val according to the number of standard deviations (num_sigmas) this absolute variation represents.
rel_variation	<p>Specifies the relative variation about the nominal value for the UNIF and GAUSS distribution functions.</p> <ul style="list-style-type: none">▪ UNIF distributions will vary about the nominal value by +/- (nominal_val*rel_variation).▪ GAUSS distributions will vary about the nominal value with nominal_val*rel_variation being equal to the number of standard deviations given by num_sigmas.
num_sigmas	<p>Describes Gaussian distributions by a mean value (nominal_val) and a standard deviation (sigma) value.</p> <p>The specified abs_variation or rel_variation must therefore be converted into a standard deviation value. The num_sigmas parameter allows you to specify how many standard deviations (sigmas) are represented by the abs_variation or rel_variation value. With num_sigmas=1, the variation specified corresponds to one standard deviation (one sigma).</p> <ul style="list-style-type: none">▪ For GAUSS, the standard deviation of the Gaussian distribution function is given by: $\text{Sigma} = \text{nominal_val} * \text{rel_variation} / \text{num_sigmas}$▪ For AGAUSS, the standard deviation of the Gaussian distribution function is given by: $\text{Sigma} = \text{abs_variation} / \text{num_sigmas}$ <p>Set num_sigmas=1 if you wish to specify variations in terms of root mean square (rms) values. Set num_sigmas to larger values to specify variations in terms of peak or peak-to-peak values with a certain error tolerance.</p> <p>For example, with num_sigmas=6.180, your input variation value will represent a peak-to-peak variation with 99.9 percent probability.</p> <p>With num_sigmas=9.507, your input variation value will represent a peak-to-peak variation with 99.9999 percent probability.</p>
multiplier	<p>If you do not specify a multiplier, the default is 1. HSPICE recalculates many times and saves the largest deviation. The resulting parameter value might be greater than or less than <i>nominal_val</i>. The resulting distribution is bimodal.</p>

Example 1

In this example, each resistor has a unique variation.

```
.param mc_var=AGAUSS(0,1,3)    $ +/-1 absolute swing or  
                                $ +/-100% relative swing  
.param val='1000*(1+mc_var)'  
v_vin vin 0 dc=1 ac=.1  
r1 vin 0  '1000*(1+mc_var)'  
r2 vin 0  '1000*(1+mc_var)'
```

Example 2

In this example, each resistor has an identical variation.

```
.param mc_var=AGAUSS(0,1,3)    $ +/- 20% swing  
.param val='1+mc_var'  
v_vin vin 0 dc=1 ac=.1  
r1 vin 0  '1000*val'  
r2 vin 0  '1000*val'
```

Example 3

In this example, local variations are applied to an instance parameter by assigning randomly generated variations directly to each instance parameter. Each resistor r1 through r3 receives randomly different resistance values during each Monte Carlo run.

```
.param r_local=AGAUSS(...)  
r1 1 2 r=r_local  
r2 3 4 r=r_local  
r3 5 6 r=r_local
```

Example 4

In this example, global variations are applied to an instance parameter by assigning the variation to an intermediate parameter before assigning it to each instance parameter. Each resistor r1 through r3 receives the same random resistance value during each Monte Carlo run.

```
.param r_random=AGAUSS(...)  
.param r_global=r_random  
r1 1 2 r=r_global  
r2 3 4 r=r_global  
r3 5 6 r=r_global
```

Monte Carlo Parameter Distribution

Each time you use a parameter, Monte Carlo analysis calculates a new random variable.

- If you do not specify a Monte Carlo distribution, then HSPICE assumes the nominal value.
- If you specify a Monte Carlo distribution for only one analysis, HSPICE uses the nominal value for all other analyses.

You can assign a Monte Carlo distribution to all elements that share a common model. The actual element value varies according to the element distribution. If you assign a Monte Carlo distribution to a model keyword, then all elements that share the model use the same keyword value. You can use this feature to create double element and model distributions.

For example, the MOSFET channel length varies from transistor to transistor by a small amount that corresponds to the die distribution. The die distribution is responsible for offset voltages in operational amplifiers, and for the tendency of flip-flops to settle into random states. However, all transistors on a die site vary according to the wafer or fabrication run distribution. This value is much larger than the die distribution, but affects all transistors the same way. You can specify the wafer distribution in the MOSFET model to set the speed and power dissipation characteristics.

Non-Gaussian Probability Distribution Functions

In traditional Monte Carlo analysis, there are only five different distributions:

- Uniform distribution, absolute
- Uniform distribution, relative
- Gaussian distribution, absolute
- Gaussian distribution, relative
- Limit distribution

While no user-defined distributions are available, you can describe functions such as those shown in [Figure 113 on page 717](#) using the methodologies described in this section.

In HSPICE, samples from the distributions given in [Figure 113](#) can be created exactly for the Case 1 (top) function using the CDF()—Cumulative Distribution

Function construct in Variation Block. The other two cases can be approximated using a PWL function for $F(x)$, which are the illustrations on the right-hand side.

The distributions can also be sampled exactly in both Variation Block and in the traditional Monte Carlo format by using the “probability inverse” and defining the relationship in an expression. Let u be a sample from the uniform random number generator over the range $[0, 1]$. This is the default behavior with the traditional Monte Carlo style. You would add 0.5 with the Variation Block uniform generator, $U()$, because the variables are sampled in $[-0.5, 0.5]$ to have zero mean.

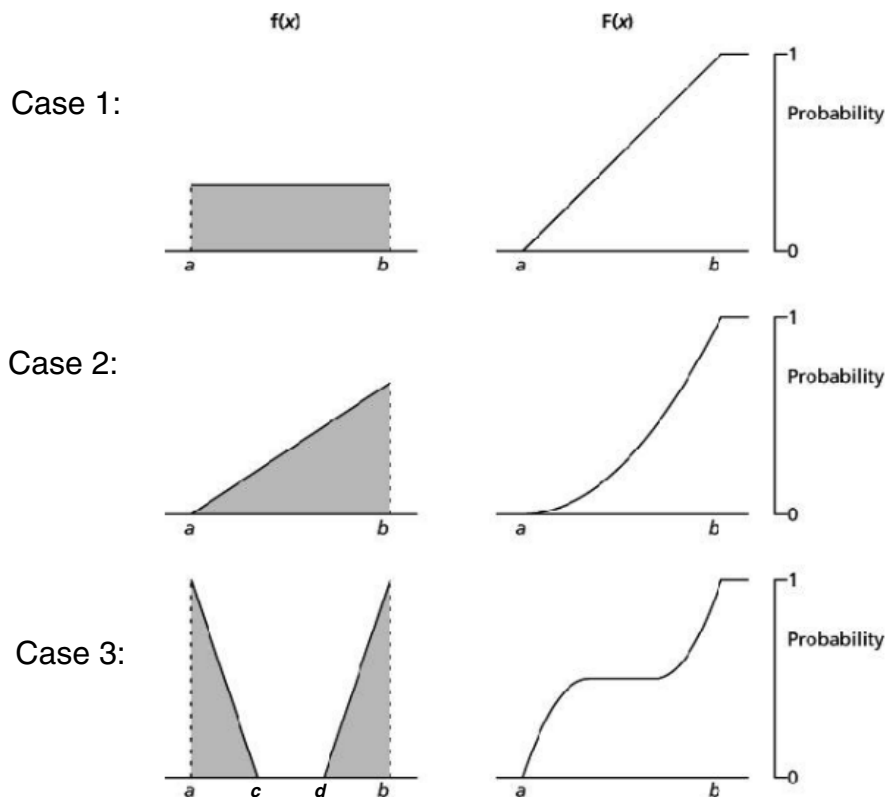


Figure 113 Non-Gaussian functions

Map u to the y-axis on the right side of the figures in Figure 113. Project the value onto the curve horizontally, and then down. This gives a value, say x . Then x is a random sample from the desired distribution. Mathematically, it works as $x = F^{-1}(u)$. For instance:

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis

Traditional Monte Carlo Analysis Characteristics

Case 1:

$$x = a + (b - a) \cdot u$$

Case 2:

$$x = a + (b - a) \cdot \sqrt{u}$$

Case 3:

$$x = a + (b - a) \cdot \sqrt{u} \text{ if } u \leq 0.5$$

$$x = d + (b - d) \cdot \sqrt{u} \text{ if } u \geq 0.5$$

Here, c and d are the locations of the two additional coordinates between a and b that define the gap in the distribution.

Monte Carlo Examples

Gaussian, Uniform, and Limit Functions

You can find the sample netlist for this example in the following directory:
`$installdir/demo/hspice/apps/mont1.sp.`

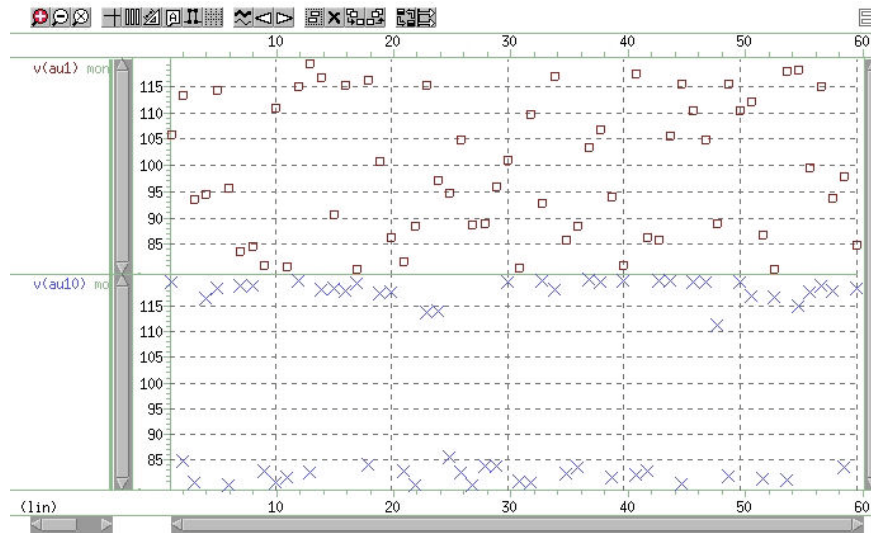


Figure 114 Uniform Functions

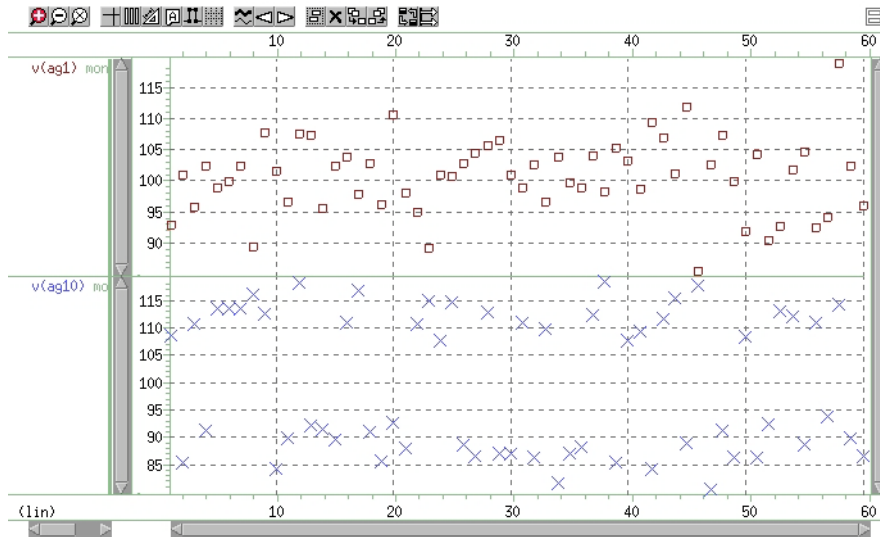


Figure 115 Gaussian Functions



Figure 116 Limit Functions

Major and Minor Distribution

In MOS IC processes, manufacturing tolerance parameters have both a major and a minor statistical distribution.

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis

Traditional Monte Carlo Analysis Characteristics

- The major distribution is the wafer-to-wafer and run-to-run variation. It determines electrical yield.
- The minor distribution is the transistor-to-transistor process variation. It is responsible for critical second-order effects, such as amplifier offset voltage and flip-flop preference.

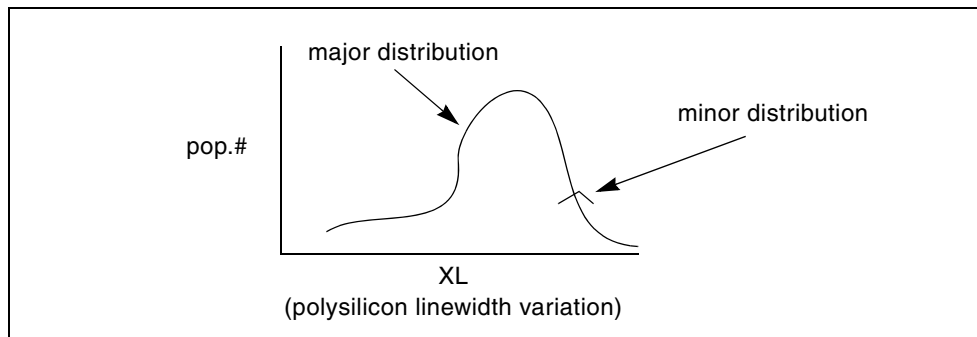


Figure 117 Major and Minor Distribution of Manufacturing Variations

The following example is a Monte Carlo analysis of a DC sweep in HSPICE. Monte Carlo sweeps the VDD supply voltage from 4.5 volts to 5.5 volts.

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/apps/mondc_a.sp
```

- The M1 through M4 transistors form two inverters.
- The nominal value of the LENGTH parameter sets the channel lengths for the MOSFETs, which are set to 1 μ in this example.
- All transistors are on the same integrated circuit die. The LEFF parameter specifies the distribution—for example, a ± 5 percent distribution in channel length variation at the ± 3 -sigma level.
- Each MOSFET has an independent random Gaussian value.

The PHOTO parameter controls the difference between the physical gate length and the drawn gate length. Because both n-channel and p-channel transistors use the same layer for the gates, Monte Carlo analysis sets XPHOTO distribution to the PHOTO local parameter. XPHOTO controls photolithography for both NMOS and PMOS devices, consistent with manufacturing physics.

RC Time Constant

This simple example shows uniform distribution for resistance and capacitance. It also shows the resulting transient waveforms for 10 different random values.

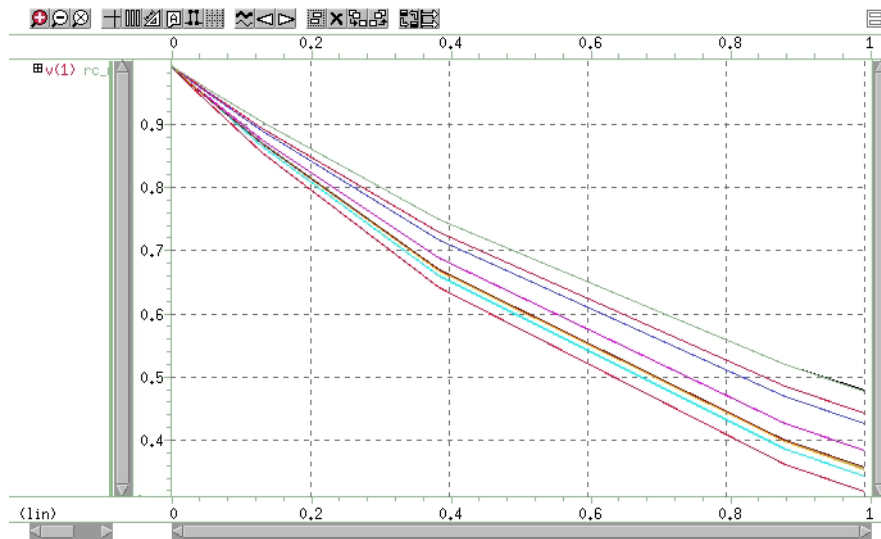


Figure 118 Monte Carlo Analysis of RC Time Constant

You can find the sample netlist for this example in the following directory:
`$installdir/demo/hspice/apps/rc_monte.sp`

Switched-Capacitor Filter Design

Capacitors used in switched-capacitor filters consist of parallel connections of a basic cell. Use Monte Carlo techniques in HSPICE to estimate the variation in total capacitance. The capacitance calculation uses two distributions:

- Minor (element) distribution of cell capacitance from cell-to-cell on a single die.
- Major (model) distribution of the capacitance from wafer-to-wafer or from manufacturing run-to-run.

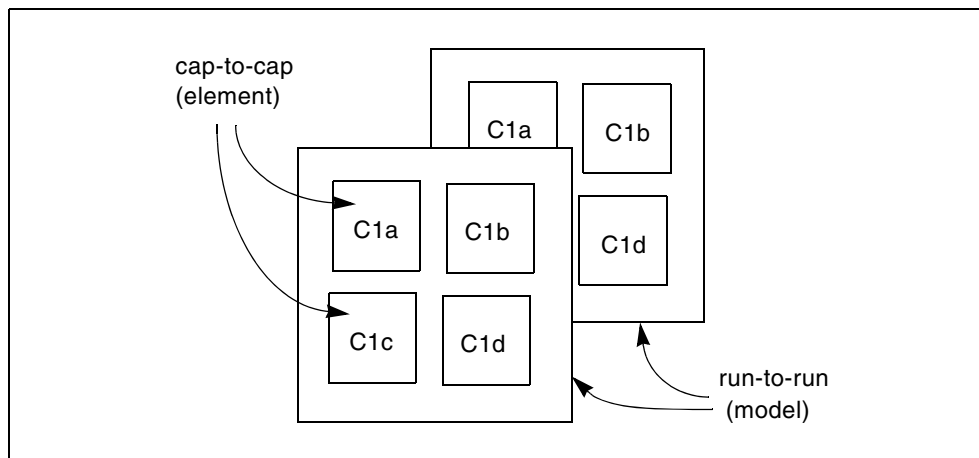


Figure 119 Monte Carlo Distribution

You can approach this problem from physical or electrical levels.

- The physical level relies on physical distributions, such as oxide thickness and polysilicon line width control.
- The electrical level relies on actual capacitor measurements.

Physical Approach:

Use the following steps for the physical approach:

1. Use a local variation in polysilicon to control the variation in capacitance for adjacent cells, because oxide thickness control is excellent for small areas on a single wafer.
2. Define a local polysilicon line-width variation and a global (model-level) poly line-width variation. In this example:
 - The local polysilicon line width control for a line 10 μm wide, manufactured with process A, is $\pm 0.02 \mu\text{m}$ for a 1-sigma distribution.
 - The global (model-level) polysilicon line-width control is much wider; use 0.1 μm for this example.
 - The global oxide thickness is 200 angstroms with a ± 5 angstrom variation at 1 sigma.
 - The cap element is square with local poly variation in both directions.
 - The cap model has two distributions:
 - Poly line-width distribution

— Oxide thickness distribution

- The model poly distribution is half the physical per-side values.

The effective length is:

```
Leff=Ldrawn - 2  
C1a 1 0 CMOD W=ELPOLY L=ELPOLY  
C1b 1 0 CMOD W=ELPOLY L=ELPOLY  
C1C 1 0 CMOD W=ELPOLY L=ELPOLY  
C1D 1 0 CMOD W=ELPOLY L=ELPOLY  
$ 10U POLYWIDTH,0.05U=1SIGMA  
$ CAP MODEL USES 2*MODPOLY .05u= 1 sigma  
$ 5angstrom oxide thickness AT 1SIGMA  
.PARAM ELPOLY=AGAUSS(10U,0.02U,1)  
+ MODPOLY=AGAUSS(0,.05U,1)  
+ POLYCAP=AGAUSS(200e-10,5e-10,1)  
.MODEL CMOD C THICK=POLYCAP DEL=MODPOLY
```

Electrical Approach:

The electrical approach assumes no physical interpretation, but requires a local (element) distribution and a global (model) distribution. In this example:

- You can match the capacitors to ± 1 percent for the 2-sigma population.
- The process can maintain a ± 10 percent 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
```

Advanced Sampling Methods

Background: Adding Enhanced SRS to Traditional Monte Carlo

HSPICE developed a new type of variation definition format called Variation Block (VB) to address issues that the traditional style or the AGAUSS style was neither intuitive nor flexible to code. Variation Block was introduced with the 2005.03 HSPICE release (see [Chapter 23, Variability Analysis Using the Variation Block](#)). The VB format enabled different types of sampling methods, such as LHS, Sobol, Niederreiter, OFAT, Factorial, and External, along with

Simple Random Sampling (SRS). The output of Monte Carlo analysis with VB style definitions is far more comprehensive than with the AGAUSS style Monte Carlo. Later it was decided to support the features associated with the VB style within the AGAUSS style. As a result, beginning with HSPICE 2010.03, the AGAUSS style is supported both in the traditional Monte Carlo flow and the VB flow. The user must invoke the latter by entering the following: `option sampling_method=SRS`. The other sampling algorithms are also supported with the AGAUSS style.

HSPICE supports statistical modeling of variability parameters for Gaussian, Uniform, and Limit distribution. The keywords GAUSS, AGAUSS, UNIF, AUNIF, and LIMIT can be used in a SPICE netlist to represent these distributions. By using these keywords and following certain rules, the user can code Global and Local variations, which together, constitute parametric variations. This method is commonly known as AGAUSS style variation definition. Until recently, HSPICE supported only the Simple Random Sampling (SRS) algorithm for Monte Carlo simulation with this type of variation definition.

Differences between Traditional SRS and VB SRS

If the user does not specify `.option sampling_method=SRS` in the netlist, HSPICE invokes the traditional SRS algorithm along with the AGAUSS style statistical models. Before making any comparison, users should note that a few differences exist between traditional SRS and the SRS invoked with the `option sampling_method` (hereafter called enhanced SRS). (Note that HPP requires enhanced SRS.) These differences are explained in the following sections:

- [Random Number Generators \(RNGs\)](#)
- [Datamining](#)

Random Number Generators (RNGs)

The RNGs used in the case of traditional SRS are different from those used for enhanced SRS. Therefore, a one-to-one match is not possible unless the user sets `.option RANDGEN=1` in the traditional SRS flow to match the RNGs.

In the traditional AGAUSS flow, there is no nominal simulation (that is, simulation without any variation on model parameters). The Monte Carlo flow with the `option sampling_method` executes nominal simulation for the first Monte index. Thus, when you compare the traditional SRS values for each sample with that of enhanced SRS, start the comparison from the first index of the traditional SRS to the second index of enhanced SRS, and so on.

The results may vary slightly because of normal limit, controlled by the option `Normal_Limit`, when using enhanced SRS. (See [Chapter 24, Monte Carlo](#)

[Analysis—Variation Block Flow](#).) In some cases the order of value assignment to independent random variables may differ because of slight differences in traditional and enhanced front-end flows.

Datamining

The traditional SRS flow generates a limited statistical summary (mean, median, and sigma, and so on) in the output listing file.

The enhanced SRS flow generates the following:

- Enhanced statistical summary and variable screening report in the `*.mpp0` file.
- Independent random variable values for each sample in the `*.mc0` file.
- Element variation contribution information to back-annotate in Synopsys' Custom Designer in the `*.annotate` file.
- Random number variable values (data block format) for user-controlled external block simulation of samples that resulted in minimum and maximum values for the measured values in the `*.corner` file.
- Partial `*.mpp0` file for traditional SRS (without variable screening report).

See [Chapter 26, Monte Carlo Data Mining](#) for more information.

Invoking Advanced SRS

To invoke sampling methods such as Latin Hypercube, Factorial, OFAT, or low-discrepancy sequences in a Monte Carlo netlist, enter `.OPTION SAMPLING_METHOD=name_of_method`.

The syntax is as follows:

```
.OPTION
SAMPLING_METHOD=SRS | LHS | Factorial | OFAT | Sobol | Niederreiter | External
```

The methods and their brief descriptions are listed below. For detailed discussion of these methods, see [Sampling Options](#) and [Comparison of Sampling Methods](#) in [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#).

Method	Brief Description
SRS	(Default) Simple random sampling performed in traditional HSPICE Monte Carlo method.

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis

Worst-Case and Monte Carlo Sweep Example

Method	Brief Description
LHS	Latin Hypercube sampling; efficient for large number of variable parameters (used with <code>.OPTION REPLICATES</code>).
Factorial	Factorial sampling: <ul style="list-style-type: none">▪ Evaluates the circuit response at the extremes of variable ranges to get an idea of the worst- and best-case behavior.▪ Creates polynomial response surface approximations.
OFAT	One-Factor-at-a-Time sampling; useful for sensitivity studies and for constructing low-order response surface approximations.
Sobol	Sobol sampling uses low-discrepancy sequences (LDS); LDS sample points are more frequently distributed compared to LHS and the sampling error is lower. Sobol is used with a sampling dimension of 40 or less.
Niederreiter	LDS sampling sequence is useful as a method for cases of a sampling dimension up to 318. If that number is exceeded, HSPICE switches to the default SRS sampling method.
External	Executes a data set of externally created perturbations. External sampling allows design and process exploration tools to run statistical experiments with the variables for each sample under their full control.

Use `.OPTION REPLICATES` after selecting `.OPTION SAMPLING_METHOD=LHS`. The `REPLICATES` option runs replicates of the Latin Hypercube samples. For more information, see [Latin Hypercube Sampling \(LHS\) on page 803](#).

Worst-Case and Monte Carlo Sweep Example

The following example measures the delay and the power consumption of two inverters. Additional inverters buffer the input and load the output.

This netlist contains commands for two sets of transient analysis: parameter sweep from -3 to +3-sigma, and a Monte Carlo analysis. It creates one set of output files (`.mt0` and `.tr0`) for the sigma sweep, and one set (`mt1` and `tr1`) for Monte Carlo.

```
$ inv.sp sweep mosfet -3 sigma to +3 sigma, use measure output
.param vref=2.5 sigma=0
```

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis
Worst-Case and Monte Carlo Sweep Example

```

.global 1
vcc 1 0 5.0
vin in 0 pwl 0,0 0.2n,5
x1 in 2 inv
x2 2 3 inv
x3 3 out inv
x4 out 4 inv
.macro inv in out
    mn out in 0 0 nch w=10u l=1u
    mp out in 1 1 pch w=10u l=1u
.eom
.param mult1=1
+ polycd=AGAUSS(0,0.06u,1)    xl='polycd-sigma*0.06u'
+ nactcd=AGAUSS(0,0.3u,1)   xwn='nactcd+sigma*0.3u'
+ pactcd=AGAUSS(0,0.3u,1)   xwp='pactcd+sigma*0.3u'
+ toxcd=AGAUSS(200,10,1)    tox='toxcd-sigma*10'
+ vtoncd=AGAUSS(0,0.05v,1) delvton='vtoncd-sigma*0.05'
+ vtopcd=AGAUSS(0,0.05v,1) delvtop='vtoncd+sigma*0.05'
+ rshncd=AGAUSS(50,8,1)    rshn='rshncd-sigma*8'
+ rshpcd=AGAUSS(150,20,1)  rshp='rshpcd-sigma*20'
* level=28 example model
.model nch nmos
+ level=28 lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl  xw=xwn tox=tox delvto=delvton rsh=rshn
...
.model pch pmos
+ level=28 lmlt=mult1 wmlt=mult1 wref=22u lref=4.4u
+ xl=xl  xw=xwp tox=tox delvto=delvtop rsh=rshp
+ ld=0.08u wd=0.2u acm=2 ldif=0 hdif=2.5u
+ rs=0 rd=0 rdc=0 rsc=0 rsh=rshp js=3e-04 jsw=9e-10
...
* transient with sweep
.tran 20p 1.0n  sweep sigma -3 3 .5
.meas s_delay trig v(2) val=vref fall=1
+      targ v(out) val=vref fall=1
.meas s_power rms power
* transient with Monte Carlo
.tran 20p 1.0n  sweep monte=100
.meas m_delay trig v(2) val=vref fall=1
+      targ v(out) val=vref fall=1
.meas m_power rms power
.probe tran v(in) v(1) v(2) v(3) v(4)
.end

```

Transient Sigma Sweep Results

The plot in [Figure 120](#) shows the family of transient analysis curves for the transient sweep of the sigma parameter from -3 to +3 from the file `inv.tr0`. In the sweep, HSPICE uses the values of sigma to update the skew parameters, which in turn modify the actual NMOS and PMOS models.

Operating-Point Results in Transient Analysis

If you want to get OP results after every Monte Carlo simulation in transient analysis, you can add the option `opfile` to the netlist. OP results output to the file `*.dp0`.

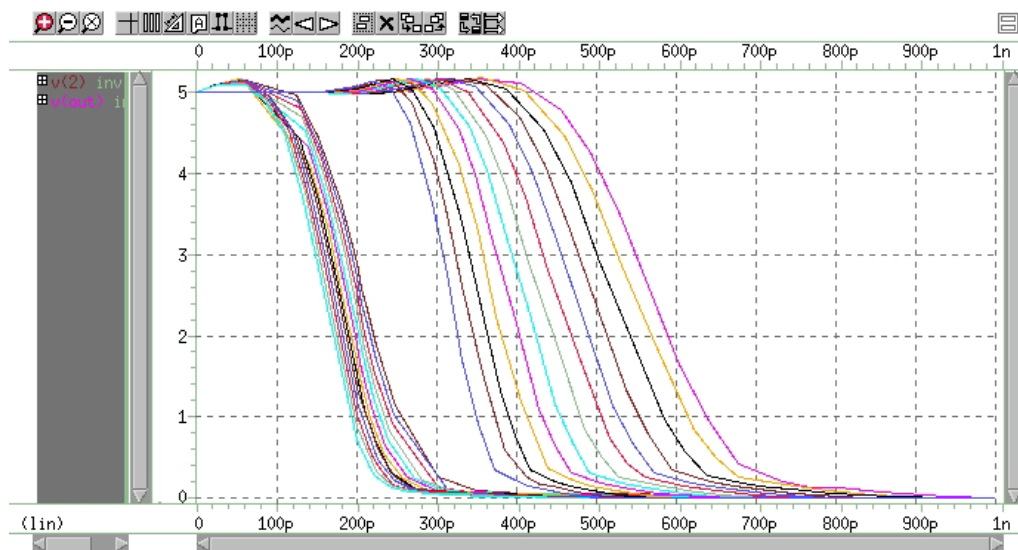


Figure 120 Sweep of Skew Parameters from -3 Sigma to +3 Sigma

To view the measured results, plot the `inv.mt0` output file. The plot in [Figure 121 on page 729](#) shows the measured pair delay and the total dissipative power, as a function of the parameter sigma. To get the specific operating point information of each Monte Carlo run, use `opfile=1`.

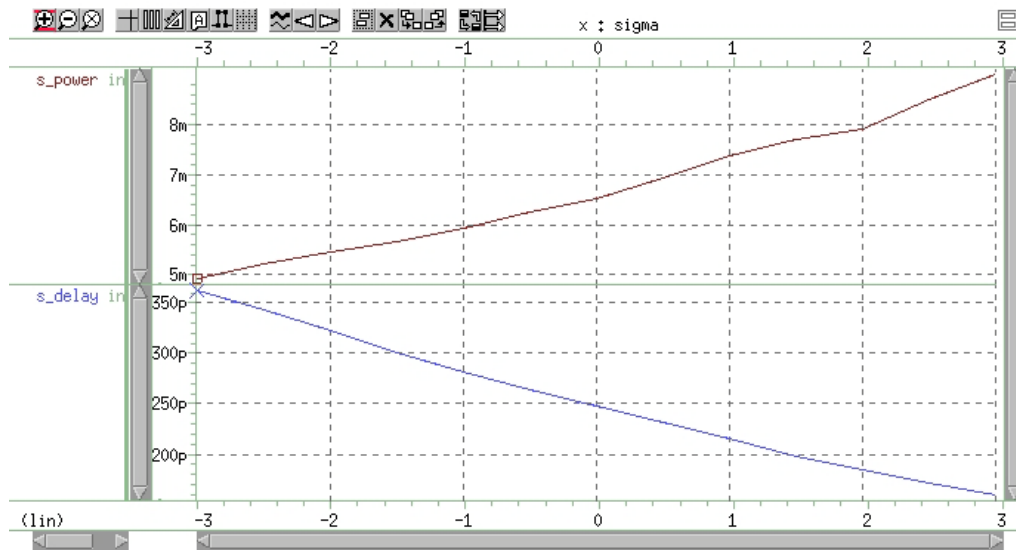


Figure 121 Sweep MOS Inverter, Pair Delay, and Power: -3 Sigma to 3 Sigma

Monte Carlo Results

This section describes the output of the Monte Carlo analysis in HSPICE. The plot in [Figure 122 on page 730](#) shows that the relationship between TOX against XL (polysilicon width=transistor length) is completely random, as set up in the input file.

To generate this plot, for example:

1. Read in the file `inv.mt1`.
2. Open the **Calculator**, select **TOX** (left mouse button), transfer to calculator (middle mouse button), and then select and transfer **XL**.
3. On the **WAVE** pulldown in the calculator, select $f(x)$, and then click the plot icon.
4. Using the right mouse button on the plotted waveform, select **Attributes** to change from the line plot to symbols.

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis

Worst-Case and Monte Carlo Sweep Example

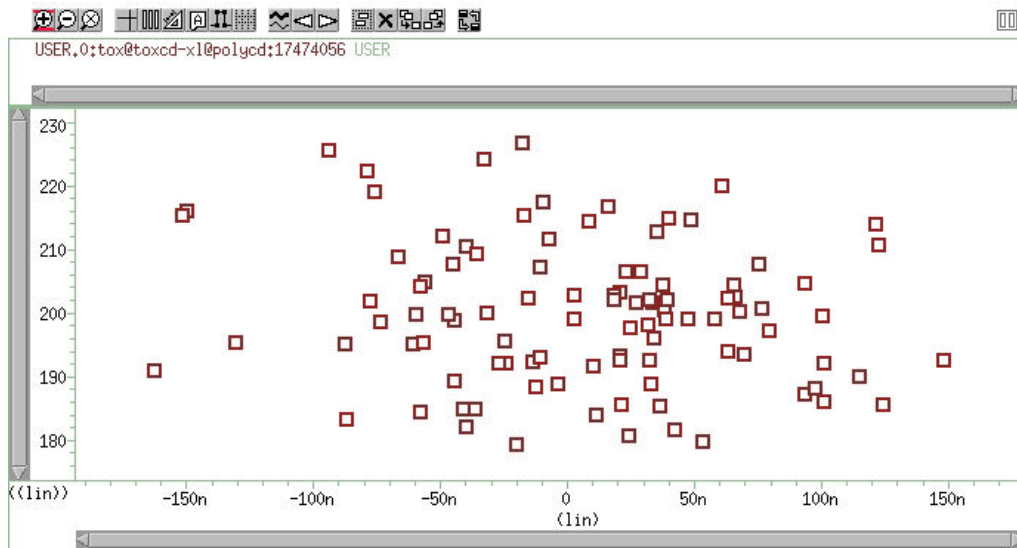


Figure 122 Scatter Plot of XL vs. TOX

Figure 123 is a standard scatter plot showing the measured delay for the inverter pair against the Monte Carlo index number.

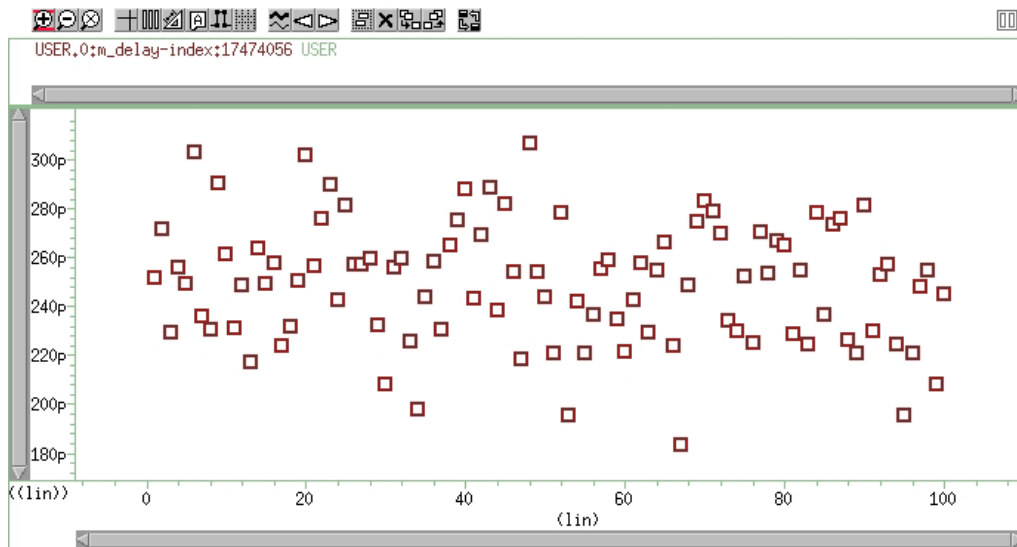


Figure 123 Scatter Plot of Inverter Pair Delay

If a particular result looks interesting; for example, if the simulation 68 (monte carlo index=68) produces the smallest delay, then you can obtain the Monte Carlo parameters for that simulation.

```
*** monte carlo index = 68 ***
MONTE CARLO PARAMETER DEFINITIONS
polycd xl = -1.6245E-07
nactcd xwn = 3.4997E-08
pactcd xwp = 3.6255E-08
toxcd tox = 191.0
vtoncd delvton = -2.2821E-02
delvtop = 4.1776E-02
vtopcd
rshncd rshn = 45.16
rshpcd rshp = 166.2
m_delay= 1.7929E-10 targ= 3.4539E-10 trig= 1.6610E-10
m_power= 6.6384E-03 from= 0.0000E+00 to= 1.0000E-09
```

In the preceding listing, the `m_delay` value of $1.79e-10$ seconds is the fastest pair delay. You can also examine the Monte Carlo parameters that produced this result.

The information on shortest delay and so forth is also available from the statistics section at the end of the output listing. While this information is useful to determine whether the circuit meets specification, it is often desirable to understand the relationship of the parameters to circuit performance. Plotting the results against the Monte Carlo index number does not help this purpose. You need to generate plots that display a Monte Carlo result as a function of a parameter. For example, [Figure 124 on page 732](#) shows the inverter pair delay to channel as a function of poly width, which relates directly to device length.

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis

Worst-Case and Monte Carlo Sweep Example

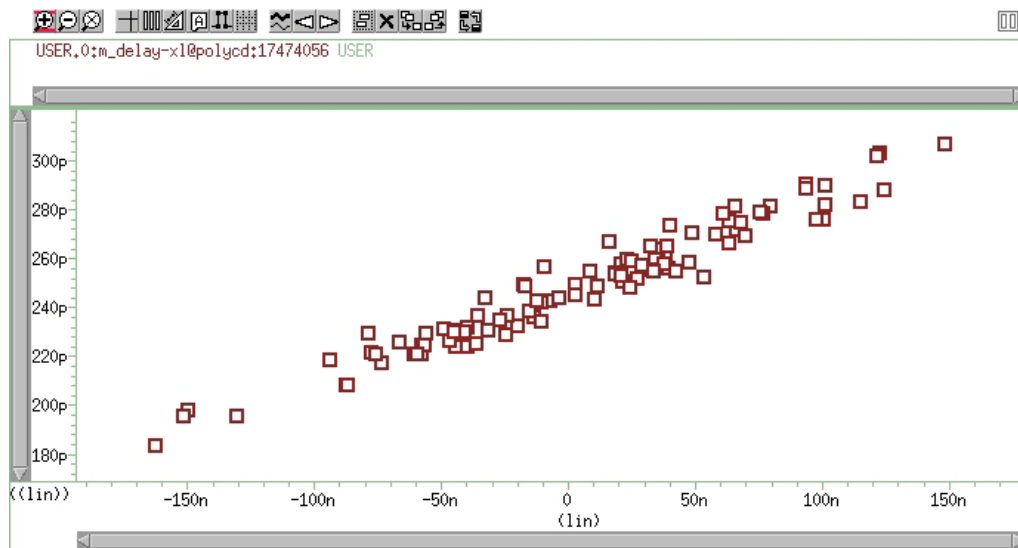


Figure 124 Delay as a function of Poly width (XL)

Figure 125 shows the pair delay against the TOX parameter. The scatter plot shows no obvious dependence, which means that the effect of TOX is much smaller than XL. To explore this in more detail, set the XL skew parameter to a constant and run a simulation.

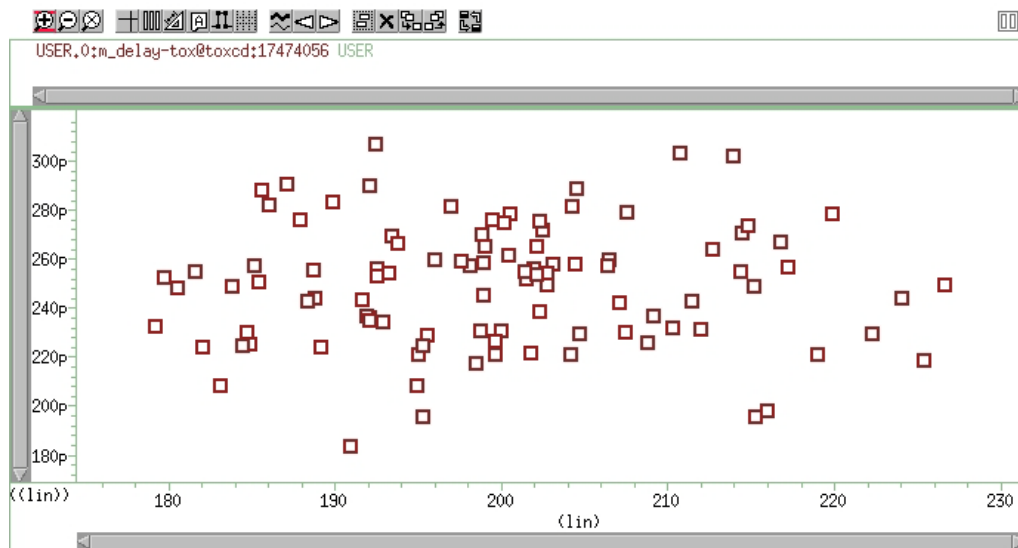


Figure 125 Sensitivity of Delay with TOX

The plot in [Figure 126](#) overlays the skew result with the ones from Monte Carlo. The skew simulation traverses the design space with all parameters changing in parallel and then produces a relationship between power and delay, which appears as a single line. Monte Carlo exercises a variety of independent parameter combinations, and shows that there is no simple relationship between the two results. Because the distributions were defined as Gaussian in the netlist, parameter values close to the nominal are more often exercised than the ones far away. With the relatively small number of samples, the chance of hitting a combination at the extremes is very small. In other words, designing for 3-sigma extreme for every parameter is probably not a good solution from an economic point of view.

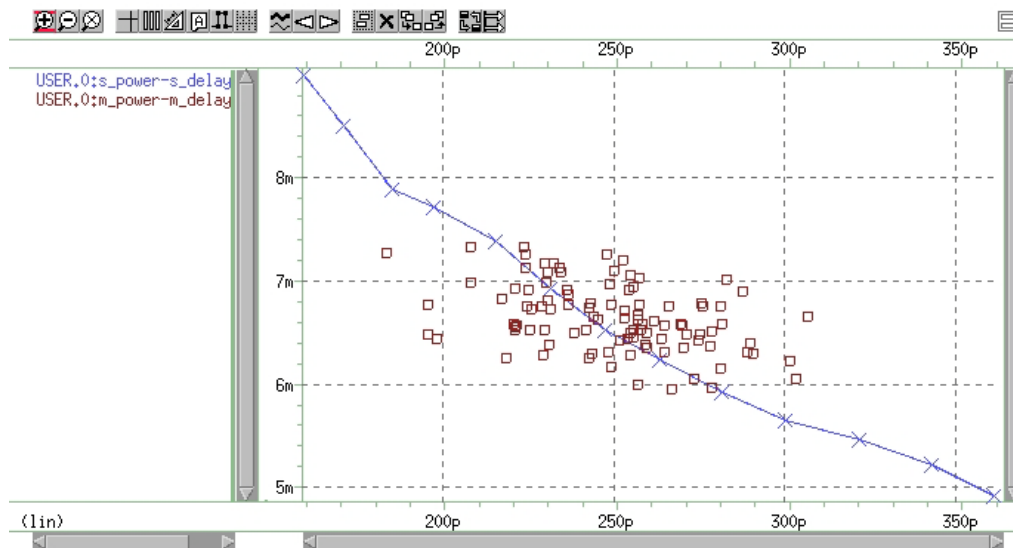


Figure 126 Superimposing Sigma Sweep Over Monte Carlo

[Figure 127 on page 734](#) superimposes the required part grades for product sales onto the Monte Carlo plot. This example uses a 250 ps delay and 6.5 mW power dissipation to determine the four binning grades.

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis

Global and Local Variations with Monte Carlo

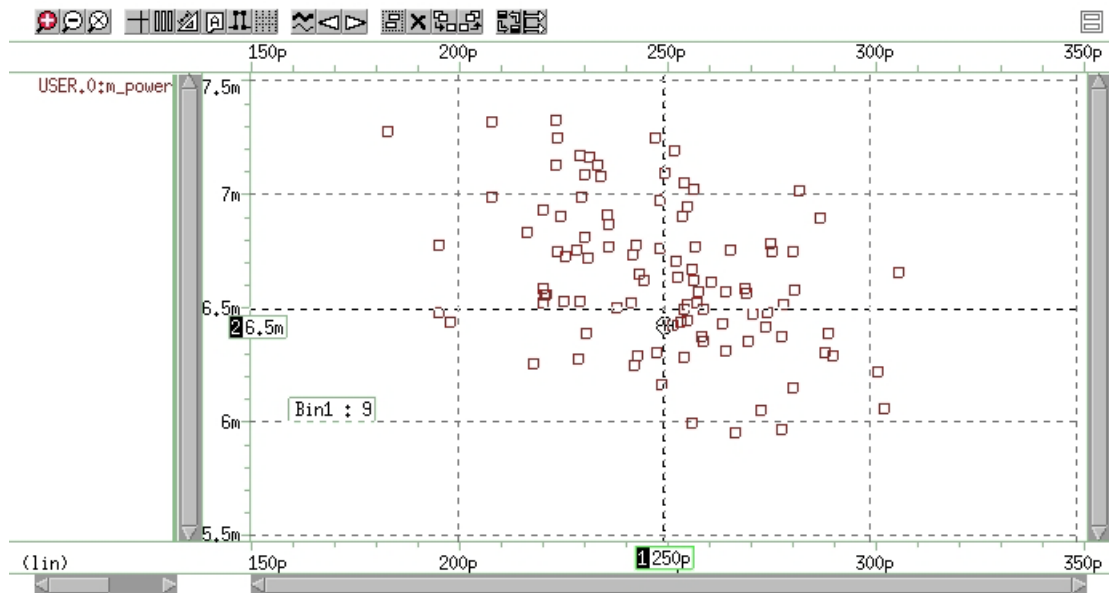


Figure 127 Speed/Power Yield Estimation

Sorting the results from `inv.mt1` yields:

- Bin1 - 9 percent
- Bin2 - 38 percent
- Bin3 - 29 percent
- Bin4 - 24 percent

If this circuit is representative of the entire chip, then the present yield should be 9 percent for the premium Bin 1 parts, assuming variations in process parameters as specified in the netlist. This example only shows the principle on how to analyze the Monte Carlo results..

Global and Local Variations with Monte Carlo

Monte Carlo analysis is dependent on a method to describe variability. Four different approaches are available in HSPICE:

1. Specify distributions on parameters and apply these to instance parameters.
2. Specify distributions on parameters and apply these to model parameters.

3. Specify distributions on model parameters using DEV/LOT construct.
4. Specify distributions on model parameters in a Variation Block.

In the following sections, the first three methods are described. The description relies on test cases, which can be found in the tar file `monte_test.tar` in directory `$installdir/demo/hspice/variability`.

The Variation Block is described in [Chapter 23, Variability Analysis Using the Variation Block](#), and Monte Carlo analysis controlled by the Variation Block is described in [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#).

Key to Demonstration Examples for Monte Carlo

The following sections discuss sample files delivered with HSPICE. See [Variability Examples](#). `monte_test.tar` is a suite of DC test files named `test1.sp` through `test20.sp` to test combinations of resistors, subckts, model/instance parameters, and so on.

The following sections discuss these demonstration files:

- [Variations Specified on Geometrical Instance Parameters](#)
- [Variations Specified in the Context of Subcircuits](#)
- [Variations on a Model Parameter Using a Local Model in Subcircuit](#)
- [Indirect Variations on a Model Parameter](#)
- [Variations Specified on Model Parameters](#)
- [Local Variations for Transistor Fingers](#)
- [Variations Specified Using DEV and LOT](#)
- [Combinations of Variation Specifications](#)

Variations Specified on Geometrical Instance Parameters

This method consists of defining parameters with variation using the distribution functions UNIF, AUNIF, GAUSS, AGAUSS, and LIMIT. These parameters are then used to generate dependent parameters or to replace instance parameters. In a Monte Carlo simulation, at the beginning of each sample, new random values are calculated for these parameters. For each reference, a new random value is generated; however, no new value is generated for a derived parameter. Therefore, it is possible to apply independent variations to parameters of different devices, as well as the same variation to parameters of a group of devices. Parameters that describe

distributions can be used in expressions, thus it is possible to create combinations of variations (correlations).

These concepts are best explained with circuit examples. In the following three examples, variation is defined on the width of a physical resistor, which has a model. If this device was a polysilicon resistor for example, then the variations describe essentially the effects of photoresist exposure and etching on the width of the poly layer.

- `test1.sp` has a distribution parameter defined called `globw`. A parameter called `globwidth` is assigned the value of `globw`. The parameter `globwidth` is assigned a different random value for each Monte Carlo sample. The parameter `globwidth` is used to define the width of the physical resistors `r1`, `r2`, `r3`, and `r4`, with model “resistor.” Because parameter `globwidth` does not have its own distribution defined, but rather gets its value from the parameter `globw`, the value for `globwidth` is the same wherever it is used; the resistors have the same width for each Monte Carlo sample, and therefore the same resistance. When plotting the simulation results `v1`, `v2`, `v3`, and `v4` from the `.meas` file, the waveforms overlay perfectly. This type of setup is typically used to model global variations (variations that affect all devices the same way).
- `test2.sp` has a distribution parameter defined called `locwidth`. This parameter is used to define the width of the physical resistors `r1`, `r2`, `r3`, and `r4`, with model “resistor.” Because the parameter has its own distribution defined, its value is different for each reference, and for each Monte Carlo sample. Therefore, the resistors always have different values, and the voltages are different. This type of setup is typically used to model local variations, which means variations that affect devices in a different way.
- `test3.sp` has two kinds of distributions defined: `globw/globwidth` as in the first example, and `locwidth` as in the second example. The sum of the two is used to define the width of the resistors. Therefore, the resistors will always have different widths: a common variation due to `globwidth` and a separate variation due to `locwidth`. In the example, the distribution for `locwidth` was chosen as narrower than for `globwidth`. When overlaying the measurement results, the large common variation can easily be seen; however, all voltages are different.

In summary, each reference to a parameter with a specified distribution causes a new random variable to be generated for each Monte Carlo sample. When referencing the parameter on an instance, the effect of a local variation is created. When referencing the parameter on an expression for a second

parameter and using the second parameter on an instance, the effect of a global variation is created.

Variations Specified in the Context of Subcircuits

The concept explained in the previous section applies to subcircuits as instances, and instances within subcircuits. Here we again use the example of a physical resistor, with variation of its width.

- In `test6.sp`, the resistor width is assigned inside the subcircuit. The variations get picked up from the top level. Because each subcircuit is a separate entity, the parameter “w” is treated as a separate reference, thus each resistor will have its own value, partly defined through the common value of “globwidth” and partly through the separate value of “locwidth.”
- `test7.sp` has two resistors in the subcircuit. Each device in each subcircuit has a separate reference to the variation, therefore each device gets its own value.
- In `test8.sp`, the variation definition for “locwidth” has been moved from the top level into the subcircuit. Each resistor has a common global variation and its own local variation.
- `test9.sp` assigns the top level variation to a local parameter, which in turn is applied to the width definition of the resistor. This happens independently within each subcircuit, thus we end up with the same values for the resistor pair in each subcircuit, but different values for the different pairs. This technique can be applied to long resistors when a middle terminal is required for connecting capacitance to the substrate. The resulting two resistor pieces will have the same resistance, but it will be different from other resistor pairs.

In summary, each subcircuit has its own parameter space, therefore it is possible to put groups of identical components into a subcircuit, and within each group all devices have the same parameter values, but between the groups, parameters are different. When specifying variations on these parameters, the effects of local variations between the groups are created.

Variations on a Model Parameter Using a Local Model in Subcircuit

If a model is specified within a subcircuit, then the specified parameter values apply only to the devices in the same subcircuit. Therefore, it is possible to calculate the value of a model parameter within the subcircuit; for example, as a function of geometry information.

When specifying variations on these parameters, the effects of local variations between subcircuits are created. If this method is used at the extreme with one device per subcircuit, then each device has its own model. This approach leads to a substantial overhead in the simulator and is therefore not recommended.

Indirect Variations on a Model Parameter

Variations on geometrical parameters are presented in [Variations Specified on Geometrical Instance Parameters](#) and [Variations Specified in the Context of Subcircuits](#). If we want to specify variations on a model parameter, for example, the threshold of a MOS device, the approach explained in the previous section with one model per device in a subcircuit could be used. However, this is impractical because the netlist needs to be created to call each device as a subcircuit, and because of the overhead. Since variations are of interest only on a few model parameters, an indirect method of varying model parameters can be used. Some special instance parameters are available for this purpose. For example, for MOS devices, the parameter `delvt0` defines a shift in threshold.

Referencing a parameter with a distribution as value for `delvt0` creates the effect of local threshold variations. A significant number of parameters of this type are available in HSPICE for BSIM3 and BSIM4 models. The variations can be tailored for each device depending, for example, on its size. A disadvantage of this method is that the netlist needs to be parameterized properly to get the correct variations. The process of preparing a basic netlist for Monte Carlo simulations with this approach is tedious and error-prone, therefore it is best handled with scripts.

For a listing of supported BSIM3 and BSIM4 instance parameters, see the *HSPICE Reference Manual: MOSFET Models*, [Supported Instance Parameters, BSIM3, BSIM4, BSIM3SOI and BSIM4SOI](#).

Variations Specified on Model Parameters

This section discusses the method of specifying distributions on parameters and using these parameters to define values of model parameters. With this approach, the netlist does not have to be parameterized. The `modmonte` option can be used to distinguish between global variations (all devices of a particular model have the same parameter set) or local variations (every device has a unique random value for the specified parameters).

- `test10.sp` shows a simple case where the model parameter for sheet resistivity is assigned a distribution defined on the parameter `rsheet`. The results show that all resistors have the same value for each Monte Carlo sample, but a different one for different samples. This setup is useful for studying global variations.
- `test11.sp` has `.option modmonte=1` added. Now every resistor has a different value.

Note that `.option modmonte` has no effect on any other approach presented here.

In summary, assigning parameters with specified distributions to model parameters allows for investigating the effects of global or local variations, but not both. The possibility of selecting one or the other with a simple option is misleading in the sense that the underlying definitions for global and local variations are not the same for a realistic semiconductor technology.

Local Variations for Transistor Fingers

You can run an MC simulation where each of the transistors' fingers of the gate are simulated to have a different process variation. Set option `MODMONTE=1` and get local variations for an instance and view MOSFET parameter variations with printing the alias `LX`, `LV`, and so on for different MOSFET parameters. For example:

```
*MOS L53 DC sweep test for monte carlo analysis
*
* MODMONTE option can be set to 1 or 0,
* MODMONTE=1: measured ids will be different for the 3 element
* MODMONTE=0: measured ids will be identical for the 3 element
*
.options ACCT OPTS LIST NOPAGE INGOLD=2 ALT999 PROBE POST=1 TNOM=25
.options MODMONTE=1 numdgt=10
.model nch nmos LEVEL=53 TOX=4E-9 wint=wint_nch VERSION=3.2
.param wint_nch = AGAUSS ( 3e-7 , 1e-7 , 3.0 )
m11 2 11 0 0 nch W=1E-6 L=0.15E-6
m12 2 11 0 0 nch W=1E-6 L=0.15E-6
m13 2 11 0 0 nch W=1E-6 L=0.15E-6
v01 2 0 1.5
v02 11 0 0.0
.dc v02 0 2.0 0.1 sweep monte=3
.meas dc ids_11 find par('i(m11)*1E3') when v(11)=1.5
.meas dc ids_12 find par('i(m12)*1E3') when v(11)=1.5
.meas dc ids_13 find par('i(m13)*1E3') when v(11)=1.5
.end
```

Variations Specified Using DEV and LOT

The two limitations of the approach described in [Variations Specified on Model Parameters](#) are resolved in this method by specifying global and local variations directly on a model parameter with the syntax:

```
parameterName=parameterValue LOT/distribution LotDist  
+ DEV/distribution DevDist
```

where:

- LOT is the keyword for global distribution
- DEV is the keyword for local distribution
- `distribution` is as explained in [Variations Specified on Geometrical Instance Parameters](#)
- `LotDist` and `DevDist` are the characteristic numbers for the distribution: 3-sigma is the fixed value for Gaussian distributions.

`test12.sp` has large global and small local variation, similar to the setup in the file `test3.sp`. The result shows four different curves, with a large common part and small separate parts. The amount of variation defined in the two files is the same. The curves look different from the `test3.sp` results because different random sequences are used. However, the statistical results (sigma) converge for a large number of samples.

There is no option available to select only local or only global variations. This can be an obstacle if the file is read-only or encrypted.

Combinations of Variation Specifications

Specifying distributions on parameters and applying them to model parameters can be used on some models and the DEV/LOT can be used approach on others in the same simulation.

- `test13.sp` has DEV/LOT specified for model `res1`, and the parameter “width” for model `res2`. The values for the resistors with model `res1` are different, and the values for resistors with model `res2` are the same.
- `test14.sp` is similar to `test7.sp` and has `modmonte=1` specified. All four resistors have different values. However, note that in reality, the sigma for width would be different when simulating local or global variations.
- `test15.sp` has instance parameter variations specified on two resistors and DEV/LOT on two others. From the waveforms, `v3` and `v4` form a first pair, and `v1` and `v2` form a second pair.

It is also possible to mix variations on instance parameters and model parameters in the same setup.

- `test16.sp` has small instance parameter variations specified on width and relatively large model parameter variations on the sheet resistivity, `rsh`. The results show four different waveforms, with a common behavior.
- `test17.sp` shows instance and model parameter variations as in the previous test case, but `.option modmonte` is set to 1. Thus, the model variations affect every device in a different way. The results show completely independent behavior of all four resistors.

If an instance parameter or instance parameter variations and model parameter variations are specified on the same parameter, then the instance parameter always overrides the model parameter. Because only few parameters can be used in both domains, this case is rather seldom, but it needs to be considered to avoid unexpected results.

- `test18.sp` has model variation specified on width with a parameter. Two resistors have width also defined on instance. The resistors with instance parameter do not vary at all. The other two resistors vary independently, as expected because `.option modmonte` is set to 1.
- `test19.sp` is similar to `test18.sp` with `.option modmonte` set to 0. The two resistors that do not have width defined on the instance line vary together.
- `test20.sp` has `DEV/LOT` specified. Instance parameters override variations on selected resistors.

Variation on Model Parameters as a Function of Device Geometry

For local variations (see [Chapter 25, Mismatch Analyses](#)), it is a common requirement to specify variation on a model parameter as a function of device geometry. For example, the MOS device threshold was observed to vary with the total device area.

The approach explained in [Indirect Variations on a Model Parameter](#) can be used. While this allows for specifying local variations on each device, it does not include the capability of using expressions based on element parameters. Thus, variation cannot be described with an expression that includes the device's geometry. Conceptually, a netlist processor could be written that

inserts the appropriate values for the parameters as a function of device size. (Synopsys does not provide such a tool).

The DEV/LOT approach has no mechanism to describe variation as a function of an element parameter.

Troubleshooting Monte Carlo Issues

Perturbation Information Missing from Output Listing in Monte Carlo and Subcircuit Local Variables

A limitation in traditional Monte Carlo is that no perturbation information is printed in the output listing file in the case where a subcircuit has local variation parameters defined.

This limitation in the traditional Monte Carlo can be seen in cases similar to the following:

```
.param
+ my_global_lt_x=AGAUSS(0,1.1n,1)
+ temp_global_lt_x=my_global_lt_x
+ subc_global_lt_n=temp_global_lt_x
+ subc_global_lt_p=temp_global_lt_x

+ local_lt_n=AGAUSS(0,0.88n,1)
+ local_lt_p=AGAUSS(0,0.88n,1)
+ local_wt_n=AGAUSS(0,5.3n,1)
+ local_wt_p=AGAUSS(0,5.3n,1)
x1 in 2 inv
.subckt inv in out subc_global_lt_n=0
subc_global_lt_p=0 local_lt_n=0
+ local_lt_p=0 local_wt_n=0 local_wt_p=0
mn out in 0 0 nch W='3e-07+subc_global_lt_n+local_wt_n'
+
L='4e-08+subc_global_lt_n+local_lt_n'
mp out in 1 1 pch W='3e-07+subc_global_lt_p+local_wt_p'
+
L='4e-08+subc_global_lt_p+local_lt_p'
.ends
-----
*.lis file output
my_global_lt_x

temp_global_lt_x= -7.1777E-10
local_lt_n

1:mn = 3.9274E-08 2:mn = 3.9461E-08
3:mn = 3.8590E-08 4:mn = 3.9071E-08
```

In the above case, HSPICE prints out variation in global variation parameter `temp_global_lt_x` directly. However, the local variation parameter `local_lt_n` is actually the expression value,
`L='4e-08+subc_global_lt_n+local_lt_n'`.
 Therefore, you have to find the `local_lt_n` value from $[3.9574E-08 - (4e-8 + \text{subc_global_lt_n})]$.

This can be overcome by a small modification in the subckt definition by reassigning the local parameters inside the subcircuit as shown next:

```
.subckt inv in out subc_global_lt_n=0 subc_global_lt_p=0
local_lt_n=0
+ local_lt_p=0 local_wt_n=0 local_wt_p=0
*Assign the local parameters inside the subckt again
.param subc_local_lt_n=local_lt_n
subc_local_lt_p=local_lt_p
+ subc_local_wt_n=local_wt_n
+ subc_local_wt_p=local_wt_p
mn out in 0 0 nch W='3e-07+subc_global_lt_n+subc_local_wt_n'
+
L='4e-08+subc_global_lt_n+subc_local_lt_n'
mp out in 1 1 pch W='3e-07+subc_global_lt_p+subc_local_wt_p'
+
L='4e-08+subc_global_lt_p+subc_local_lt_p'
.ends
```

This directly gives the `subc_local_lt_n` value as follows:

```
my_global_lt_x
temp_global_lt_x= 1.7577E-10
local_lt_n
1:subc_local_lt_n= 9.2081E-10 2:subc_local_lt_n= 1.4300E-10
3:subc_local_lt_n= -2.6043E-10 4:subc_local_lt_n= -2.6994E-09
```

Chapter 22: Monte Carlo—Traditional Flow Statistical Analysis
Global and Local Variations with Monte Carlo

Variability Analysis Using the Variation Block

Introduces variability, describes how it can be defined in HSPICE, and introduces the Variation Block.

HSPICE ships numerous examples for your use. See [Variability Examples](#) for path to demo files.

These topics are covered in the following sections:

- [Overview of Variation on Silicon](#)
- [Variability in HSPICE](#)
- [Overview of the Variation Block](#)
- [Variation Block Structure](#)
- [Variation Block Examples](#)
- [Group Operator {...} and Subexpressions](#)
- [Interconnect Variation in StarRC with the HSPICE Flow](#)
- [Control Options and Syntax](#)

Overview of Variation on Silicon

As semiconductor technologies migrate to ever-smaller geometries, larger relative variations in device characteristics are being observed. These fluctuations in device characteristics have been analyzed and classified for the purpose of dealing with the variations in manufacturing during the design phase. The following types of variations can be identified at the wafer level:

Chapter 23: Variability Analysis Using the Variation Block

Overview of Variation on Silicon

- Global variations from foundry, lot, or due to wafer processing.
- Across-wafer variations due to materials, gas flow, thermal, optical, and spin processes.
- Linear variations across the area of a chip.
- Local variations are observed between closely spaced identically designed devices as a result of microscopic random processes. Microscopic variations include line edge roughness, finite number of dopant atoms in the channel, and atomic level oxide thickness changes.

In analog design, certain circuit characteristics can be made insensitive to global variations and across-chip variations by applying the concept of matched devices; however, these characteristics are still affected by the local variations. In digital designs for nanometer technologies, large local variations can cause unacceptable variations in path delays and signal slopes.

Large circuits suffer from spatial or position-dependent variations, which create problems with clock skew for devices that are far apart. Finally, device characteristics are affected by features in proximity (metal coverage, fill patterns, mechanical strain, shape variation due to lithography, and so on) and orientation. Most of these variations are systematic and can be reduced through layout restrictions or accounted for in post-layout verification.

Historically, only the effects of variation on device characteristics (transistors, resistors, and capacitors) have been considered. In nanometer technologies, variations in the interconnect should also be taken into account because the relative variation in the resistance and capacitance has increased due to smaller wire width and interconductor spacing.

These variations combined, summarized as *parametric variability*, dominate yield loss in nanometer technologies. The circuits function in terms of connectivity, but do not meet specifications on metrics such as speed, leakage, or offset. For example, while the threshold of MOS devices gets smaller, approaching 200 mV, the variation in threshold gets larger, with standard deviation up to 30 mV for short devices. Due to the low supply voltages, in combination with requirements for high speed, the circuits stop working with these large spreads in device characteristics.

Therefore, simulating (or predicting) the effects of these variations on circuit response is increasingly important, in particular when considering the high mask costs and time-to-market constraints for the majority of today's products.

To simulate the effects of the variations in device characteristics due to materials and manufacturing, they need to be described in a way that the simulator can handle in an efficient manner. Traditionally, global variations were

specified through process corner files, and the other types of variations mentioned above were either guessed or ignored. In recent years, statistics blocks were added to the model files. They describe variations in terms of distributions on device model parameters. An even newer approach for defining variations is the Variation Block, described later in this chapter.

The following analyses are available in HSPICE to simulate the effects of variations on circuit response:

- Monte Carlo analysis is the traditional method for finding the variation in circuit response resulting from parameter variations.
- DC and AC mismatch analyses are efficient methods for simulating the effects of variations on a circuit's DC or AC response.

To get satisfactory answers from these analyses, the variation definitions must have been generated for the target technology of the design, similar to device models.

Variability in HSPICE

Three approaches are available for defining variability in HSPICE:

- Defining a Variation Block; for example:

```
.Variation
  global and local variation definitions
.End_Variation
```

- Defining variations on parameters; for example:

```
.param var=AGAUSS(20,1.2,3)
```

For a discussion of this topic, see [Chapter 22, Monte Carlo—Traditional Flow Statistical Analysis](#).

- Defining variations on models using `lot` and `dev` parameters in the model file; for example:

```
vth0=0.6 lot/0.1 dev/0.02
```

For a discussion of this topic, see [Chapter 22, Monte Carlo—Traditional Flow Statistical Analysis](#).

Chapter 23: Variability Analysis Using the Variation Block

Variability in HSPICE

The Variation Block approach replaces the older methods of defining variations on parameters and models in HSPICE because it best fulfills the requirements for simulating nanometer technology devices.

The advantages of the Variation Block over previous solutions are:

- The Variation Block consolidates variation definitions in single records.
- A clear distinction exists between Global, Local, and Spatial Variations.
- A subset of variation types can be selected in a dependent simulation.
- The syntax allows for defining Local and Global Variation as a function of device geometry, and Spatial Variation as a function of device location.
- Monte Carlo results derived from the Variation Block are consistent with those from DCMatch or ACMatch analyses.
- Additional files, suitable for data mining, are generated.

```

.Variation
  .Global_Variation
    NMOS SNPS20N Vth0=0.07  U0=10 %
    PMOS SNPS20P Vth0=0.08  U0= 8 %
  .End_Global_Variation
  .Local_Variation
    NMOS SNPS20N
+   Vth0='2.5e-8/sqrt(get_E(W)*get_E(L)*get_E(M))'
+   U0  ='6.1e-5/sqrt(get_E(W)*get_E(L)*get_E(M))' %
    PMOS SNPS20P
+   Vth0='2.5e-8/sqrt(get_E(W)*get_E(L)*get_E(M))'
+   U0  ='6.1e-5/sqrt(get_E(W)*get_E(L)*get_E(M))' %
  .Element_Variation
    R r=10 %
  .End_Element_Variation
  .End_Local_Variation
  .End_Variation

```

$\sigma_s(\Delta V_{th0}) = 70 \text{ mV}$
 $\sigma_s\left(\frac{\Delta U_0}{U_0}\right) = 0.1$

$\sigma_s(\Delta V_{th0}) = \frac{2.5 * 10^{-4}}{\sqrt{Area}} \text{ V}$
 $\sigma_s\left(\frac{\Delta U_0}{U_0}\right) = \frac{6.1 * 10^{-2}}{\sqrt{Area}}$

Figure 128 Variation Block Example

In this example, the following *global* variations are defined:

- On NMOS devices with model snps20n
 - Absolute variation on threshold vth0, Normal distribution with sigma=70 mV

- Relative variation on mobility μ_0 , Normal distribution with $\sigma=10\%$
- On PMOS devices with model snps20p: the global variations are defined similarly

The following *local* variations are defined:

- On NMOS devices with model snps20n and PMOS devices with model snps20p, respectively
 - Absolute variation on threshold v_{th0} , Normal distribution with $\sigma=2.5e-8/\sqrt{\text{total_device_area}}$
 - Relative variation on mobility μ_0 , Normal distribution with $\sigma=6.1e-5/\sqrt{\text{total_device_area}}$
- On resistors which do not have a model: relative variation of 10 percent on the implicit value parameter

The following sections present these topics:

- [Overview of the Variation Block](#)
- [Variation Block Structure](#)
- [Variation Block Examples](#)
- [Interconnect Variation in StarRC with the HSPICE Flow](#)
- [Control Options and Syntax](#)

Overview of the Variation Block

The characteristics of circuits produced in semiconductor processing are subject to variability, as is the case for any manufactured product. For a given target technology, the nominal device characteristics are described with a set of parameters, which applies to a certain device model (for example, BSIM4). In HSPICE, the variability of the model parameters is described through a *Variation Block*. A Variation Block is a container for specifying variations introduced by the effects in manufacturing on geometry and model parameters.

Variations in HSPICE are modeled as Global, Local, and Spatial variations.

- Global Variations are variations in device characteristics from lot to lot, wafer to wafer, and chip to chip; they are caused by variations in starting material and differences between equipment and manufacturing procedures. Global Variations affect all devices with the same model name in the same way.

Chapter 23: Variability Analysis Using the Variation Block

Overview of the Variation Block

- Local Variations are defined as variations between devices in proximity, or with common centroid layout on the same chip; they are caused by microscopic variations in materials and geometry, and affect different devices differently.
- Spatial Variations are defined as variations due to the physical arrangement of the devices on the layout; they are caused by gradients from material properties, imperfections of lenses, and spin processes. The dependence on distance means that large designs are more affected by Spatial Variations.

All three classes can be described in the Variation Block in a flexible way by user-defined expressions. Because there are currently no industry-wide standards for specifying process variability, this feature allows each company to implement their own proprietary model for variability. The Variation Block is generally provided by a modeling group, very similar to device models (for example, BSIM) because it must be created specifically for each technology from test circuits.

Like a model, the Variation Block can be part of a library which is encrypted; therefore, the content is not accessible to the designers. They can introduce additional Variation Blocks in their netlist to define options and variations on generic elements. See [Control Options and Syntax](#) and [Variations of Element Parameters](#).

The structure of the Variation Block allows for building expressions to model interdependence and hierarchy of the variations. For example, one random variable can control the variation in oxide thickness of both PMOS and NMOS devices, as it is generally the same for both types of devices.

Note that the earlier methods for specifying variation are not compatible with the Variation Block. For controlling the behavior of Variation Blocks, see [Control Options and Syntax on page 781](#). The Variation Block is currently used for Monte Carlo, and DC/AC mismatch analyses; for a description of these analyses, see [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#) and [Chapter 25, Mismatch Analyses](#), respectively.

For the functions available to build expressions as presented in the next sections, see [Using Algebraic Expressions on page 379](#).

Variation Block Structure

A Variation Block is divided into four sections:

- General section
- Subblock for Global Variations
- Subblock for Local Variations
- Subblock for Spatial Variations

This section presents the syntax of a Variation Block, followed by a discussion of the contents of the four sections.

```
.Variation
  Define options
  Define common parameters that apply to all subblocks
  .Global_Variation
    Define the univariate independent random variables
    Define additional random variables through transformations
    Define variations of model parameters
  .Element_Variation
    Define variations of element parameters
  .End_Element_Variation
  .End_Global_Variation
  .Local_Variation
    Define the univariate independent random variables
    Define additional random variables through transformations
    Define variations of model parameters
  .Element_Variation
    Define variations of element parameters
  .End_Element_Variation
  .End_Local_Variation
  .Spatial_Variation
    Define the univariate independent random variables
    Define additional random variables through transformations
    Define variation of model parameters
  .End_Spatial_Variation
  .End_Variation
```

General Section

In the general section, options can be defined that control the variability analyses that use the content of the Variation Block. Options can be specified, one per logical record.

Note: “.OPTION” (with a leading period) does not work for options specified in the Variation Block.

The correct Variation Block syntax is:

```
Option OptionName = value
```

See [Control Options and Syntax](#) at the end of this chapter for a listing and description of Variation Block control options.

Parameters, also, can be defined that apply to all subblocks in which variations are specified; however, these cannot contain any distribution-related functions. Parameters defined within a Variation Block have local name scope and are completely independent of parameters defined outside it.

For example: `parameter PI=3.1416`

Subblocks for Global, Local, and Spatial Variations

Within the variation subblocks, univariate independent random variables can be defined. These are random variables with specific distributions over a certain sample space. Additional random variables can be generated through transformations. These random variables form the basis for correlations and complicated distributions.

A basic rule of the Variation Block approach is to place the model definition on the top level, instead of inside a subcircuit, as necessary in the old approach. In all three subblocks, variations on model parameters can be defined. This is where Global or Local Variations on the parameters of semiconductor devices are specified.

Note: The `.MALIAS` command is supported for diode, BJT, JFET, and MOSFET models in `.Global_Variation` and `.Local_Variation` blocks.

A special section within the subblock for Local Variations allows for defining Local Variations on elements. Use this section either for specifying local temperature variations or variations on generic elements that do not have a model, as used early in the pre-layout design phase, or for off-chip components; for example, resistors and capacitors. Local and Global variation support the block operator brackets described in [Group Operator {...} and Subexpressions on page 772](#).

The following sections discuss these topics:

- [Independent Random Variables](#)
- [Dependent Random Variables](#)
- [Absolute Versus Relative Variation](#)
- [Variations on Model Parameters](#)
- [Variations on Subcircuit Parameters](#)
- [Variations on Top-Level Parameters](#)
- [Variations on Temperature](#)
- [Access Functions, Get_E\(\), Get_P\(\), Get_M\(\), Get_O\(\)](#)
- [Spatial Variation](#)

Independent Random Variables

When describing variations, a standard normal (Gaussian) distribution is assumed, unless otherwise specified explicitly. This default behavior is explained in later sections. Other types of distributions or correlations are modeled by applying transformations to the independent random variables. These independent random variables are derived from three basic distributions:

- Uniform distribution: defined over the range from -0.5 to 0.5: $U()$
- Normal distribution: with mean=0 and variance=1, default range +/-4: $N()$
- Use the `LIMIT` distribution to create discrete “corners” for a variable. Specify it as:

```
Parameter xx=LIMIT(nominal_val, abs_variation)
```

These distributions cannot be defined within expressions; variables must be assigned and then the variables can be used within expressions. See examples of this operation in [Non-Gaussian Probability Distribution Functions on page 716](#).

The `LIMIT` keyword provides a random-limit distribution function by using absolute variation. The syntax adds +/- `abs_variation` to `nominal_val` based on whether the random outcome of an underlying uniform distribution is over the range of -1 to 2.

Chapter 23: Variability Analysis Using the Variation Block

Variation Block Structure

For example, in the following Variation Block `aa=Limit(-2,2)` means the `nominal_value` is -2, and the `abs_variation` is 2.

```
.Variation
.Local_Variation
  Parameter aa = Limit(-2,2)
  R Rmodel res= Perturb('aa')
.End_Local_Variation
.End_Variation
```

The sampling values in an *.mc file are:

```
index r1:@:rmodel:@:aa:@:ILL  r2:@:rmodel:@:@aa:@:ILL
r3:@:rmodel:@:aa:@:ILL
1      0.          0.          0.
2     -4.0000     -4.0000     -4.0000
3      0.          0.          0.
4      0.          0.          -4.0000
5      0.          -4.0000     -4.0000
6      0.          -4.0000     -4.0000
7     -4.0000     0.          0.
```

In a Monte Carlo sampling, \pm `abs_variation` is added to `nominal_val`. So the sampling values are either -4 or 0; these values appear in a random order.

- User-defined cumulative distribution function: CDF (xyPairs)

If $f(x)$ is the probability density of a random variable x , then the cumulative distribution function is the integral of $f(x)$. A cumulative distribution function can be approximated by a piecewise linear function, which can be described as a sequence of pairs of points $[x_i, y_i]$. The following rules apply:

- At least two pairs are required
- White space or a comma is required between each number
- The CDF starts at zero: $y_1=0$
- The CDF ends at one: $y_n=1$
- x_i values must be monotonically increasing $x_{i+1} > x_i$
- y_i values must be monotonically non-decreasing $y_{i+1} \geq y_i$

- $$\int_{-\infty}^{\infty} x \cdot f(x) \cdot dx = 0$$

where, the probability density function, $f(x)$, is the derivative of the cumulative density.

Example

The probability density function is shown in Figure 129 (1). Figure 129 (2) gives the corresponding cumulative distribution function. This is coded in the Variation Block as:

```
Parameter var=CDF(-0.1 0 -0.05 0.5 0.05 0.5 0.1 1.0)
```

The histogram generated by taking 1000 samples is shown in Figure 129 (3).

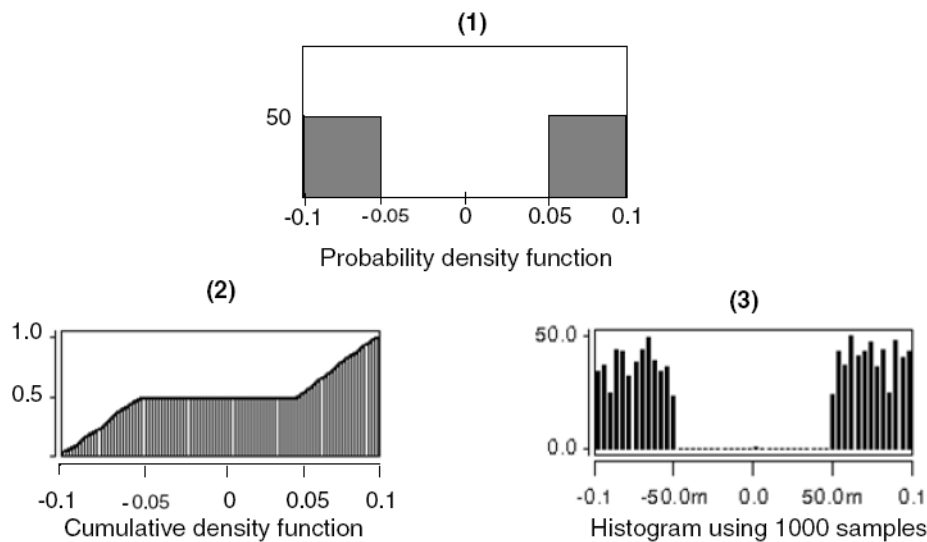


Figure 129 Probability, cumulative density and 1000-sample histogram

The distributions N() and U() do not accept any arguments.

The syntax for defining independent random variables is:

```
Parameter a=U() b=N() c=CDF(x1, y1, . . . , xn, yn)
```

These distributions cannot be defined within expressions; variables must be assigned and then the variables can be used within expressions. See examples of this operation in [Non-Gaussian Probability Distribution Functions on page 716](#).

The LIMIT keyword provides a random-limit distribution function by using absolute variation. The syntax adds +/- abs_variation to nominal_val

based on whether the random outcome of a -1 to 1 distribution is greater than or less than 0.

Dependent Random Variables

To model distributions which are more complicated than the ones which are available through the predefined independent random variables, transformations can be applied by using expressions on independent random variables. A dependent variable can also be created as a function of more than one independent random variable to express correlation.

Example 1

This example creates a random variable with normal distribution, with mean A and standard deviation B.

```
Parameter var=N() Y='A + B * var'
```

Example 2

This example creates a random variable with a uniform distribution from D to E, where D and E are arbitrary constants.

```
Parameter var=U() Y='0.5*(D+E) + (E-D) * var'
```

Example 3

A variable x has a log-normal distribution if log(x) is normally distributed. The probability density function for the log-normal distribution is:

$$f(x) = \frac{1}{\sigma x \sqrt{2\pi}} \cdot \exp\left(-\frac{(\ln(x) - \mu)^2}{2\sigma^2}\right)$$

where μ is the mean and σ the standard deviation of associated normal distribution. Samples from such a distribution can be generated as

```
Parameter var=N()  
Parameter nor='mu+sigma*var'  
Parameter lognor='exp(nor)'
```

Example 4

If the components of the random vector $\underline{x} = (x_1, x_2, \dots, x_n)$ are all independently distributed as standard normal and the vector

$\underline{y} = (y_1, x_2, \dots, y_n)$ is defined as $\underline{y} = \underline{a} + \underline{Bx}$

then \underline{y} is distributed as multivariate normal with mean \underline{a} and covariance matrix $\underline{BB'}$ where the prime indicates transpose.

Now consider the inverse problem of generating samples from a multivariate normal distribution with given mean vector $\underline{\mu}$ and covariance matrix $\underline{\Sigma}$.

The covariance matrix has the following properties:

- The matrix is symmetric.
- The diagonal elements are non-negative.
- The matrix is positive semi-definite where all eigenvalues are real and non-negative.

Now consider the Cholesky decomposition \underline{L} of $\underline{\Sigma} = \underline{LL'}$. Then samples from the distribution of \underline{y} are generated as $\underline{y} = \underline{\mu} + \underline{L} \cdot \underline{x}$ with x being samples from the standard normal distribution.

Consider a numerical example with ($\underline{\mu} = \mathbf{0}$) and

$$\underline{\Sigma} = \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 2 & 0.3 \\ 0.5 & 0.3 & 1.5 \end{bmatrix}$$

$$\underline{L} = \begin{bmatrix} 1 & 0 & 0 \\ .5 & 1.323 & 0 \\ .5 & 0.0378 & 1.1174 \end{bmatrix}$$

An observed covariance matrix from a million samples gives

$$\begin{bmatrix} 1.0003 & 0.4990 & 0.5005 \\ 0.4990 & 1.9977 & 0.2994 \\ 0.5005 & 0.2994 & 1.4990 \end{bmatrix}$$

The correlation matrix is closely related to Σ through diagonal scaling and has these properties:

Chapter 23: Variability Analysis Using the Variation Block

Variation Block Structure

- The matrix is symmetric.
- All diagonal elements are unity.
- All off-diagonal elements are bounded by unity in magnitude.
- The matrix is positive semi-definite.

Note that the first three properties are not sufficient when describing multivariate normal distributions. For example, in the matrix:

$$\begin{bmatrix} 1 & 0.7 & 0.0 \\ 0.7 & 1 & 0.8 \\ 0.0 & 0.8 & 1.0 \end{bmatrix}$$

All pair-wise correlations seem correct, but the eigenvalues of the matrix are (2.06, 1, -0.06) and the matrix is indefinite.

Absolute Versus Relative Variation

By default, the specified variation is absolute, which means additive to the original model or element parameter; however, sometimes it is more appropriate to specify relative variations that are defined by appending a space and a “%” sign to the expression. The simulator divides the result of the expression by 100, and multiplies the result by the original parameter value and the random number from the appropriate generator to calculate the change.

Example

In the following example, the variation on the threshold parameter `vth0` is specified as normal with absolute sigma of 80 or 70 mV, and the variation on the mobility `u0` is specified as relative 15 or 13 percent.

```
.Global_Variation
  Nmos snps20N vth0=0.08 u0=15 %
  Pmos snps20P vth0=0.07 u0=13 %
.End_Global_Variation
```

Variations on Model Parameters

Variations on model parameters can be defined in subblocks for Global, Local, and Spatial Variation. In the course of the simulation, these variations are then applied to the specified device model parameters. Model parameter variations are described with the following syntax:

```
Model_Type Model_Name Model_Parameter=Expression for Sigma
```

The syntax `Model_Parameter=Expression` for `Sigma` is a shorthand notation for `Variation_in_Model_Parameter='Expression for Sigma'`.

If the expression references only constants and parameters that evaluate to constants, then a Gaussian variation with zero mean and a sigma equal to the expression is automatically implied. To describe variation as a function of previously defined independent random variables, use the construct `'Perturb()'`, with the following syntax:

```
Model_Type Model_Name Model_Parameter=Perturb('Expression')
```

The expression for sigma should be enclosed in quotes, see the general HSPICE rules for [Using Algebraic Expressions on page 379](#).

The following lines define a global Variation, with implicit normal distribution, with zero mean and sigma of 10, on the parameter `rsh` of resistors with model `Rpoly`.

```
.Global_Variation
  R Rpoly rsh=10
.End_Global_Variation
```

In the next example, the independent variable `Toxvar` is used to model global Variations on oxide thickness. `Toxvar` is an independent random variable with a normal distribution, with mean=0 and sigma=1. In the device models `nch` and `pch`, `Toxvar` is applied to the parameters `tox` with a different multiplier. The oxide thicknesses in the two models vary in parallel; they are correlated.

```
.Global_Variation
  Parameter Toxvar=N()
  Nmos nch tox=Perturb('7e-10*Toxvar')
  Pmos pch tox=Perturb('8e-10*Toxvar')
.End_Global_Variation
```

HSPICE supports the following model types: NMOS, PMOS, R, Q, D, and C.

Variations can only be defined on parameters that are explicitly specified in the associated device model.

For binned models, variations can be defined separately by specifying the model name with the bin extension; for example, devices from bins 1 and 2 receive different variation on the parameter `lint`, which models length variation:

```
Nmos snps20N.1 lint=10n
Nmos snps20N.2 lint=12n
```

Variations on Subcircuit Parameters

The Variation Block allows for defining variation on parameters, which are specified in the subcircuit definition record with a default value. Default values can be overwritten by specifying them at subcircuit instantiation.

The syntax is:

```
Subckt SubcktName Parameter='expression for sigma'
```

The following rules apply for these types of definitions:

- Only parameters that are defined as formal numeric arguments on the subcircuit definition record can be subject to variation. (This is the line which starts with `.SUBCKT` and possibly has continuation lines.)
- The subcircuit must not be defined within another subcircuit.
- If the subcircuit contains a model, then variations on the model parameters, as described in section, [Variations on Model Parameters](#), are not supported. Instead, variations need to be defined on a subcircuit parameter and the parameter used inside the model.

The subckt parameters variation feature addresses the following three needs:

- A component is defined with an expression, not available in a model:

```
r1 1 0 'Rsh*l/w*(1+b1*(tanh(b2*abs(v(1,0)/l))))'
```

This expression models a voltage-dependent resistor, with non-linear dependence not available in a traditional model. If this resistor is called within a subcircuit, and parameters are specified on the subcircuit definition record, then variation can be modeled, for example, on `Rsh`, `l`, and `w`.

```
i1 0 1 1m
x1 1 0 rtanh
.subckt rtanh a b rsh=1k w=1u l=1u
.param b1 = -0.4 b2 = 8u
r1 a b 'Rsh*l/w*(1+b1*(tanh(b2*abs(v(a,b)/l))))'
.ends
.Variation
.Global_Variation
    subckt rtanh rsh=10 %
.End_Global_Variation
.Local_Variation
    subckt rtanh rsh=3 %
.End_Local_Variation
.End_Variation
```

- A component is represented by a network and the subcomponents must have the same value when local variations are specified:

```
r1 end1 middle rmodel w='w+dw' l='l/2+dl'
r2 end2 middle rmodel w='w+dw' l='l/2+dl'
c1 middle sub cmodel w=w l=l
```

In this example, the resistor has a center tap, where a capacitor is connected. If these three components are defined within a subcircuit and parameters *dw*, *dl*, and *rsh* are defined on the subcircuit definition record, then the two resistors of the same network always have the same value; local variations only cause different instantiations of the subcircuit to have different equivalent resistance between the network terminals.

- You need to calculate the value of a device model parameter through an equation because the built-in equations are not adequate. For example:

```
.subckt nch n1 n2 n3 n4 dvth0_glob=0 dvth0_loc=0 du0_glob=0
+ du0_loc=0
+ dtox=0 dlint=0 dwint=0 l=60n w=120n as='w*90n' ad='w*90n' ...
.param vth0_base=0.345 u0_base=0.015
.param vth0_geo=function1(w,l,temper,vth0_base)
.param u0_geo=function2(w,l,temper,u0_base)
.param dvth0_geo=function3(w,l,dvth0_loc)
.param du0_geo=function4(w,l,temper,du0_loc)
M1 n1 n2 n3 n4 nch25 w=w l=l as=as ad=ad ...
.MODEL nch25 NMOS LEVEL = 54
+ vth0='vth0_geo+dvth0_glob+dvth0_geo'
u0='u0_geo*(1+du0_glob)*(1+du0_geo)'
+ tox='2.6n+dtox' lint='2.1n+dlint' wint='5.3n+dwint' ...
.ends
X1 d1 g1 s1 b1 nch l=60n w=150n
X2 d2 g2 s2 b2 nch l=80n w=120n
.Variation
.Global_Variation
    subckt nch dvth0_glob=0.03 dtox=0.12n du0_glob=0.2 dlint=2n
    + dwint=3n
.End_Global_Variation
.Local_Variation
    subckt nch dvth0_loc=2.0m dtox=0.03n du0_loc=0.03 dlint=21p
    + dwint=47p
.End_Local_Variation
.End_Variation
```

The values of model parameters *vth0* and *u0* are defined through user-defined equations (*function1* and *function2*), with dependency on device size and temperature. This necessitates a local model (*nch25*). The parameters with variations are declared on the subcircuit definition line. In this

example, global and local variations are processed differently through the subcircuit, therefore the respective variations have to be specified on separate parameters. Global variations (`dvth0_glob` and `du0_glob`) are applied to the model parameters `vth0` and `u0` directly. Local variation definitions (`dvth0_loc` and `du0_loc`) are adjusted for device size using `function3` and `function4`, and result then in the variations `dvth0_geo` and `du0_geo` applied to the model parameters `vth0` and `u0`.

While this use model is supported, it is not desirable because it leads to one model per device, which is inefficient in terms of memory and performance.

Variations on Top-Level Parameters

Variations on top-level parameters can be defined for Global Variation. The keyword `Top` is available to specify top-level parameter variation.

```
Top top_level_parameter='expression'  
Top top_level_parameter=Perturb('expression')
```

The following example defines the +/-10% global variation on parameter `VDD` with uniform distribution.

```
.param vdd=2.5  
.Variation  
    .Global_Variation  
        Parameter uniV=U()  
        Top vdd=perturb('20*uniV') %  
    .End_Global_Variation  
.End_Variation
```

Note: The top-level parameter variation can only be specified in Global variation.

Variations on Temperature

You can define Variation on temperature for global variations to support the whole space of Process-Voltage-Temperature. The temperature variations affect all devices in the netlist.

The keyword `Temp` (or `Temper`) specifies temperature variation.

```
Temp temp='expression'  
Temp temp=Perturb('expression')
```


The following example defines the global variation on temperature.

```
.Variation
.Global_Variation
    Temp temp='10'
.End_Global_Variation
.End_Variation
```

Note: You can only specify temperature variation in global variation.

Access Functions, Get_E(), Get_P(), Get_M(), Get_O()

Important: An error results if any of the arguments to these functions is invalid or undefined. This strong action is due to the need for the technology library to be bug free.

Get_E()

Certain variations depend on element geometry, as defined with parameters at instantiation. The `Get_E()` access function (only supported by the Variation Block) allows accessing these parameters in expressions by using the following syntax:

```
Get_E(Element_Parameter)
```

where *Element_Parameter* is the name of an element parameter, which you must define on the instantiation line (except for the `DTEMP` parameter and the multiplier `M` which have implicit values). Use this access function for specifying variations as a function of device geometry. The `Get_E()` access function reports the effective device geometry, after resolving parameters, scales and adjustments by process parameters, such as, `xw`, `xl`, `wint`, `lint`. Refer to the [HSPICE Reference Manual: MOSFET Models](#) for equations which depend on the model `LEVEL` and [Geometric Scaling for Diode Models](#) in the *HSPICE Reference Manual: Device and Element Models* for scaling equations.

For example, you often specify that the local variation on the threshold is as inversely proportional to the square root of the total area of the device, as calculated from the product of the element parameters `W`, `L`, and `M`.

```
Nmos nch vth0='1.234e-9/sqrt(Get_E(W)*Get_E(L)*Get_E(M))'
```

In addition, Variation Block can handle cases of calling `NF` and `M` for elemental variation with the `Get_E()` function. For example:

```
.Variation
.Global_Variation
    parameter var = N()
```

Chapter 23: Variability Analysis Using the Variation Block

Variation Block Structure

```
.Element_Variation
  M(model_name ~= 'NCH') W = Perturb('var*100n*Get_E(NF)')
.End_Element_Variation
.End_Global_Variation
.End_Variation
```

Get_E() Function for Obtaining Channel L/W for MOSFETs

Beginning with G-2012.06-SP1, you can also use the `get_e()` function to obtain channel length and width with MOSFET models as follows:

- `get_e(L) | get_e(W)` — Returns channel length/width for all MOSFET models.
- `get_e(LDRAWN) | get_e(WDRAWN)` — Returns user-defined L/W for MOSFET models.
- `get_e(Leff) | get_e(Weff)` — Returns effective channel length/width for MOSFET models.

The following is an example of the `get_e()` function with different input:

```
.option scale = 0.9
mn1 net031 inn net044 nmosbulk snps20N L=1u W=7u m=4 nf=2
get_e(wdrawn) = 7u
get_e(w) = (wdrawn/nf)*scale = 3.15u
get_e(weff) = 3.14u
```

Get_P()

Another function allows for accessing the values of global parameters by using the following syntax:

```
Get_P(Global_Parameter)
```

The circuit context provides the parameter value, for example, from the subcircuit, if defined inside, otherwise from the top level. In the following example, sweep parameter “tol” determines the resistor variation:

```
.param tol=1
ral 1 0 1k
il 0 1 1m
.Variation
  .Local_Variation
    .Element_Variation
      R R='Get_P(tol)' %
    .End_Element_Variation
  .End_Local_Variation
.End_Variation
.dc tol 1 5 1 monte=100
```

Get_M()

The Variation Block format also allows the access of model parameters. HSPICE provides the function `Get_M()` for this purpose.

```
Get_M(Model_Parameter_Name)
```

The `Get_M()` function normally has a single argument.

```
.Variation
  .Global_Variation
    nmos snps20N u0 = '2.345e-6*GET_M(u0) '
  .End_Global_Variation
.End_Variation
```

Get_O() to Return the Value of HSPICE Options

If a variation must be expressed as a function of a simulator option (specified as `.option=optionval` outside the Variation Block), the access function `Get_O()` is available, using the construct `Get_O(option_name)`. For example, if you use the element parameter `nf` (number of fingers) with some advanced models, the device width reported by the `Get_E` function depends on the value of `.OPTION WNFLAG`. For variation as a function of total device area, the following definition produces the expected results, independent of the settings of `WNFLAG`:

```
vth0 = `6.0621e-9/sqrt(Get_E(W)*Get_E(L)*Get_E(M)*\`
      (1-Get_O(WNFLAG)+Get_O(WNFLAG)*Get_E(NF))`
```

Variations of Element Parameters

Variations affect devices not only in the underlying model parameters, but also through variations of properties specified at instantiation of an element, or variations on implied properties, such as local temperature. Also, early in the design phase, passive devices sometimes have only a nominal value, but no model as yet because designers have made no decision on a specific implementation. For these elements, you can specify variations on the implicit value parameter; for example: `R1 1 0 1k`.

You define variations on element parameters for Local Variations in a section within the Local Variation subblock.

You describe element parameter variations with the following syntax:

```
Element_Type Element_Parameter = 'Expression for Sigma'
```

The syntax `Element_Parameter = 'Expression for Sigma'` is shorthand notation for:

```
Variation_in_Element_Parameter = Expression for Sigma
```

Chapter 23: Variability Analysis Using the Variation Block

Variation Block Structure

If the expression references only constants and parameters that evaluate to constants, then it automatically implies that a Gaussian variation with zero mean and a sigma are equal to the expression. To describe variation as a function of previously defined independent random variables, use the construct `Perturb()`, with the following syntax:

```
Element_Type Element_Parameter = Perturb('Expression')
```

Enclose the expressions in quotes (see the general HSPICE rules for [Using Algebraic Expressions on page 379](#)). See also, [Parameters and Expressions on page 78](#) for limitations.

The following lines define a normal distribution with sigma of 10 on the resistors without model:

```
.Element_Variation
  R R=10
.End_Element_Variation
```

In the following example, the specified temperature variation affects only resistor `ra2` with a uniform distribution from 0 to 10 degrees (the resistor resides next to a power device).

```
ra1 1 0 1k
rb1 2 0 1k
ra2 3 0 rpoly l=10u w=1u
rb2 4 0 rpoly l=10u w=1u
.model rpoly r rsh=100 tc1=0.01
.Variation
  .Local_Variation
    .Element_Variation
      Parameter tempvar=U()
      R(Element_Name~='ra*' && Model_Name~='rpoly')
      + dtemp=Perturb('10*tempvar+5')
    .End_Element_Variation
  .End_Local_Variation
.End_Variation
```

Because different classes of devices might be affected differently, use of a condition clause provides a selection mechanism based on element name and model name:

```
Element_Type(condition_clause) Element_Parameter= 'Expression
for Sigma'
```

The condition clause allows for specifying variations on selected elements, according to their name or associated model. You can indicate wildcard substitutions as “?” for single character and “*” for multiple characters.

Examples for condition clause syntax are:

```
Element_Type(model_name~='modelNameA')
Element_Type(element_name~='elNameB')
Element_Type(model_name~='modelNameC' OPERATOR
  element_name~='elNameD') par='expression'
Element_Type(model_name~='modelNameE' OPERATOR
  model_name~='modelNameF') par='expression'
Element_Type(element_name~='elNameG' OPERATOR
  element_name~='elNameH') par='expression'
```

where OPERATOR can be && (AND), || (OR). The operator “~=” stands for “matches”.

All pattern matching operations are case-insensitive. HSPICE ignores a leading subcircuit prefix when matching the element name.

Example

In this example, only resistor `ra1` varies.

```
ra1 1 0 1k
rb1 2 0 1k
.Variation
  .Local_Variation
  .Element_Variation
    R(element_name~='ra*') R=20
  .End_Element_Variation
  .End_Local_Variation
.End_Variation
```

Supported element types and their parameters are:

Table 75 Supported elements and parameters

Element	Parameters
M	DTEMP, L, W, AD, AS, PD, PS, NRD, NRS, RDC, RSC, VDS, VGS, VBS, DELVTO
R	Rval* DTEMP, L, W, TC1, TC2, C, AC, SCALE
C	Cval* DTEMP, L, W
Q	AREA DTEMP, AREA, AREAB, AREAC, TEMP
J	L, W, DTEMP
D	DTEMP, L, W
L	Lval* DTEMP, TC1, TC2
I	DCval* mag phase

Table 75 Supported elements and parameters

Element	Parameters
V	DCval* mag phase

The asterisk “*” denotes implicit value parameter. The DTEMP parameter is implicit; it needs specification on the element instantiation line.

Example for Voltage Source

Netlist element: V1 1 0 0.1 AC

Variation definition:

- V DC=5% (5% of 0.1)
- V MAG=5% (5% of 1)
- V PHASE=5 (5 degrees)

You define variations on element parameters for Global Variations in a section within the Global Variation subblock with almost the same syntax as within the Local Variation. But there is a limitation to when you can use a condition clause: If there is a condition clause in the variation definition line of a Global Element Variation, then use the Variation Block keyword `Parameter` to designate at least one independent random variable. The `Perturb` function uses an independent random variable element parameter perturbation. Other forms of perturbation are illegal. For example:

```
Parameter var1=N() var2=U()
Element_Type (condition_clause) Element_Parameter =
Perturb('Expression')
```

Example

In this example, `ra1` and `rb1` varies with global variation. So in each Monte Carlo trial, `ra1` and `rb1` are of the same resistor value.

```
ra1 1 0 1k
rb1 2 0 1k
.Variation
.Global_Variation
.Element_Variation
Parameter a=N()
R(element_name~='r*') R=Perturb('20*a')
.End_Element_Variation
.End_Global_Variation
.End_Variation
```

Note: If there are only two resistors of `ra1` and `rb1` in netlist, then an equivalent definition of Global Element Variation is:

```
.Variation
  .Global_Variation
    .Element_Variation
      R R=20
    .End_Element_Variation
  .End_Global_Variation
.End_Variation
```

But the next definition is *illegal* because the condition clause matches all resistors in the netlist:

```
.Variation
  .Global_Variation
    .Element_Variation
      R(element_name~='r*') R=20
    .End_Element_Variation
  .End_Global_Variation
.End_Variation
```

Spatial Variation

To make the Spatial Variation useful, HSPICE needs the coordinate of a particular device. The element instantiation must extend to include placement information. For example, for a MOS device:

```
Mid Dn Gn Sn Bn ModelName w=width l=length x=xcoor y=ycoor
```

In the Spatial Variation definition, you access the element coordinates by using the `Get_E()` function.

For the current release, HSPICE supports only netlists with a single subcircuit, with devices on the top level or in the subcircuit. All devices of the model that has Spatial Variation defined, must have coordinates. These coordinates are numbers specified by you (no parameters allowed).

Special Rules Regarding Variation Block Usage

Generally, a foundry creates the contents of the Variation Block. To safeguard against unintentional overwriting of these variation definitions:

- The name-space of the Variation Block is separate from the netlist contents.
- Once you specify a variation on a parameter, you cannot redefine it later, even in `.ALTER` statements. For example, if you want to change the corners defined in a model library file with a `.ALTER` statement, then you must specify the Variation Block in a separate `*.lib` section.

Variation Block Examples

You can review the following simple Variation Block in the example netlists `opampdcm.sp` and `opampmc.sp`. These netlists are available in the HSPICE demo directory: `$installdir/demo/hspice/variability`

The example defines the following variations:

- Global Variations on `vth0` (absolute)
- Global Variations on `u0` (relative)
- Local Variations on `vth0` (absolute), as a function of device area
- Local Variations on `u0` (relative), as a function of device area
- Local Variation on the implicit value of resistors (relative)

```
.Variation
  .Global_Variation
    Nmos snps20N vth0=0.07 u0=10 %
    Pmos snps20P vth0=0.08 u0=8 %
  .End_Global_Variation
  .Local_Variation
    Nmos snps20N vth0='1.234e-9/
      sqrt(Get_E(W)*Get_E(L)*Get_E(M))'
      + u0='2.345e-6/sqrt(Get_E(W)*Get_E(L)*Get_E(M))' %
    Pmos snps20P vth0='1.234e-9/
      sqrt(Get_E(W)*Get_E(L)*Get_E(M))'
      + u0='2.345e-6/sqrt(Get_E(W)*Get_E(L)*Get_E(M))' %
  .Element_Variation
    R r=10 %
  .End_Element_Variation
  .End_Local_Variation
. End_Variation
```

Principal Component-based Global Variation Modeling

In this example, the independent random variables `A1`, `A2`, and `A3` are the principal components on which all variations (`nmos` and `pmos`) are modeled. See [1] for details.


```
.Global_Variation
    Parameter A1=N() A2=N() A3=N()
    Nmos nch
+   tox =Perturb('-6.2E-12*A1-8.1E-12*A2-2.7E-12*A3')
+   vth0=Perturb('-3.6E-03*A1+8.9E-03*A2-1.5E-03*A3')
+   cjn =Perturb('-3.2E-06*A1+6.7E-06*A2-4.3E-06*A3')
+   u0  =Perturb(' 5.6E-04*A1-9.7E-04*A2+7.6E-04*A3')
+   ....
    Pmos pch
+   tox =Perturb('-7.5E-12*A1-6.9E-12*A2-8.8E-12*A3')
+   vth0=Perturb('-7.4E-03*A1+3.3E-03*A2-7.2E-03*A3')
+   cjn =Perturb('-5.0E-06*A1+8.9E-06*A2-3.2E-06*A3')
+   u0  =Perturb(' 7.6E-04*A1-4.3E-04*A2+4.8E-04*A3')
+   ....
.End_Global_Variation
```

Local Variation Example for Submicron Technology

This Local Variation data was created using the methodology outlined in [2]. Note the different dependencies on w and l for the different parameters.

```
.Local_Variation
    Nmos nch
+   tox ='3.1e-07/sqrt(Get_E(L)*Get_E(W)*Get_E(M))' %
+   wint ='6.2e-12/sqrt(Get_E(L)*Get_E(M))'
+   lint ='2.0e-12/sqrt(Get_E(W)*Get_E(M))'
+   nch ='1.9e-06/sqrt(Get_E(L)*Get_E(W)*Get_E(M))' %
.End_Local_Variation
```

Spatial Variation Example

```
.Variation
    .Spatial_Variation
        Parameter a = N( )
        Parameter b = U( )
        Parameter Pi = 3.14159265
        Parameter Angle = 'Pi*2*b'
        NMOS snps20n
+   vth0 = Perturb('20*a*sqrt(Get_E(x)* Get_E(x)+ Get_E(y)*
Get_E(y)) \\
        *cos(Angle-atan(Get_E(y)/Get_E(x))-Get_E(x)<0?Pi:0)')
    .End_Spatial_Variation
.End_Variation
```

The Spatial Variation is specified as a plane with a slope sigma of 20 mV/mm, and arbitrary rotation on the chip surface.

Group Operator {...} and Subexpressions

To improve readability of complex variation specifications, a group operator {...} and subexpressions are available. Used within a defined group, subexpressions can reference element and global parameters.

Syntax

```
ModelType ModelName {Parameter ...ModelParameter= ...}
```

The group operator {...} separates variation definitions group by group. Each group uses one model, which means all parameters defined inside a group operator are specific to this model. A group definition starts after the Model Name, and must end after the last model parameter specification of the same Model Name. Parameter definitions support expressions with `Get_E()`, `Get_P()`, and `Get_M()`.

ModelParameter definitions have no leading Parameter key.

Syntax Extension with Bins

```
ModelType ModelName {  
  Parameter ...  
  ModelParameter= ...  
  ModelName.1 ModelParameter= ...  
  .....  
  ModelName.m ModelParameter= ...  
}
```

Model parameter definitions within a group before the first bin name (ModelName.1 in the example) apply to all bins; whereas the following definition is bin specific: ModelName.1 ModelParameter= ...

Example

In this example, note that the expressions before NMOS apply to all bins, whereas those for mn.12 are bin specific.

```
.Global_Variation  
  Parameter PG1=N() PG2=N() PG3=N()  
+ dxl=' 4.3e-9 * PG1 '  
+ dvth0='0.02 * (-0.29 * PG1 + 0.95 * PG2) '  
+ dtaxe='1.3e-10 * (0.39 * PG1 -0.87 * PG2 + 0.28 * PG3) '  
+ F1='1.0/(2*SQRT(dvth0*dvth0)) '
```

```

+ F_FF='(-dvth0+SQRT(dvth0*dvth0))*F1'
+ F_SS='(dvth0+SQRT(dvth0*dvth0))*F1'
NMOS mn.12 {
    Parameter u0varg='-dvth0*(F_FF*2.1+F_SS*0.6)'
    xl=Perturb(dxl)
    vth0=Perturb('dvth0')
    lvth0=Perturb('dvth0*(F_FF*0.097+F_SS*0.054)')
    u0=Perturb('u0varg') %
    wu0=Perturb('u0varg')%
    lu0=Perturb('u0varg')%
    pu0=Perturb('u0varg')%
    toxex=Perturb('dtoxe') toxpy=Perturb('dtoxe')
}
.End_Global_Variation
.Local_Variation
NMOS mn.12 {
    Parameter sqrtarea='SQRT(Get_E(W)*Get_E(L)*Get_E(M))'
    vth0='1.2e-9/sqrtarea'
    u0='2.3e-6/sqrtarea'
}
.End_Local_Variation

```

Rules for Using the Group Operator

The following rules apply when using the group operator:

- You cannot define independent random variables inside a group.
- Group operators do not support condition clauses inside a group.
- Any specifications that appear at the same line and after the opening '{' are ignored; a parameter definition should begin at a new line after the bracket.
- Group operators only support model parameter, *not* subcircuit parameter definitions.
- You can define the same `ModelType` and `ModelName` only once in a group; HSPICE aborts the simulation if it finds duplicate group definitions.

Parameter Scope

Parameters defined inside and outside a group have the following scope: parameters defined inside a group cannot conflict with those defined outside it. However, the same parameter can be redefined inside another group, and these are invisible to each other.

Chapter 23: Variability Analysis Using the Variation Block
 Group Operator {...} and Subexpressions

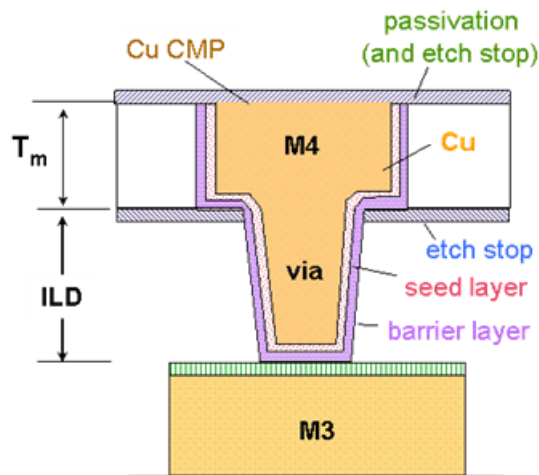
The parameter scopes are as follows:

<pre>.Param a=2</pre>	<p>Defined outside a Variation Block; can be referenced using <code>get_P()</code> syntax</p>
<pre>.Variation Parameter a=3</pre>	<p>Global parameter within Variation Block; can be used in all subblocks</p>
<pre> .Global_Variation Parameter b='a*Get_P(a) ' NMOS nch { Parameter c=0.4*b } PMOS pch { Parameter c='-0.3*b' }</pre>	<p>$b = 6$; valid in global variation subblock</p> <p>$c = 2.4$; only visible in this group;</p> <p>Can be redefined in other groups</p> <p>$c = -1.8$; c is only visible in this group;</p>
<pre>.End_Global_Variation .Local_Variation Parameter b='2*a*Get_P(a) '</pre>	<p>Can be redefined in other groups</p> <p>$b=12$; valid in local variation subblock</p>
<pre> .Element_Variation R r='0.1*b' % .End_Element_Variation .End_Local_Variation</pre>	<p>Relative sigma of r is $0.1*2*2*3*0.01=0.012$</p>
<pre>.End_Variation</pre>	

Interconnect Variation in StarRC with the HSPICE Flow

With its Z-2006.12 release, the Synopsys layout extraction tool, StarRC, established a Sensitivity-Based Extraction Flow, which can generate a variation-aware netlist to interpret and produce simulation results based on the probability distribution of interconnect variations. The currently available methodology of running worst-case corners produces pessimistic results, as opposed to the new method, which calculates the actual distribution, and which then allows for selecting design limits based on yield.

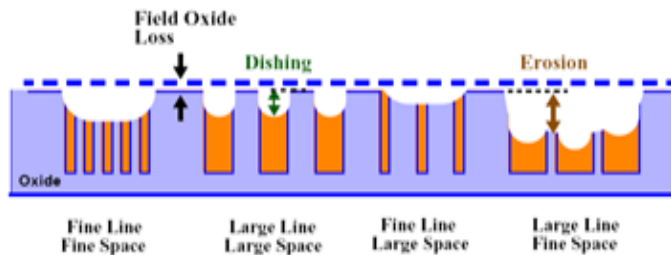
The Sensitivity-Based Extraction Flow, StarRC, extracts resistors and capacitors associated with the interconnect. HSPICE then works as a post-processor to do statistical analysis with the output file from StarRC. This file contains sensitivity information that HSPICE requires to support Variation Block-based ACMatch, DCMatch, and Monte Carlo analyses. [Figure 130](#) shows a typical cross-section of the wires on levels 3 and 4. The metal and interlayer dielectric thicknesses, the conductor widths and the material properties of the conductors and dielectrics can be subject to variation.



Generic dual damascene process

Figure 130 Interconnect structure

In addition to random variation, the wires also have height variation due to CMP, as shown in [Figure 131](#). This variation is mostly systematic and depends on wire widths and local metal density. StarRC accounts for the corresponding change in resistance and capacitance.



Source: Ph.D Thesis, T. Park, MIT 2002

Figure 131 Systematic Variation Due to CMP

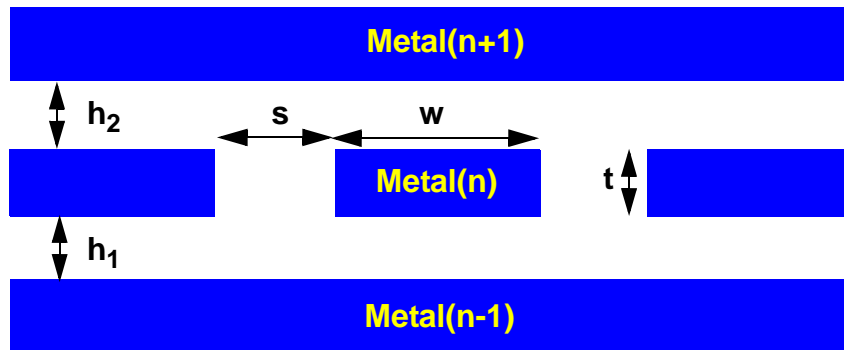
Refer to the *StarRC User Guide*, Chapter 11: Variation-Aware Extraction for more information.

Variation Block and Statistical Sensitivity Coefficients

Consider the idealized interconnect representation shown in [Figure 132 on page 777](#). The horizontal and vertical dimensions as well as the material properties are subject to random variations.

A Pade' style approximation that relates electrical values to these variations gives results that closely match simulations from field solvers. We call the coefficients in the Pade' approximation *statistical sensitivities*.

StarRC generates and provides statistical sensitivity coefficients that correspond to each parasitic value. These coefficients measure the expected change in the capacitance/resistance due to the variation of an interconnect process parameter. By definition, the fractional change of capacitance/resistance value due to a unit variation in a specific parameter is the statistical sensitivity of the capacitance/resistance in question with respect to that parameter.



Interconnect Process Variables:

- t — Metal thickness
- h_1, h_2 — Dielectric thickness
- w, s — Line width and spacing
- ϵ — Dielectric constant
- ρ — Resistivity

Typical 16-layer process has over 150

Figure 132 Random Variation in Interconnect

The combination of the nominal capacitance/resistance tables, and the corresponding statistical sensitivity coefficients, provides the necessary and sufficient coverage for all possible effects of parameter variations on capacitance/resistance. This eliminates the need for using extensive sets of capacitance tables, and provides a realistic coverage of all possible ranges of random variation.

Application of statistical sensitivity coefficients requires that the parameter variations be small. This restriction is acceptable for nanometer semiconductor processes since a large part of the process variation tends to be systematic and is considered and modeled under the scope of deterministic process variation.

Given the distribution of parameter variations, based on statistical sensitivity information, you can get the statistical effects on capacitance and resistance values in Monte Carlo, DCMatch, and ACMatch analyses. The `Interconnect_Variation` subblock defines interconnect variability. Currently, the variation is restricted to the global level. StarRC creates the `Interconnect_Variation` subblock and includes it as part of the post-layout netlist.

Usage Example and Input Syntax

The following sections illustrate the parts of an interconnect Variation Block:

- [1: Interconnect Variation Block](#)
- [2: Model Card in the Header Section](#)
- [3: Parasitic Section](#)

1: Interconnect Variation Block

The information in the Variation Parameters section is re-coded as follows:

```
.Variation
  .Interconnect_Variation
    .Global_Variation
      ID= param_id Name = param_name [R_Sensitivity_Type =
      + param_type] [C_Sensitivity_Type = param_type]
      [L_Sensitivity_Type = param_type] [K_Sensitivity_Type =
      + param_type] [CV= coeff_of_var]
      ...
    .End_Global_Variation
  .End_Interconnect_Variation
.End_Variation
```

Argument	Description
param_id	Is a non-negative integer to uniquely identify the parameter. In this way, every parameter is associated with a different integer. These unique identifiers are used in the parasitic section to represent the sensitivity information.
param_name	Are alphanumeric characters without any spaces or meta characters.
param_type	Valid values are N, D, or X. These refer to the form of the sensitivity expression and indicate if the particular parameter variation appears in the numerator, the denominator, or does not influence the element value. If not specified, the default is X.
coeff_of_var	This argument is numeric and optional. The default value is 1.

Variation blocks have global scope and the above definition should appear outside any subcircuit definitions. `R_Sensitivity_Type`, `L_Sensitivity_Type`, and `K_Sensitivity_Type` help to define the form of the sensitivity expression. This is a generalization of the Taylor series-based

variation form, $1 + \sum_{i \in I} s_i \Delta p_i$, to the more general Pade'

$$\text{approximation: } \frac{1 + \sum_{i \in I} s_i \Delta p_i}{1 + \sum_{j \in J} s_j \Delta p_j}.$$

The index sets I and J are disjoint, for example, a parameter can influence either the numerator or the denominator, but not both.

In the current StarRC-VX and HSPICE releases, only resistors have the more general Pade' form, capacitors have the Taylor series form, and inductors (normal and K-Matrix style) have no variation.

Example of the extended NETNAME-style information

```
.Variation
  .Interconnect_Variation
    .Global_Variation
      ID=0 Name=ME1_T R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.06
      ID=1 Name=ME1_W R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.04
      ID=2 Name=ME1_R R_Sensitvity_Type=N C_Sensitvity_Type=X
CV=0.05
      ID=3 Name=ME12_T R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.06
      ID=4 Name=ME12_ER R_Sensitvity_Type=X C_Sensitvity_Type=N
CV=0.02
      ID=5 Name=ME2_T R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.08
      ID=6 Name=ME2_W R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.07
      ID=7 Name=ME2_R R_Sensitvity_Type=N C_Sensitvity_Type=X
CV=0.04
      ID=8 Name=ME23_T R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.054
      ID=9 Name=ME23_ER R_Sensitvity_Type=X C_Sensitvity_Type=N
CV=0.02
      ID=10 Name=ME3_T R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.08
      ID=11 Name=ME3_W R_Sensitvity_Type=D C_Sensitvity_Type=N
CV=0.07
      ID=12 Name=ME3_R R_Sensitvity_Type=N C_Sensitvity_Type=X
CV=0.04
    .End_Global_Variation
```

Chapter 23: Variability Analysis Using the Variation Block

Interconnect Variation in StarRC with the HSPICE Flow

```
.End_Interconnect_Variation  
.End_Variation
```

In the previous example,

- The ID field must be a non-negative integer. HSPICE uses the ID field to link variation information to the sensitivity in the C and R records.
- The Name field is alphanumeric and should not contain any white space or meta characters. Use the Name field only for output annotation.
- The CV field is numeric and the CV field is interpreted as the standard deviation for a (default) normal distribution.

2: Model Card in the Header Section

The purpose of the model card in the header section is to communicate to HSPICE the model name used in the parasitic section for the resistors as well as the reference temperature. The reference temperature is equal to the GLOBAL_TEMPERATURE in ITF with units in degree Celsius.

Syntax

```
.model model_name R Tref=global_temperature
```

Example

```
.model resStar R Tref=25
```

3: Parasitic Section

The resistance and capacitance records take the form:

```
Cxxx node1 node2 val SENS [param_id, param_id, ...]=  
      [sens_coeff, sens_coeff, ...]  
Rxxx node1 node2 model_name R=val TC1=val TC2=val  
SENS [param_id, param_id, ...] = [sens_coeff, sens_coeff, ...]  
.....
```

Examples

A C record in NETNAME format is as follows:

```
C1 G2[21]:F12 Y2:897 0.699 Sens [0,1,5,6] =  
    [0.009,0.001,0.006,0.010]  
C2 X3:962 RX[12]:F74 0.324 Sens [0,1,5,8] =  
    [0.010,0.006,0.017,-0.003]
```

An R record in NETNAME format is as follows:

```
R1 G2[21]:F12 G2[21]:8 resStar R=0.699 TC1=0.0023 TC2=4e-7  
    Sens [5,6,7] = [0.51,0.64,0.86]
```

The `Sens` keyword defines the start of sensitivity information and the two vectors are the sparse sensitivity indices and the corresponding values. The first vector may contain only ordered non-negative integers that map to the `Interconnect_Variation` section, while the second vector of real numbers is interpreted as the sensitivities. The lengths of the two vectors must match. There must be one blank space between the `Sens` keyword and the sensitivity indices.

Note: For interconnect output, see [Interconnect Output Formats](#) in [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#).

Control Options and Syntax

You can specify options, one per logical record in a Variation Block. Several of the listed options are useful if a Variation Block is part of a model file that a designer cannot edit. However, you can add a Variation Block with options to control how the contents of all Variation Blocks are used in the analysis. For Monte Carlo-specific options: see [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#)

Note: No period is required before the word `Option` in the Variation Block, and is, in fact, illegal.

The following options can be specified within the Variation Block:

- `Option Ignore_Variation_Block=Yes` Ignores the Variation Block and executes earlier style variations (traditional Monte Carlo analysis). By default, the contents of the Variation Block are executed and other definitions (`AGAUSS`, `GAUSS`, `AUNIF`, `UNIF`, `LOT`, and `DEV`) are ignored. Previous methods of specifying variations on parameters and models are not compatible with the Variation Block. Thus no changes are required in existing netlists other than adding the Variation Block.
- `Option Ignore_Local_Variation=Yes` Excludes effects of local variations in simulation. Default is `No`.
- `Option Ignore_Global_Variation=Yes` Excludes effects of global variations in simulation. Default is `No`.
- `Option Ignore_Spatial_Variation=Yes` - Excludes effects of spatial variations in simulation. Default is `No`.

Chapter 23: Variability Analysis Using the Variation Block

Control Options and Syntax

- Option `Ignore_Interconnect_Variation=Yes` - Excludes effects of interconnect variations in simulation. Default is `No`. (See [Interconnect Variation in StarRC with the HSPICE Flow](#).)
- Option `Vary_Only Subckts=SubcktList` - Use either this option for the following option, but not both, to limit variation to the specified subcircuits. Actual subcircuit names are specified here (not the hierarchical names).
- Option `Do_Not_Vary Subckts=SubcktList` - Excludes variation on the specified subcircuits. Use either this option to limit variation to the specified subcircuits or the one above, *but not both*. Actual subcircuit names are specified here (not the hierarchical names).
- Option `Other_Percentile=data_block_name` - Use this option to specify quantiles lower than 1 percent when data mining (`.mpp0` file) in conjunction with `.DATA=percentile` in a netlist to get results for user-provided percentiles. This option allows you to help to see how much impact there is from trailing data points, or to count samples near the absolute minimum for a sample set. See [Using the Other_Percentiles Option on page 865](#).

References

- [1] K. Singhal and V. Visvanathan: Statistical device models from worst-case files and electrical test data. IEEE Trans. Semiconductor Manufacturing, November 1999. (Global variation modeling by principal components)
- [2] P.G. Drennan and C.C. McAndrew: Understanding MOSFET mismatch for analog design. IEEE J. Solid-State Circuits, March 2003. (Modeling mismatch in nanometer technologies)

Chapter 23: Variability Analysis Using the Variation Block
References

Monte Carlo Analysis—Variation Block Flow

Describes enhanced Monte Carlo analysis in HSPICE using Variation Block.

For information on bisection in conjunction with Monte Carlo, see [Chapter 30, Timing Analysis Using Bisection](#).

HSPICE ships numerous examples for your use. See [Variability Examples](#) for paths to demo files.

The following sections discuss these topics:

- [Overview: Monte Carlo Using the Variation Block Flow](#)
- [Monte Carlo Analysis in HSPICE](#)
- [Sampling Options](#)
- [Comparison of Sampling Methods](#)
- [Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo](#)
- [Application Considerations](#)
- [Troubleshooting Monte Carlo-VB Issues](#)
- [References](#)

Overview: Monte Carlo Using the Variation Block Flow

Monte Carlo analysis is the generic tool for simulating the effects of variations in device characteristics on circuit performance. HSPICE expresses the variations in device characteristics as distributions on the underlying model parameters. For each sample of the Monte Carlo analysis, HSPICE assigns random values to these parameters and executes a complete simulation, to

Chapter 24: Monte Carlo Analysis—Variation Block Flow

Overview: Monte Carlo Using the Variation Block Flow

produce one or more measurement results. The series of results from a particular measurement represents a distribution in statistical terms; for example, mean value and standard deviation (σ). With an increasing number of samples, the shape of the distribution gets a better definition with the effect that the two quantities converge to their final values.

You can analyze the results by arranging them in bins. Each bin represents how many results fall into a certain range (slice) of the overall distribution. A plot of these bins is a histogram that shows the shape of the distribution as the number of results versus slice. As the number of samples increases, the shape of the histogram gets smoother.

The ultimate interest of Monte Carlo simulation is to find out how the distribution in circuit response relates to the specification. Such a simulation considers these aspects of yield here:

- What is the percentage of devices which meet the specification?
- Is the design centered with respect to the specification?

The aspect of over-design is closely related. This is when the circuit characteristics are within specification with a wide margin, which could be at the expense of area or power and ultimately cost.

A typical design process is iterative, first for finding a solution which meets the nominal specification, and then moving on to a solution that meets yield and economic constraints, including the effects of variations in device characteristics. In this optimization process, it helps to understand the relationship of the design parameters to the circuit response, and the relationships of the different types of circuit response. This information is available after running Monte Carlo analysis and you can best present this data by Pairs Plots. This is a matrix of two-dimensional plots for investigating pair-wise relationships and exploring the data interactively. HSPIICE does not produce such plots, but makes the necessary data available from Monte Carlo simulation. [Figure 133 on page 787](#) shows an example of a Pairs Plot from a simple resistive divider.

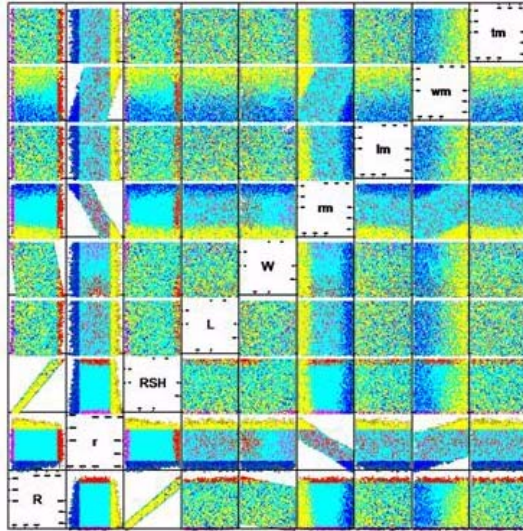


Figure 133 Pairs Plot example

An application note, “Pairs Plots from HSPICE Monte Carlo,” describes the basic ideas and includes a MATLAB script to create such a plot. Contact the Synopsys Support Center for a copy of the application note.

Monte Carlo analysis is computationally expensive, so other types of analysis may produce certain results more efficiently. For cases where only the effects of variations on the DC or AC response of a circuit is of interest, you can use DCMatch/ACMatch analyses (see [Chapter 25, Mismatch Analyses](#)).

Monte Carlo Analysis in HSPICE

Monte Carlo analysis has long been available in HSPICE and it uses two approaches:

- Define distributions on global parameters (using AGAUSS, GAUSS, UNIF, and AUNIF) in a netlist. For example:

```
.param var=AGAUSS(20,1.2,3)
```

- Define distributions on model parameters with DEV and LOT constructs in a model file. For example:

```
vth0=0.6 lot/0.1 dev/0.02
```

[Chapter 22, Monte Carlo—Traditional Flow Statistical Analysis](#) documents the previous two methods.

[Chapter 23, Variability Analysis Using the Variation Block](#) describes the Variation Block approach that satisfies some key requirements for modern semiconductor technologies. This new approach is not always compatible with the earlier ones; see the first option in [Monte Carlo-Specific Variation Block Options on page 791](#) for ways to select one or the other method.

[Figure 134 on page 789](#) shows the Monte Carlo simulation flow when you specify global and local variations.

- Sample number 1 of a Monte Carlo analysis always executes with nominal values and no variation.
- For subsequent samples, HSPICE updates the parameters specified for variation in the Variation Block with random values.
- For global variations, the same random value for all elements that share a common model changes a specified parameter.
- For local variation, a different random value for each element changes the specified parameter.
- The changes due to global and local variations are additive and saved in a file for post-processing.
- After an update of the elements, HSPICE executes the simulation and saves the measurement results. When HSPICE simulates all the requested samples, it calculates the statistics of the measurement results and includes them in the run listing.

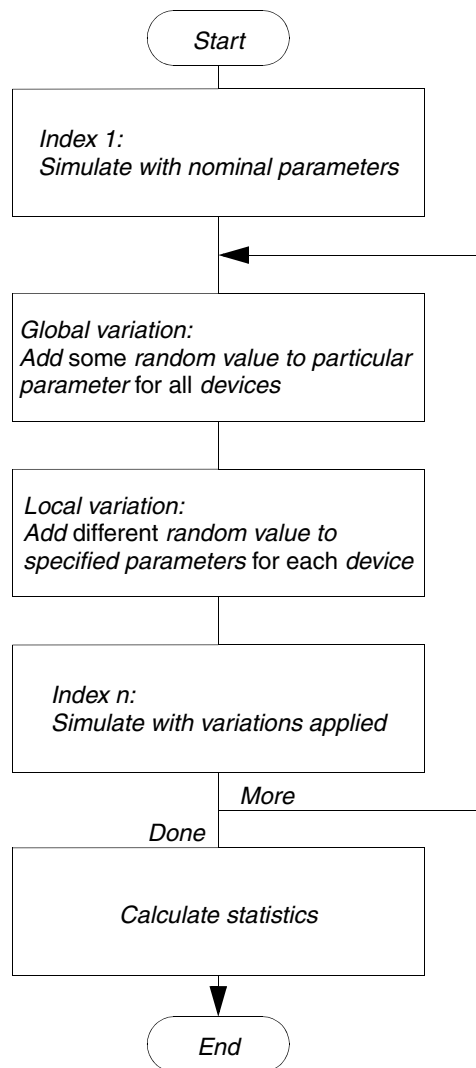


Figure 134 Monte Carlo analysis flow in HSPICE

See the following sections for these topics:

- [Input Syntax](#)
- [Monte Carlo-Specific Variation Block Options](#)
- [Output for Variation Block Monte Carlo](#)

Input Syntax

You always execute Monte Carlo analysis in conjunction with another analysis (see [Traditional Monte Carlo Analysis Characteristics](#) in Chapter 20 for full discussion):

```
.DC sweepVar start stop step [SWEEP MONTE=MCCommand]
.AC type step start stop [SWEEP MONTE=MCCommand]
.TRAN step start stop [SWEEP MONTE=MCCommand]
```

Syntax for *MCCommand*:

```
MONTE=val | list (num) | val Firstrun=num |
+ list (num1:num2 [num3] [num4:num5])
```

Parameter	Description
<i>val</i>	Specifies the number of random samples to produce.
Firstrun= <i>num</i>	Specifies the sample number on which the simulation starts.
list (<i>num</i>)	Specifies the sample number to execute.
list (<i>num1:num2 num3 num4:num5</i>)	Executes samples from num1 to num2, sample num3, and samples from num4 to num5.

The parameter values and results are always the same for a particular sample, whether generated in one pass or using *Firstrun* or the *list* syntax (see [Monte Carlo Setup on page 710](#)). Therefore, you can split Monte Carlo analyses or distribute them across multiple machines. See [Running Distributed Processing \(DP\) on a Network Grid on page 50](#) in [Chapter 4, Multiple Simulations, DP, and HPP](#).

DC Sweep Examples

These examples apply a DC sweep to a parameter *k*. The first case produces 10 samples. The second case produces five samples, starting with sample number 6. The last two examples simulate samples 5, 6, 7, and 10.

```
.dc k start=2 stop=4 step=0.5 monte=10
.dc k start=2 stop=4 step=0.5 monte=5 firstrun=6
.dc k start=2 stop=4 step=0.5 monte=list (5:7 10)
```

Monte Carlo-Specific Variation Block Options

When you execute simulations by using the Variation Block, HSPICE ignores control options for the traditional Monte Carlo style (see [Control Options and Syntax](#) in Chapter 19).

However, you can specify the following Monte Carlo-specific options in the first section of the Variation Block:

- `Option Use_AGAUSS_Format = Yes|No`
Enables you to combine traditional Monte Carlo Gaussian trials (with the exclusion Lot/Dev) with Variation Block advanced sampling methods (see [Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo](#) in this chapter).
- `Option Random_Generator = [Default | MSG]`
Specifies the random number generator used in Variation Block-based Monte Carlo analysis. `Random_Generator=MSG` invokes the generator from releases prior to 2007.09. `Random_Generator=Default` uses a long cycle generator.
- `Option Stream =[x | Random | Default]`
Specifies an integer stream number for random number generator (only for Variation Block). The minimum value of `x` is 1, the maximum value of `x` is 20; If `Stream=Random`, HSPICE creates a random stream number between 1 and 20 according to the system clock, and prints it in the `*.lis` file for later use. `Stream=Default` is equivalent to `Stream=1`.
- `Option Seed=x | random`
Where `x` is a positive integer from 1 to 259200. Setting `Random` allows HSPICE to select an integer from the range. This option also works for AGAUSS-style Monte Carlo when you use advanced sampling methods.
Note: `Option Seed` is only valid for the random number generator of MOA and overrides the setting of `Option Stream`. Use `Stream` only when `Seed` is not set.
- `Option Normal_Limit=Value`
Limits the range for the numbers generated by the random number generator for normal distributions. The default value is 4, that is, numbers in the range ± 4 are generated. The range allowed is 0.1 to 20. (When set to `inf`, `inf=20`.)

- `Option Output_Sigma_Value=Value`
This option helps in reporting results in terms of sigma values which are typically 1, 3, or 6 sigma-based on the standards used in different companies. Default is 1, range is 1 to 10. This does not affect the input sigma.
- `Option Print_Only Subckts=SubcktList`
Use either this option or the earlier to limit output in the `*.mc#` file to the specified subcircuits or the following one. *Do not use both*. Specify actual subcircuit names here (not the hierarchical names). See [Parameter File Parameter File on page 795](#).
- `Option Do_Not_Print Subckts=SubcktList`
Use either this option or the next to exclude output from the specified subcircuits to the `*.mc#` file. *Do not use both*. Specify actual subcircuit names here (not the hierarchical names). See [Parameter File Parameter File on page 795](#).
- `Option MC_File_Only=yes|no`
Use this option to generate a random number sample file (`*.mc0`) without invoking any analysis (applicable to AGAUSS style also). The feature is useful for an external block sampling simulation when you want to modify the samples before running the Monte Carlo simulation. If the netlist has a Monte Carlo command, then the MC command provides the number of samples; if the netlist has no MC command, then the number of samples is zero.
- `Option External_File=filename`
Use this command to enable read-in of an external block line-by-line-during the simulation stage. This command distributes memory consumption and avoids overtaxing the front end with a data block containing large samples. This option is also available for DP + DC Monte Carlo. For an example, see [Controlling the Read-in of an External File on page 808](#).
- `Option Add_Variation=yes`
Use this option to amplify local variation of the model parameters, especially when variation is provided by a foundry. Usually, the base variation is set by the foundry, but with this option you can add variation on model parameters based on a multiplier you supply when using a combined Variation Block and AGAUSS-style simulation. For example:

In this example, the first four lines are variations provided by the foundry.
`Option Add_Variation=yes` and `.Option Sampling_Method` are user-

supplied required options and `nmos nch_mac.nch tox= 10%` is the VB global variation where `nch_mac` is the subckt name and `nch` is the binned model name.

```
.lib 'mismatch_totalflag_b.1' stat
.lib 'mismatch_totalflag_b.1' global
.lib 'mismatch_totalflag_b.1' total
.lib 'mismatch_totalflag_b.1' tt
.Variation
  Option_Add_Variation=yes
.Global_Variation
  nmos nch_mac.nch= 10%
.End_Global_Variation
.Local_Variation
  nmos nch_mac.nch tox= 10%
.End_Local_Variation
.End_Variation
```

- `Option Other_Percentile=data_block_name`
Use this option to specify quantiles lower than 1 percent. This option allows you to help to see how much impact there is from trailing data points, or to count samples near the absolute minimum for a sample set. Refer to [Using the Other_Percentiles Option on page 865 in Chapter 26, Monte Carlo Data Mining](#) for more information.
- `Option Mirror_Components = instanceList` Use this option to specify the list of instances. The instance list uses the same set of random values in Monte Carlo simulation. This option does not support external sampling, the sampling values in external data block always has higher priority. This option supports SRS, LHS, Factorial, OFAT, Sobel, Niederreiter sampling methods. This option also supports wildcard instance name matching.

Note: This option is supported in:

- VB local/element and AGAUSS type variation only.
- Monte Carlo simulation only.

Example for Ignore_Global and Normal_Limit Options

In the following example, global variations are not simulated, and the normal distributions are exercised to ± 6 . For information regarding Local and Global Variations, see [Subblocks for Global, Local, and Spatial Variations](#) in the chapter [Variability Analysis Using the Variation Block](#).

Chapter 24: Monte Carlo Analysis—Variation Block Flow

Monte Carlo Analysis in HSPICE

```
.Variation
    Option Ignore_Global_Variation=Yes
    Option Normal_Limit=6
.Global_Variation
    Definitions for global variations
.End_Global_Variation
.Local_Variation
    Definitions for local variations
.End_Local_Variation
.End_Variation
```

Output for Variation Block Monte Carlo

The following sections cover these topics:

- [Simulation Listing](#)
- [Measurement Output File](#)
- [Parameter File](#)

Simulation Listing

The output listing file contains a summary of the names of all input parameters that are subject to global or local variations. The measured results print for each sample. Then HSPICE reports the statistics for the measured data.

Partial printout of an output listing:


```

MONTE CARLO DEFINITIONS
Random number generator is default, and stream = 1
Global variations:      model          parameter
                       snps20n       vth0
                       snps20n       u0

Local variations:      model          parameter
                       snps20n       vth0
                       snps20n       u0

Element variations:    element        parameter
                       r1            r

*** monte carlo index =      1 ***
    systoffset1= 1.3997E-03
*** monte carlo index =      2 ***
    systoffset1= -9.2694E-04
  
```

```

MONTE CARLO STATISTICS
meas_variable = systoffset
mean  = 1.4398m      varian = 1.2391u
sigma = 1.1132m      avgdev = 893.3815u
max   = 5.3035m      min    = -1.4532m
1-sigma = 1.1132m    median = 1.4184m
  
```

More detailed statistics are given in the `*.mpp` file (see [Summary Statistics on page 860](#) and [Variable Screening on page 868](#) in [Chapter 26, Monte Carlo Data Mining](#)).

Measurement Output File

Measure commands save simulation results for each sample, along with its index number. Depending on the analysis type, the name of the result file has an extension of `.ms#`, `.ma#`, or `.mt#`, where # denotes the regular sequence number for HSPICE output files.

Parameter File

A file with an extension of `*.mc#` saves the changes in all parameter values subject to variation. The structure of this file is similar to regular measure files. The header section presents the names of the parameters and independent variables as follows:

- For independent variables:
 Variable_Name:@:ID

Chapter 24: Monte Carlo Analysis—Variation Block Flow
 Monte Carlo Analysis in HSPICE

- For global variation on model parameter:
`Model_Name:@:Parameter_Name:@:ID`
- For local variation on element parameter:
`Element_Name:@:Parameter_Name:@:ID`
- For local variation on element/model parameter:
`Element_Name:@:Parameter_Name:@:Model_Name:@:ID`
- For local variation + AGAUSS function on a subcircuit parameter:
`Element_Name.parameter:@:Parameter_Name:@:Subckt_Name:@:ID`
- For top-level parameters (see [Variations on Top-Level Parameters on page 762](#)):
`top:@:parameter_name:@:GGR` or `top:@:parameter_name:@:GGA`
 for absolute and relative variation, respectively
- For temperature variation (see [Variations on Temperature on page 762](#)):
`temp:@:temp:@:IGN` for implicit independent variable with normal distribution or `temp:@:temp:@:GGA` for dependent variable absolute variation
- For interconnect variation:
`Param_Name:@:ID` and `Element_Name:@:ID` where ID is a 3-character string for identifying the type of the parameter

[Table 76](#) lists the independent parameter types, respectively.

Table 76 Independent Parameter Type Identifier

First character	Second character	Third character
I	Independent variable	G Global L Local S Spatial G or L Limit I Interconnect
		N Normal distribution U Uniform distribution C Custom distribution L Limit distribution

[Table 77](#) lists the Independent Random Variable Types and Keywords for Variation and AGAUSS in *.mc# files. “N”, “U,” and “C” are normal, unified, and custom distributions, respectively.

Table 77 IRV types/keywords for VB and AGAUSS format sampling in mc0# files

IRV Type	Keyword
Variation Block (Three-Letter Keyword String)	

Table 77 IRV types/keywords for VB and AGAUSS format sampling in mc0# files

IRV Type	Keyword
N(), Global	IGN
U(), Global	IGU
C(), Global	IGC
L() Global (limit distribution)	IGL
N(), Local	ILN
U(), Local	ILU
C(), Local	ILC
L(), Local (limit distribution)	ILL
N(), Spatial	ISN
U(), Spatial	ISU
C(), Spatial	ISC
N(), Interconnect	ITN
U(), Interconnect	ITU
C(), Interconnect	ITC
AGAUSS Format (Four-Letter Keyword String)	
GAUSS()/AGAUSS()	IGNC
UNIF()/AUNIF()	IGUC
GAUSS()/AGAUSS(), Global (Limit Distribution)	IGLC
GAUSS()/AGAUSS(), Local	ILNC
GAUSS()/AGAUSS(), Local	ILUC
GAUSS()/AGAUSS(), Local (Limit Distribution)	ILLC

[Table 78](#) lists the dependent parameter type identifiers.

Table 78 Dependent Parameter Type Identifier

First character		Second character		Third character	
M	Model	G	Global	R	Relative
E	Element	L	Local	A	Absolute
S	Subcircuit Variables	S	Spatial		

Table 78 Dependent Parameter Type Identifier

First character	Second character	Third character
T	Interconnect	

The independent variables include explicitly specified random variables (for example: $A=N()$), and the internally generated random variables for implicit definitions in the expressions for sigma (for example: $Nmos\ snps20\ vth0=0.07$). HSPICE reports values for parameters that have absolute variation specified in the Variation Block as absolute deviation from the nominal value. The output reports values for parameters that have relative variation specified as a relative deviation in percent. If the netlist, or the model, or both are encrypted, HSPICE prints hash codes in the appropriate places, which are meaningful to HSPICE for [External Sampling](#).

Generating the *.mc Output File

With the 2011.09 HSPICE release, the Monte Carlo feature generates the random value sample file, *.mc#, output in an efficient way, by printing only one file in the error check stage. The *.mc file eliminates duplication of information that appears in other files, for example, “alters” and “status” columns, which appear in measurement file outputs. Independent random variable data (IRV) are only printed to the *.mc file.

For variations defined with Variation Block style, related variations such as $xxx:@:xxx:@: MGA$, $xxx:@:xxx:@: EGA$, and so forth, are omitted because such information is not required for data mining and unnecessarily increases the file size.

The values in the AGAUSS style *.mc file are the values from the standard normal or uniform distribution and not the derived values after accounting for mean and sigma.

The *.mc file incorporates data *that had been displayed in* *.mcs#, *.mca#, and *.mct# files, which HSPICE no longer generates.

If there are multiple Monte Carlo commands in the netlist, only one *.mc file is generated in each .ALTER simulation.

In an *.mc# file, the sampling number is selected from the largest of all simulations. If Monte Carlo commands are specified with complex options, such as,

```
.dc step start stop sweep Monte = list(xx, xxx, xxx)
```

HSPICE only exports the numbers defined in the list() option.

Character separators in *.mc files are indicated by the :@: (colon-"at"-sign-colon) combination. This combination avoids confusion with hierarchy separators which use a single @ character.

Example: *.mc# File

```

index          snps20n:@:vth0@IGN          snps20n:@:u0:@:IGN
               snps20n:@:vth0:@:MGA          snps20n:@:u0:@:MGR
               xi82.mn6:@:snps20n:@:vth0:@:MLA
xi82.mn6:@:snps20n:@:u0:@:MLR
               xi82.mn1:@:vth0@ILN          xi82.mn1:@:u0@ILN
               xi82.mn1:@:snps20n:@:vth0:@:MLA
xi82.mn1:@:snps20n:@:u0@MLR
               xi82.mn2:@:vth0:@:ILN          xi82.mn2:@:u0:@:ILN
               xi82.mn2:@:snps20n:@:vth0:@:MLA
xi82.mn2:@:snps20n:@:u0:@:MLR
               xi82.rcomp:@:r:@:ILN          xi82.rcomp:@:r:@:ELR
               status          alter#
1.0000         0.              0.              0.
               0.              0.              0.
               0.              0.              0.
               0.              0.              0.
               0.              0.
               1.0             1.0000
2.0000         0.6141         0.6284         4.299e-02
               6.284e-02      2.1837         0.2184
               1.7554         0.1755         1.6017
               0.1602         0.4769         4.769e-02
               -1.0088        0.5350
  
```

In this example, HSPICE first reports the changes due to the global variations on parameters vth0 (absolute) and u0 (relative). Then HSPICE reports the changes on each device due to local variations on the same parameters. Finally, HSPICE reports the local variation on the parameter r of the element rcomp. Note that the parameter value applied to the device for a particular sample is the nominal value, plus the reported change due to global variations, plus the reported change due to local variations, and so on.

The contents of this parameter file are useful for data mining (see [Chapter 26, Monte Carlo Data Mining](#)). You can investigate the relationship of circuit response variation to parameter variation in combination with the measured data in the regular output file, by using, for example, a Pairs Plot as shown in [Figure 133 on page 787](#).

Note: The contents of this file are subject to change.

Interconnect Output Formats

An example output for interconnect variation is as follows. The Monte Carlo sampling output file *.mc#, uses one identifier keyword for interconnect variation parameter. In the following, IGN is the extension for independent variables. TGA is the extension for dependent variables. The T is present for interconnect parameters:

```
$ This file format is subject to change
.TITLE '* two capacitors for model parameter variation testing'
index      fox_c_t:@:IGN      fox_a_t:@:IGN      ild_b_t:@:IGN
           imd1c_t:@:IGN      imd1d_t:@:IGN      imd2a_t:@:IGN
           r1:@:TGA           r2:@:TGA           r11:@:TGA
           r22:@:TGA          r211:@:TGA         r222:@:TGA
           c1:@:TGA           c2:@:TGA           c11:@:TGA
           c22:@:TGA
status      alter#
1.0000      0           0.           0.
           0.           0.           0.
           0.           0.           0.
           0.           0.           0.
           0.           0.           0.
           0.
           1.0           1.0000
2.0000      0.6141      0.6284      0.8866
           5.198e-02      1.2452      -1.5600
           -1.031e-02      -3.537e-04      -5.191e-02
           6.133e-05      2.464e-03      2.964e-03
           2.464e-04      3.037e-04      5.073e-04
           5.796e-04
           1.0           1.0000
3.0000      -0.1087      -0.6694      3.363e-02
           1.6842      -1.0088      0.5350
           3.551e-03      9.223e-05      1.782e-02
           -2.440e-04      -7.813e-04      1.220e-03
           -7.813e-05      -2.212e-04      5.374e-05
           1.055e-04
```

Sampling Options

HSPICE provides simple random sampling (SRS) as well as advanced sampling schemes for related applications. OFAT and Factorial sampling use statistical design-of-experiment techniques while LHS and LDS reduce the

sampling error in the results. External sampling provides a generic interface that permits users to overload the internal random number generators and pass HSPICE sample values generated from other statistical tools. See also [Comparison of Sampling Methods](#). The sampling options are as follows:

- [Simple Random Sampling \(SRS\)](#)
- [One-Factor-at-a-Time \(OFAT\) Sampling](#)
- [Factorial Sampling](#)
- [Latin Hypercube Sampling \(LHS\)](#)
- [Sobol and Niederreiter Sampling \(LDS\)](#)
- [External Sampling](#)

Simple Random Sampling (SRS)

`Option Sampling_Method = SRS`

Traditional Monte Carlo selects the samples in a random manner from the specified distributions. This is the default sampling method. You can select this through the option `Option Sampling_Method = SRS`.

In SRS (or other) samplings, on the first point (which is also a nominal point), the limit distribution values use the nominal (no perturbation) values.

One-Factor-at-a-Time (OFAT) Sampling

`Option Sampling_Method=OFAT`

This sampling method varies One-Factor-at-a-Time, a Design of Experiments feature [1]. It is useful for sensitivity studies and for constructing low-order response surface approximations. The number of samples is 2^{m+1} with m independent variables. OFAT ignores the number that the Monte Carlo command specifies, and m must be less than 2500. Sampling starts with no perturbation (nominal), then negative and positive perturbation only on the first parameter, negative and positive perturbation only on the second parameter, and so forth. In OFAT sampling, unsampled limit distributions return their nominal values, similar to SRS. The amounts of perturbation are the extreme values for a uniform distribution, and the `Normal_Limit` values for a normal distribution. [Figure 135](#) illustrates OFAT examples.

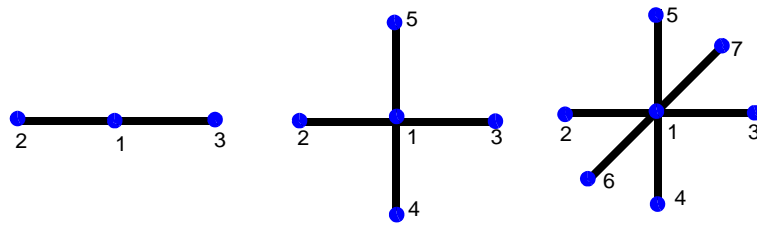


Figure 135 One-Factor-at-a-Time sampling with one, two, and three independent variables

A sub-option, `intervals=n`, generates $2n+1$ equally spaced samples along the range of each independent variable. The total number of sample points increases to $2mn + 1$. The full syntax is:

Option `Sampling_Method=OFAT Intervals=2`

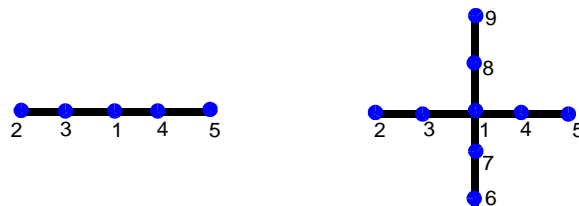


Figure 136 Suboption intervals

An alternative sub-option is to declare `Intervals = +0.05 | -0.05` to perform one-sided OFAT sampling in instances such as cell characterization. The keys `0.5` specify the one-sided samples with the samples being at 0 and `+ Normal_Limits` and key `-0.5` for one sided samples with the points being at nominal and `-Normal_Limit`.

Factorial Sampling

Option `Sampling_Method=Factorial`

Use this option to:

- Evaluate the circuit response at the extremes of variable ranges to get an idea of the worst and best case behavior.
- Create polynomial response surface approximations.

Factorial sampling evaluates the circuit at the center of the hypercube (nominal) and at all its corners (see Figure 137). There are $1+2^m$ samples for a circuit with m independent variables; this method ignores the number of samples that the Monte Carlo command specifies. To prevent large runaway jobs, HSPICE restricts the problem dimension to $m \leq 12$, which results in ~4K simulations. If you violate the size constraint, HSPICE ignores the command and generates an error message.

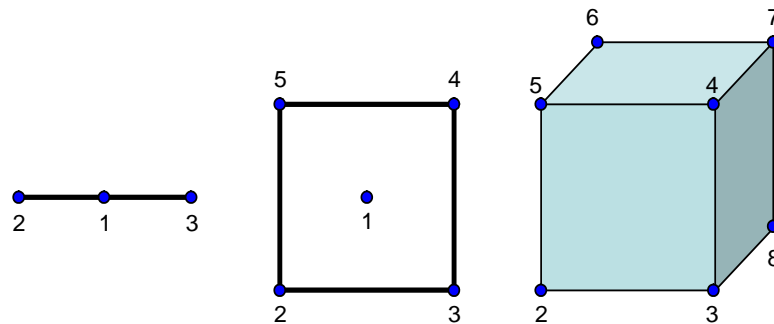


Figure 137 Factorial Hypercube Evaluation at Center and Corners for 1, 2, and 3 independent random variables

Latin Hypercube Sampling (LHS)

Option `Sampling_Method=LHS`

Latin Hypercube Sampling is an efficient sampling technique for Monte Carlo analysis of systems. These systems are modeled by computers and have large number of variable parameters [2] [3]. Advantages of LHS are:

- The estimation error is smaller on most real world problems and you can use a smaller sample size to get the same precision in the results.
- The sample points are evenly spread over the entire range of variation of each parameter.

- The circuit is exercised over a wide range of parameter values. The circuit often detects weak spots in the design.
- You can replicate the sampling using Option `Replicates=Value`
This option runs replicates of the Latin Hypercube samples. The sample with nominal conditions is simulated once. HSPICE repeats the LHS run the number of times specified by `Value`. For example, if, in a regular run, you have 10+1 (including nominal value) iterations, if you set `Replicates=2`, you generate 21 (or $2 * Value + 1$) Latin Hypercube samples.

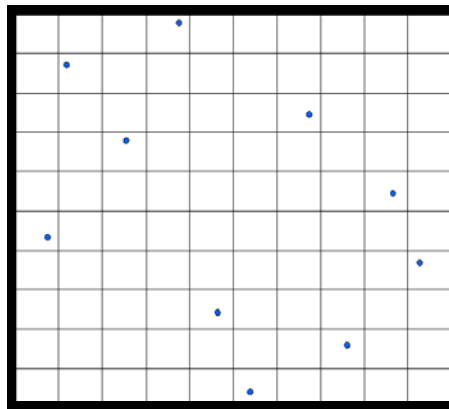


Figure 138 Example of the distribution of 10 sampling points in two dimensions

Monte Carlo with `firstrun` and `list` options:

LHS is a special sampling method. Unlike SRS, its sample values are decided by the sample size. LHS sampling with `firstrun` or `list` keyword in Monte Carlo analysis requires a new keyword `lhs_sample_size` to be set, which indicates the sample size of LHS.

Syntax:

```
.TRAN/DC/AC... sweep monte=list(num1:num2,...lastnum2,  
    lastnum1) lhs_sample_size=num
```

Or

```
.TRAN/DC/AC... sweep monte=num1 firstrun=num2  
    lhs_sample_size=num
```

Where the last monte sweep index in `list` and `firstrun` syntax `lastnum1` and `num2 - num1 - 1` respectively.

The keyword `lhs_sample_size` should be set such that the last monte index must be equal to or less than `replicates*lhs_sample_size+1`.

Note: With `firstrun`, the LHS sample size is set using `lhs_sample_size` and not with `monte` keyword. The monte number should be adjusted for the number specified in the `lhs_sample_size`.

If this condition of `lhs_sample_size` is not satisfied, then the following error message is displayed:

```
**error** (file:line) monte sweep index exceeds  
LHS_SAMPLE_SIZE limit
```

If the `LHS_SAMPLE_SIZE` is multi-defined, the maximum one will be used with the following warning message:

```
**warning** multiple definitions of LHS_SAMPLE_SIZE, the  
maximum one will be used.
```

Examples

For example, assuming `replicates=1`, if you set `monte=10`, `firstrun=8`, and `lhs_sample_size=15` then the last monte index is $10+8-1=17$, which is more than 16 (`replicates*lhs_sample_size+1`). Therefore, HSPICE displays the following error message:

```
**error** (file:line) monte sweep index exceeds  
LHS_SAMPLE_SIZE limit.
```

Sobol and Niederreiter Sampling (LDS)

Option `Sampling_Method=SOBOL`

Option `Sampling_Method=NIEDERREITER`

Two Low-Discrepancy Sequences (LDS) quasi-random number generators—Sobol and Niederreiter—support Variation Block-based Monte Carlo analysis [4] [5] [6]. LDS sample points are even more evenly distributed compared to LHS and the sampling error is lower. Dimension limitations exist for both Sobol and Niederreiter. For Sobol, the current maximum dimension is 40; for Niederreiter, the current maximum dimension is 318.

Each of these sampling options has an auto-switch flow:

- If you use Sobol with a sampling dimension of more than 40, then HSPICE switches to use Niederreiter.
- If you use Niederreiter with a sampling dimension of more than 318, then HSPICE switches to the default SRS sampling method.

Figure 139 shows the locations of 1024 samples in two dimensions for simple random sampling and the Sobol Sequence. The Sobol points are better spaced while those from SRS are “lumpy.” This is the general property of LDS.

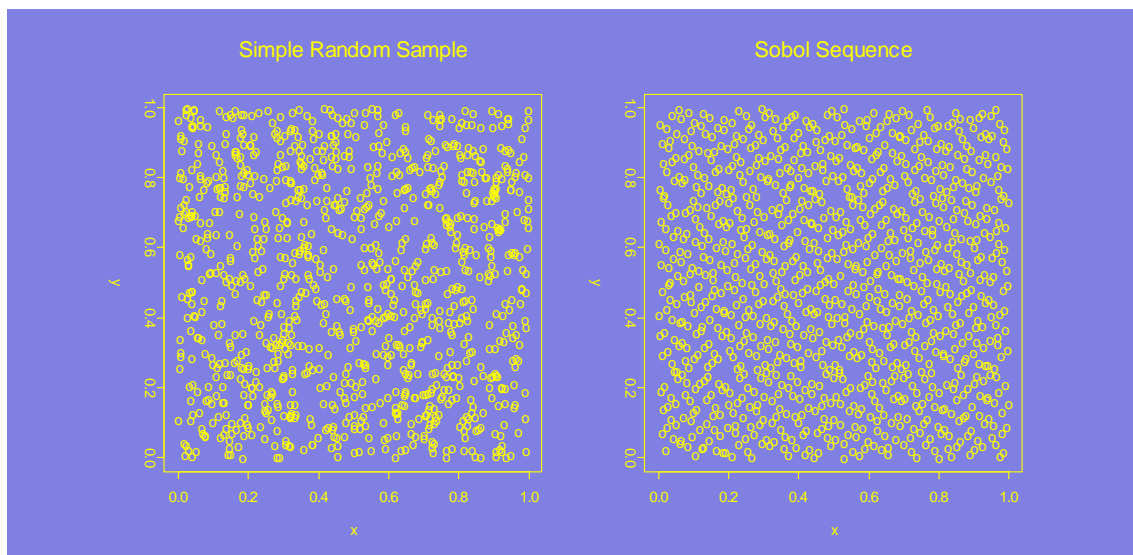


Figure 139 Comparing low-discrepancy Sequence: Simple Random versus Sobol

External Sampling

You can also execute a data set of externally created perturbations instead of relying on one of the built-in sampling methods. External sampling allows design and process exploration tools to run statistical experiments with the variables for each sample under their full control. In this way, for example, you can exclude certain transistors from the analysis (that is, have the parameters remain fixed instead of being randomly varied).

Note: External sampling does not support encrypted or protected model libraries.

The following sections provide these details:

- [Usage Model for External Sampling](#)
- [Syntax](#)
- [Controlling the Read-in of an External File](#)
- [Using the MC_File_Only Option](#)
- [Allowing Zero Variations in External Sampling for Missing IRVs](#)

Usage Model for External Sampling

Use the following procedure to add the custom-generated samples to HSPICE Monte Carlo flow:

1. Execute HSPICE with a standard simulation command (.AC, .DC, .TRAN) and `monte=1` to produce an *.mc0 file, which lists all the independent variables (see [Parameter File on page 795](#)).
2. Create a data block outside HSPICE with the desired perturbations on the independent variables for global and local variations.
3. Run an HSPICE simulation with externally generated data block content.
4. Repeat steps 2 and 3, depending on the outcome of the previous experiments.

Syntax

The external sampling feature is defined in two parts in the Variation Block, a data block and an option.

The data block syntax is the same as for the regular HSPICE data block from `.Data` to `.EndData`. The first variable is always the index. All identifiers for the variables start with “I” because this is the only variable type which can be set externally. You invoke the feature itself by specifying the external sampling method, with the appropriate block name. For example:

Chapter 24: Monte Carlo Analysis—Variation Block Flow Sampling Options

```
.Variation
.Data test1
  Index p1:@:IGN  snps20p:@:vth0:@:ILN  x1.mn1:@:q1:@:ILN  x2.mn1:@:vth0:@:ILN
1      .0          .0                .0                .0
2      .0          0.1086            .0                .0
3      .3          .0                .0                .0
4      .0          .0                1.0              .0
5      .0          .0                .0                1.0
6      .0          .0                1.0              -1.0
.EndData

  Option Sampling_Method=External Block_Name=test1
.End_Variation
```

To run a particular sample from the data block, use the following:

```
monte = list(num) or
monte = list(<num1:num2> <num3> <num4:num5>)
monte=list(n1:n2) can exceed a limit in the data block.
```

If the netlist or the model, or both, are encrypted, the hash codes printed in the parameter file are recognized by HSPICE when reading in the external sampling data block.

Additional rules:

1. HSPICE does not check the range of values in the supplied data block against option value `Normal_Limit`.
2. Independent random variables which are not specified in the data block are assigned new random values generated by a new seed.

Controlling the Read-in of an External File

Use `Option External_File=filename` to enable read in of an external data block line-by-line during the simulation stage. This command distributes memory consumption and avoids overtaxing the front end with a data block containing large samples. This option is also available for DP+ DC Monte Carlo. For example:

```
Option Sampling_Method=External Block_Name=extern_data
+ External_File=extern.mc0
.Data extern_data
...
.Enddata
```

Using the MC_File_Only Option

The `MC_File_Only` option enables you to generate a random value sample file without running an actual Monte Carlo simulation. Use the Variation Block Option `MC_File_Only=yes|no` to circumvent the need to run a double Monte Carlo simulation. The syntax is:

```
.Variation  
  Option MC_File_Only=yes|no  
.End_Variation
```

You can use this option in multiple applications during an external sampling flow. Consider a scenario where you expect that a few elements under local variation simulation must have the same variations on them. In a normal Monte Carlo simulation, every element gets a unique random value when local variations are defined on them. Thus you need to manually modify the assigned sample value to the selected elements before running the Monte Carlo simulation. Do this in HSPICE by using the External Sampling method.

The steps are:

1. Run the regular HSPICE simulation with `.Option Sampling_Method=SRS` and `MC_File_Only=Yes`. (Even though this is a Variation Block option, it is applicable to the AGAUSS style too.)

Result: This run generates an MC sampling file (`*.mc0`).

2. Manipulate the data in this file such that each random parameter corresponding to one element has the same value for each sample matching with that of other elements of interest.

Explanation: This step is simplified by printing out the `*.mc0` file in `*.csv` format which can be directly read in an Excel sheet. You can manipulate columns in the Excel sheet easily. Set `.option MEASFORM=3` in the netlist to generate the file in CSV format.

3. Convert this sample file into a DATA block by adding `.DATA blockname .ENDDATA` statements.
4. Run the MC simulation again with `.Option Sampling_Method=External Block_Name=blockname` and include the modified sample file.

Allowing Zero Variations in External Sampling for Missing IRVs

If you only want to vary a subset of transistor parameters in the netlist and avoid HSPICE creating an enormous table for devices that you do not care

about, you can declare a sub-option to `SAMPLING_METHOD=External` —
`.OPTION SET_MISSING_VALUES=Zero`.

The syntax is: `.OPTION SET_MISSING_VALUES = Random | Zero`

The default value is `Random` if the option is not specified in a netlist.

Use this option to control missing random values in a `.data` block for external sampling:

- `Set_Missing_Values=Random`: HSPICE generates its own random values for the missing random variables in a `.data` block.
- `Set_Missing_Values=Zero`: HSPICE generates zero values for those missing random variables in `.data` block in external sampling.

Syntax example:

```
.option Sampling_Method = External Block_Name = XXXX  
+ File_Name = YYYY Set_Missing_Values = Random|Zero
```

Comparison of Sampling Methods

This section provides illustrations to describe the qualitative behavior of various sampling methods.

The sampling methods are described in detail in the following:

- [Pairs Plot for SRS Samples](#)
- [1024 Points, Latin Hypercube Sample](#)
- [Four-Dimensional Sobol Sequence](#)
- [Space Filling Properties for Sobol Samples](#)
- [Smoothed Density Plots](#)
- [Samples from Multivariate Normal Densities](#)

Pairs Plot for SRS Samples

[Figure 140 on page 811](#) shows a pairs plot for 1024 samples from SRS in four dimensions with uniform distributions.

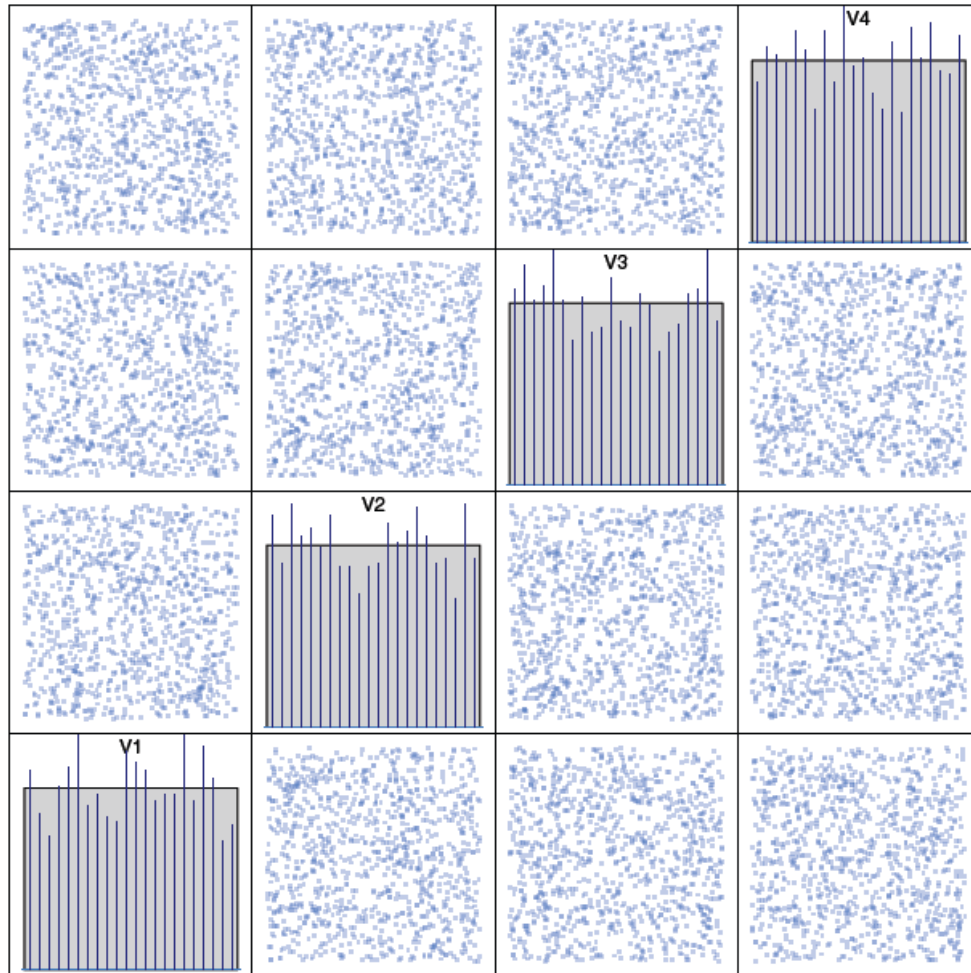


Figure 140 Sampling: SRS Distribution: uniform n ; 1024 maxCor; 7.2 percent

The diagonal subplots show the sample histogram together with the ideal uniform density as a shaded area for each of the random variables. The off-diagonal subplots give the two-dimensional projections of the sample points. Note that the 1-D projections depart from their ideal behavior. The correlation coefficients are also computed for the sample and the maximum absolute value is found to be 7 percent. As the samples are independent, the ideal value is zero. Such qualitative behavior is typical of small samples in traditional Monte Carlo.

1024 Points, Latin Hypercube Sample

Figure 141 shows 1024 LHS points, again in four dimensions and with uniform distributions. The 1-D histograms match the ideal shaded density—this is a

consequence of the way in which LHS are constructed. The 2-D projections have a somewhat better distribution with the maximum correlation being 4.5 percent for the particular sample. The correlation value changes with the random generator seed and small correlations are not currently enforced in constructing LHS in HSPICE. The correlation could be more or less than SRS in a particular experiment.

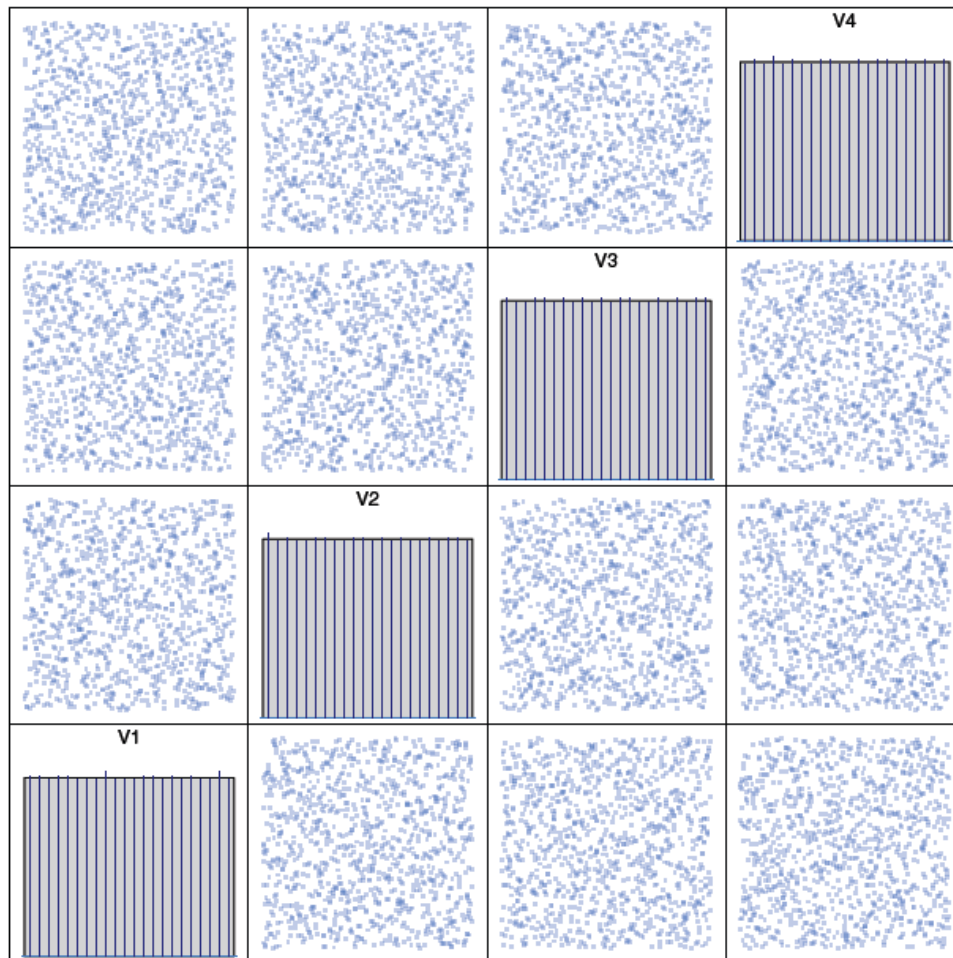


Figure 141 Sampling: LHS Distribution: uniform n ; 1024 maxCor; 4.5 percent

Four-Dimensional Sobol Sequence

[Figure 142](#) shows the first 1024 sample points from the four-dimensional Sobol sequence. Like LHS, the 1-D histograms match the ideal shaded density — this is a property of low-discrepancy sequences. The uniformity in two dimensions is also better and the largest absolute correlation coefficient is 0.5 percent.

Two-dimensional uniformity is one of the criteria used in generating low-discrepancy sequences. Notice that the Sobol sequence has a well-defined pattern that does not look random, for example, between variables one and two. The reason is that low-discrepancy points are selected by special algorithms that try to fill the space as uniformly as possible.

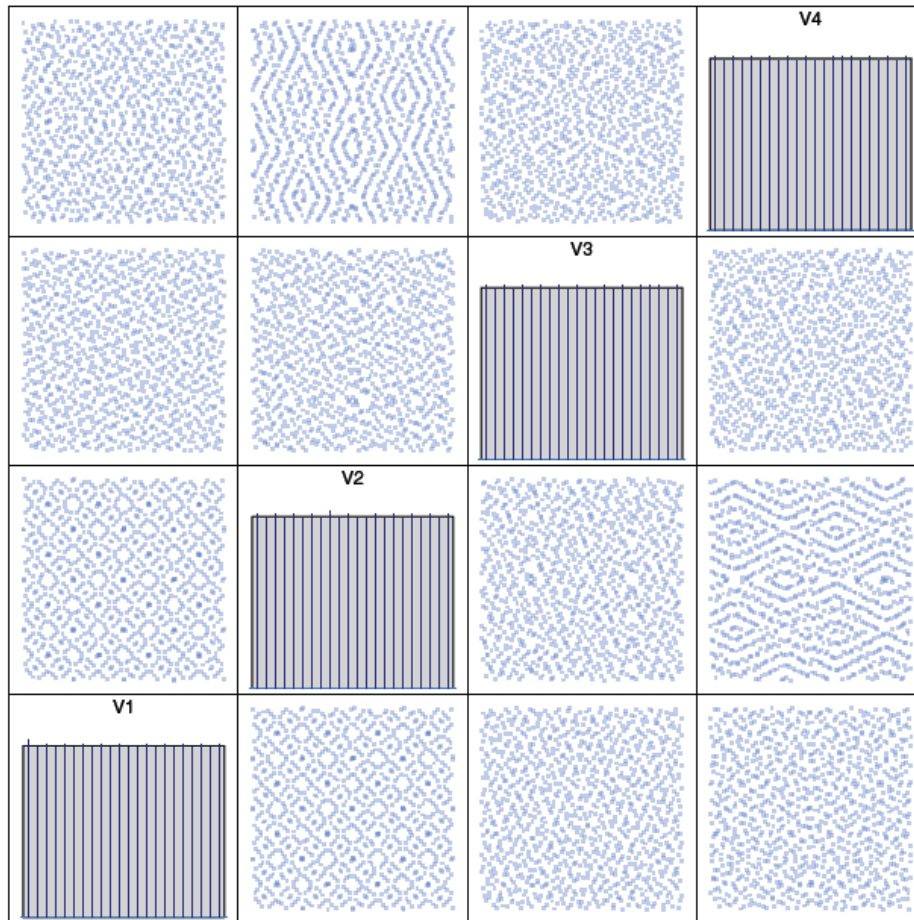


Figure 142 Sampling: Sobol Distribution: uniform n ; 1024 $maxCor$; 0.5 percent

Space Filling Properties for Sobol Samples

The space filling property is illustrated in [Figure 143 on page 814](#). It shows the 2-D projection of samples for variables one and three for sample sizes ranging from 64 to 2048. Starting in the lower left, the subplot shows the first 64 Sobol points in blue. The subplot with $n = 128$ shows the first 64 points in red (which are the same as the blue points in the first subplot) and the samples from 65 through 128 shown in blue. Proceeding in a similar manner, other subplots are

generated with the red points representing the previous samples and the blue points representing the new ones. The Sobol samples are generated in a structured manner with the new points filling the “holes” left behind by the prior samples.

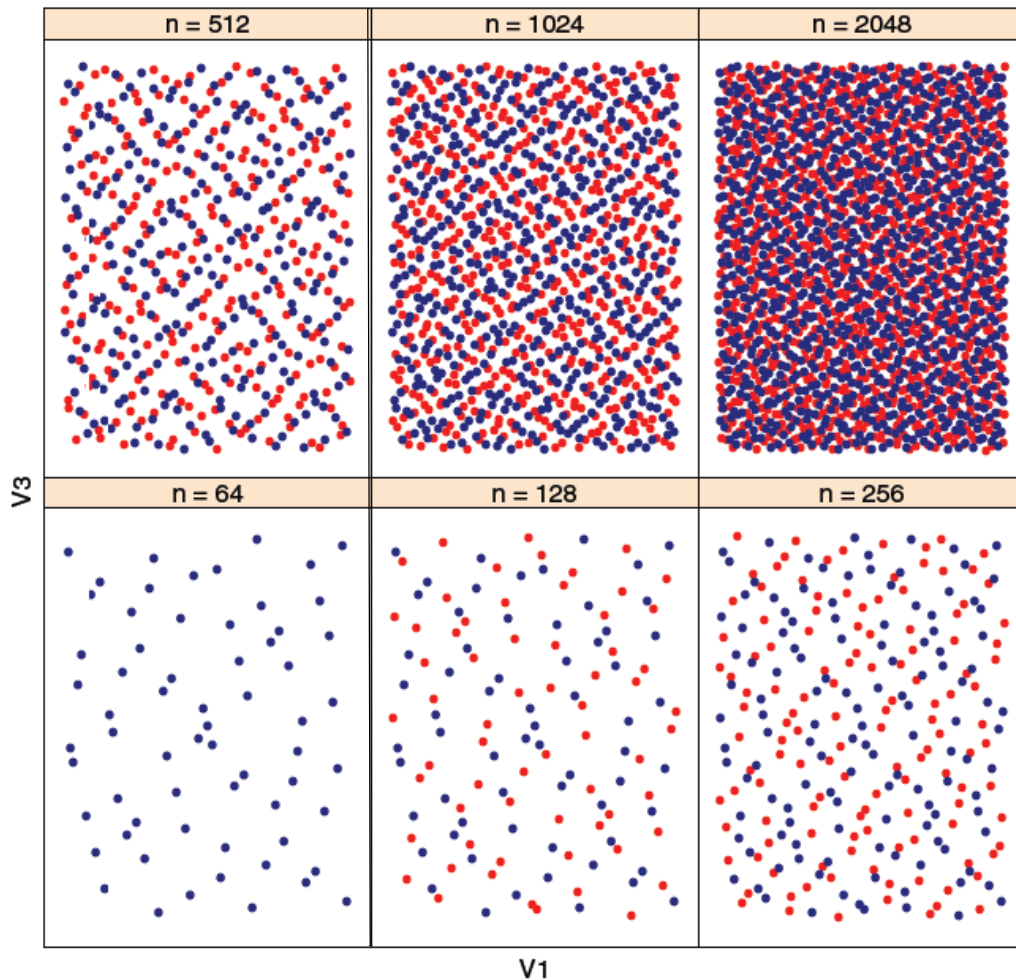


Figure 143 Space filling property of Sobol sampling

Smoothed Density Plots

While the qualitative behavior of the 1-D projections as histograms in [Figure 144 on page 815](#) through [Figure 146 on page 817](#) is clear, the quality of the 2-D projections is harder to visualize and the largest absolute correlation coefficient is not an intuitive measure. [Figure 144 on page 815](#) shows the smoothed density plots for 128 samples. The samples are from a uniform

density and the ideal plot is the one shown in the bottom right subplot. The traditional Monte Carlo, SRS, is shown in the top left subplot and departs from the ideal behavior for small sample sizes. The behavior of the LHS density depends on the specific random number generator seeds. The density for the Sobol sequence is close to the ideal and the property generally holds true for low-discrepancy sequences.

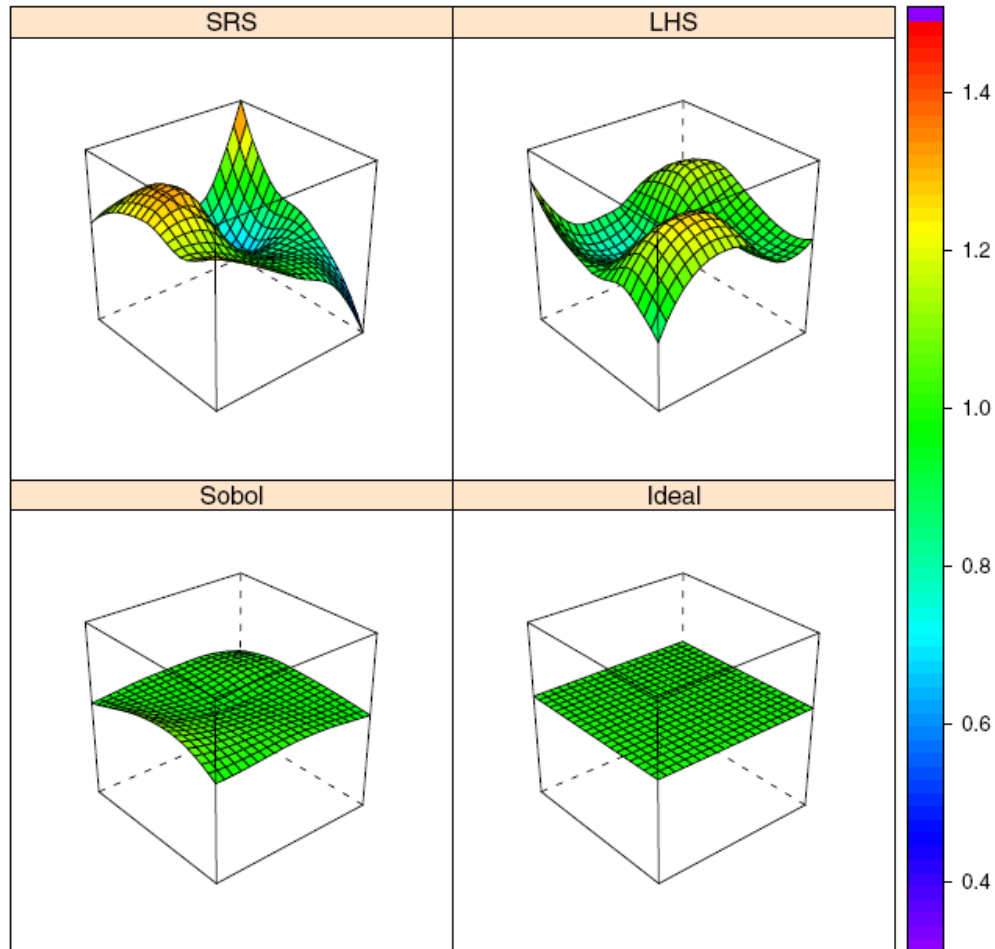


Figure 144 Comparison of smoothed density plots $n = 128$

Samples from Multivariate Normal Densities

Results from similar experiments are shown for samples from multivariate normal densities in four dimensions for SRS, LHS, and Sobol sequences in [Figure 145](#) through [Figure 147](#) on page 818. The diagonal subplots show the 1-D histograms for each variable together with the ideal shaded density. The

Chapter 24: Monte Carlo Analysis—Variation Block Flow
Comparison of Sampling Methods

off-diagonal subplots show two-dimensional densities as topographic plots with color scale from deep blue to red with red indicating larger values. Both LHS and Sobol samples match the 1-D ideal density and are better than SRS. The largest absolute correlation coefficient is smaller for Sobol samples. However, the largest absolute correlation coefficient for LHS is more than that for SRS in this example.

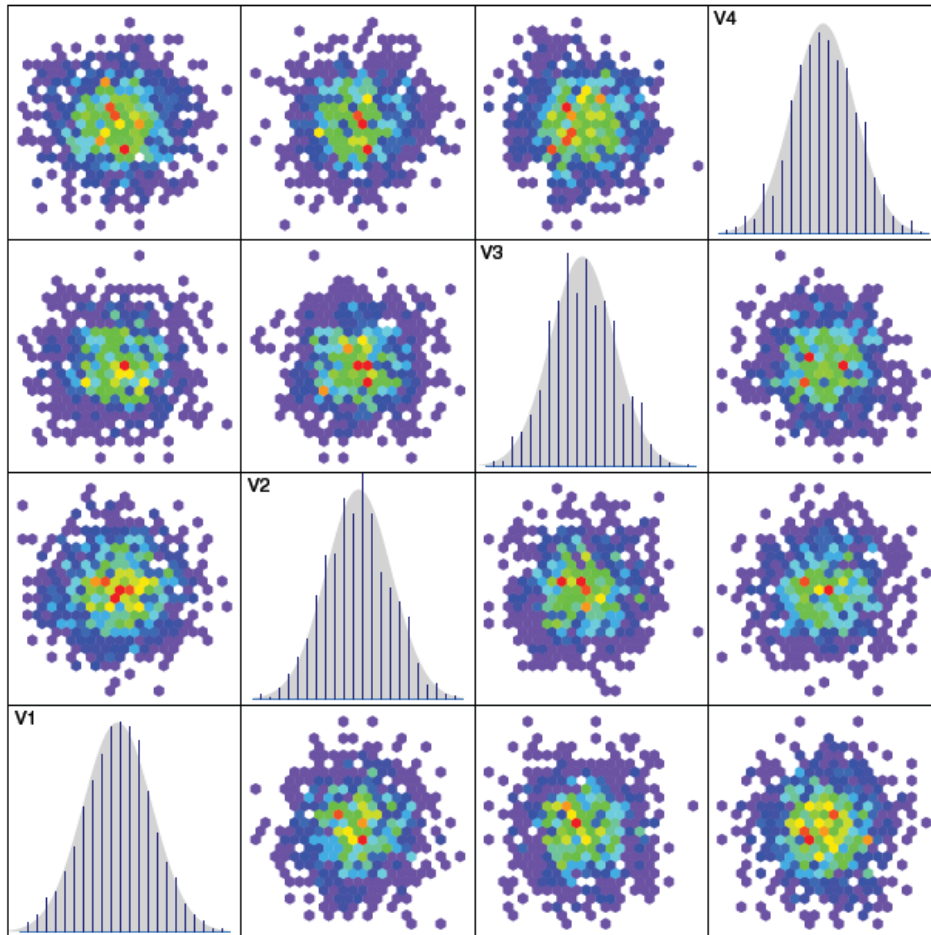


Figure 145 Sampling: SRS Distribution: normal n ; 1024 maxCor; 3.8 percent

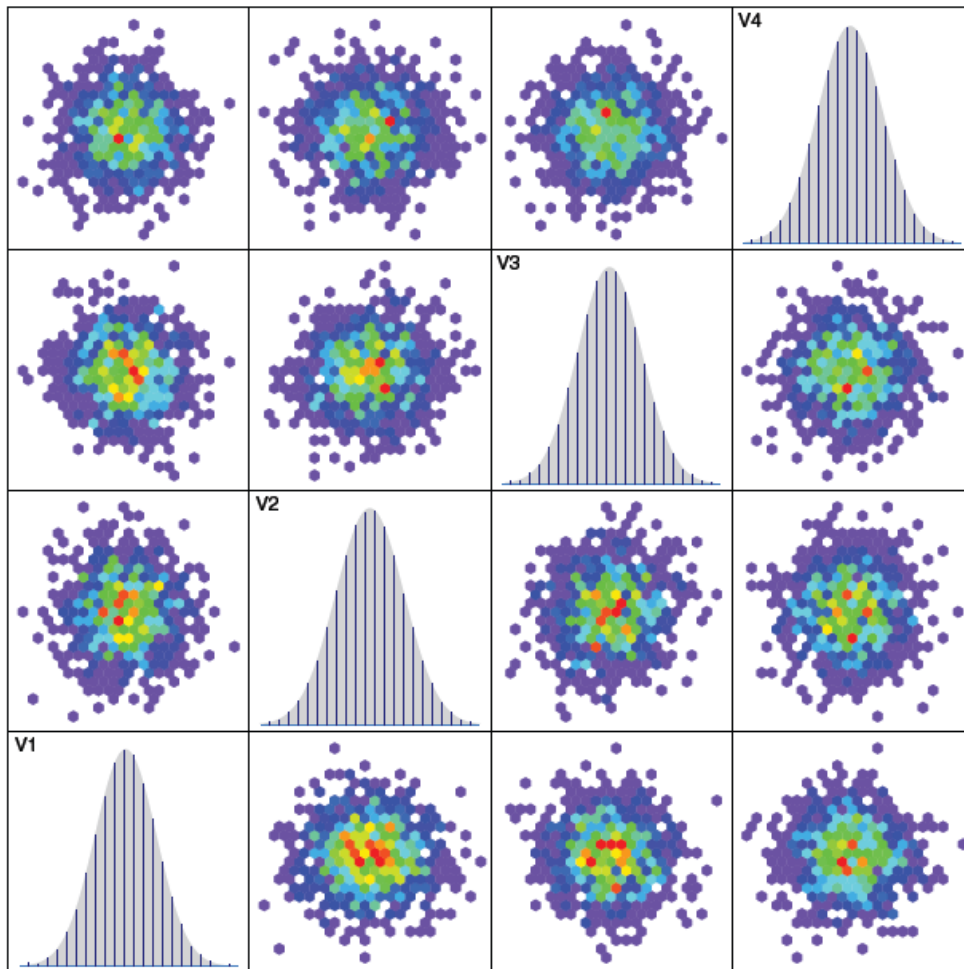


Figure 146 Sampling: LHS Distribution: normal n; 1024 maxCor; 7.2 percent

Chapter 24: Monte Carlo Analysis—Variation Block Flow
Comparison of Sampling Methods

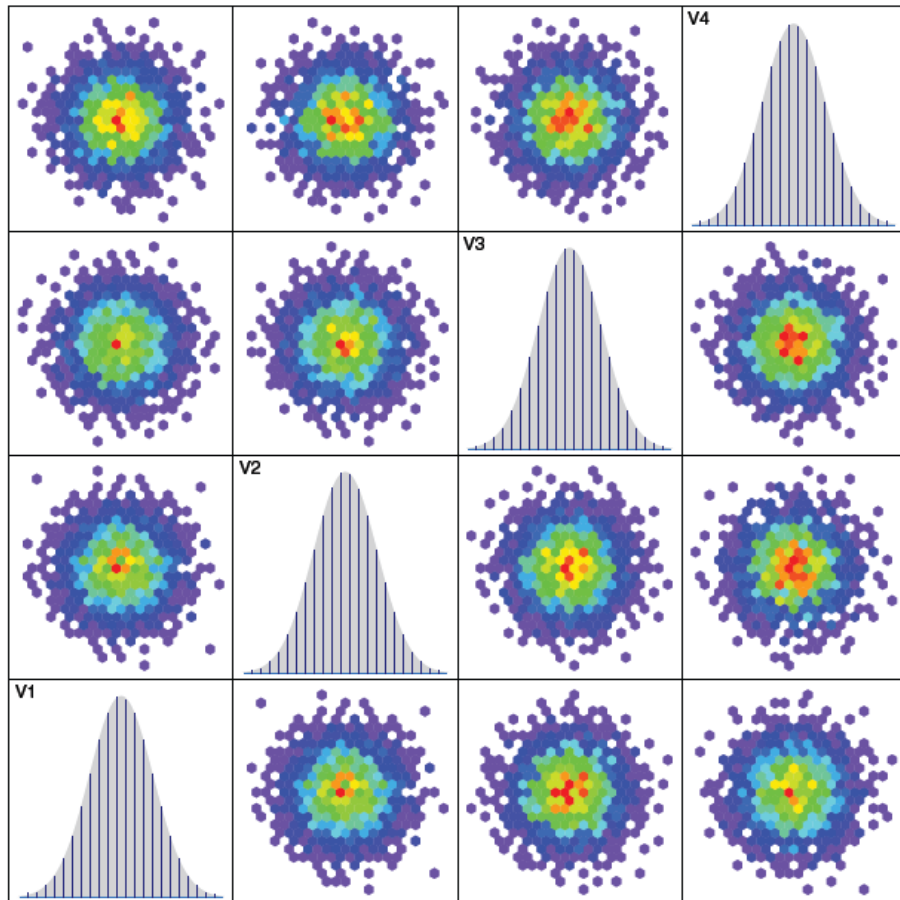


Figure 147 Sampling: Sobolj Distribution: normal n ; 1024 maxCor; 0.9 percent

A better view of the 2-D projections can be seen in the smoothed density plots shown in [Figure 148 on page 819](#) for 64 samples. As with uniform densities, we see that LHS looks better than SRS and that the Sobolj sample is much closer to the ideal density.

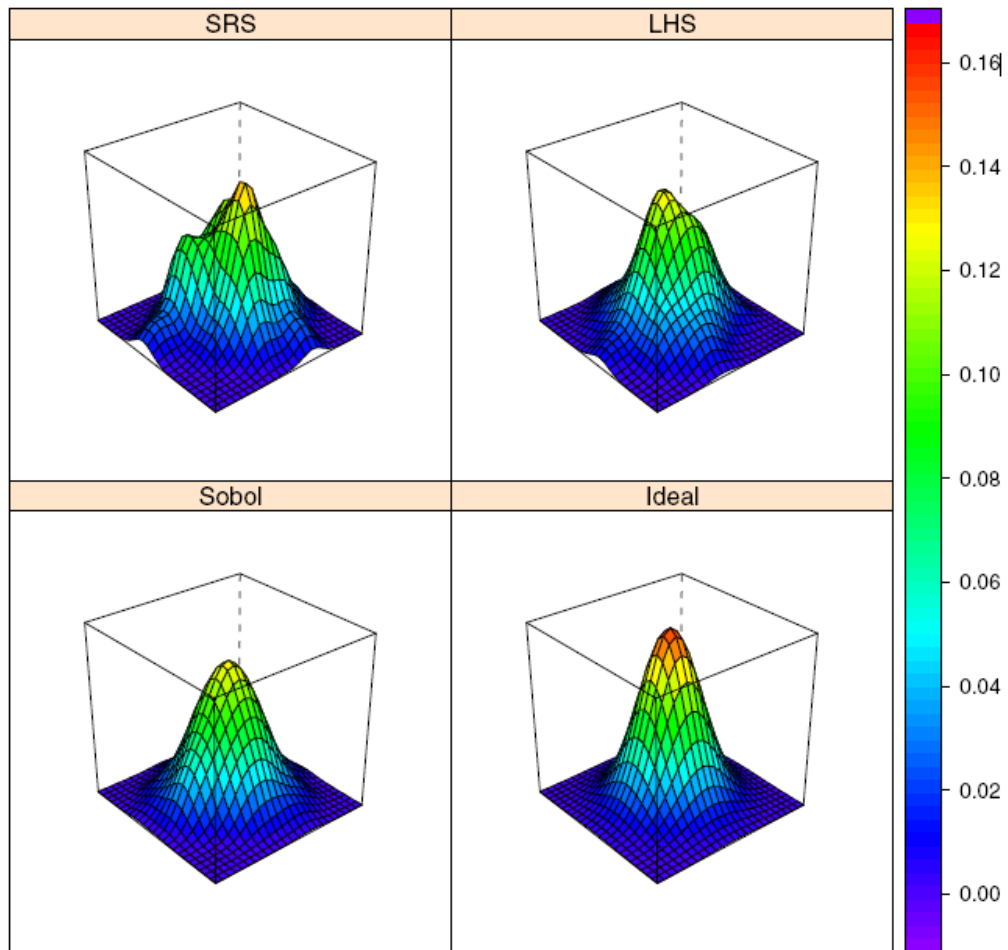


Figure 148 Comparing sampling methods, smoothed density plots: $n = 64$

Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo

You can use the following option in your Variation Block to enable HSPICE to run some advanced sampling methods that were designed for the Variation Block, including SRS, LHS, OFAT, Factorial, Sobol and Niederreiter with traditional Monte Carlo variation definitions (called `AGAUSS` here).

For example:

Chapter 24: Monte Carlo Analysis—Variation Block Flow

Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo

```
.Variation
  Option Use_AGAUSS_Format = Yes
.End_Variation
```

By specifying this option, use of both forms of definitions can work simultaneously. The default is *Yes*.

Note: An alternative to using this option inside the Variation Block structure, is to set the HSPICE global option `.OPTION SAMPLING_METHOD`, which enables use of the advanced sampling methods [SRS|LHS|Factorial|OFAT|SOBOL|NIEDERREITER] **Default:** SRS.
`.OPTION REPLICATES = value` is added for LHS sampling.

In addition to the `Use_AGAUSS_Format = Yes` option, the following Variation Block options can be applied for both style definitions:

- Option `Output_Sigma_Value = number`
- Option `Sampling_Method = OFAT Intervals = number`
- Option `Random_Generator = MOA|MSG|Default`
- Option `Normal_Limit = val`

The following options work for variations defined inside a Variation Block only:

- Option `Vary_Only|Do_Not_Vary|Print_Only|Do_Not_Print + Subckts = SubCkt1, SubCkt2, ...`

Supported models for the `Option Use_AGAUSS_Format` include: resistor, capacitor, BJT, diode, JFET, MOSFET, and independent voltage/current sources. If a model parameter variation is defined both in the traditional (AGAUSS) format style and Variation Block style, then only the variation in the Variation Block is active in Monte Carlo; the other one is ignored.

Messages in *.lis File for Sampling_Method Option

- If `.OPTION SAMPLING_METHOD` is not set, but Variation Block is defined, then HSPICE writes the following information message to the `*.lis` file:

```
Only Variation Block is active, to enable AGAUSS format
variation, set option sampling_method = SRS.
```
- If `Option Sampling_Method` is set, then HSPICE reports:

Option `sampling_method` was set, Variation Block and AGAUSS format variation are in effect.

- If both AGAUSS and VB variations are set on the same model parameters, HSPICE issues this warning message:

```
**warning** model parameter 'Model_Parameter_Name' is already defined in Variation Block; the duplicate is ignored.
```

The following topics are discussed in the next sections:

- [Gaussian Style Random Variable Definition](#)
- [Example 1: Variation Duplicated in Traditional Format and Variation Block](#)
- [Example 2: Subcircuit and Macro Models](#)

Gaussian Style Random Variable Definition

HSPICE permits independent random variables (IRV) to be defined by one of four probability distribution functions: GAUSS (relative normal distribution), AGAUSS (absolute normal distribution), UNIF (relative uniform distribution), AUNIF (absolute uniform distribution) with the following syntax:

```
.PARAM randpar1=GAUSS(nominal_val, rel_variation, sigma)  
.PARAM randpar2=AGAUSS(nominal_val, abs_variation, sigma)  
.PARAM randpar3=UNIF(nominal_val, rel_variation)  
.PARAM randpar4=AUNIF(nominal_val, abs_variation)
```

Here, the default for `sigma` is 3, and the equivalent mathematical description is:

```
randpar1=N(nominal_val, nominal_val*rel_variation/sigma)  
randpar2=N(nominal_val, abs_variation/sigma)  
randpar3=U(nominal_val*(1-rel_variation),  
nominal_val*(1+rel_variation))  
randpar4=U(nominal_val-abs_variation,  
nominal_val+abs_variation)
```

where $N(a, b) = a + b \cdot N(0, 1)$; $U(a, b) = a + (b - a) \cdot U(0, 1)$, $N(0, 1)$ and $U(0, 1)$ are the standard, normal, and uniform distributions, respectively.

Before each Monte Carlo trial, each IRV is assigned a different random number according to the corresponding probability density.

Chapter 24: Monte Carlo Analysis—Variation Block Flow

Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo

Example 1

```
.param randpar1=AGAUSS(0, 1, 1) par2='0.1*randpar1' par3=par2
.model SYNOP_NMOS nmos vth0='0.1 + randpar1'
+ M1 d g s b SYNOP_NMOS w='randpar1' l='randpar1'
+ M2 d g s b SYNOP_NMOS w='par2' l='par3'
```

In Example 1, before each Monte Carlo trial, the IRV `randpar1` gets four different numbers from the random number stream and uses them to create `par2`, `vth0`, and M1's width and M1's length, respectively. As `par3` is equal to `par2`, M1's `vth0` is equal to M2's `vth0`, M2's `w` is equal to M2's `l`, but M1's `w` is different from M1's `l`. Thus direct use of `randpar1` represents a different random number while indirect use through `par2` and `par3` leads to common random numbers. This was an early attempt to model local and global variations in instance parameters. However, only global variation was possible on model parameters.

Example 2

For traditional Monte Carlo, the `modmonte=1` option enables local variation on model parameters, so that different instances with the same model can get different random numbers in a single Monte Carlo trial. In the following example, before each Monte Carlo trial, the IRV `randpar1` will get five different random numbers from the random number stream and allocate them to `par2`, M1's `vth0` and M2's `vth0`, M1's `w` and M1's `l`. The only difference from Example 1 is that M1's `vth0` is different from M2's `vth0`.

```
.option modmonte=1
.param randpar1=Gauss(0, 1, 1) par2='0.1*randpar1' par3=par2
.model SYNOP_NMOS nmos vth0='0.1+ randpar1'
+ M1 d g s b SYNOP_NMOS w='randpar1' l='randpar1'
+ M2 d g s b SYNOP_NMOS w='par2' l='par3'
```

Input/Output with New Capability

There is no change to measurement and Monte Carlo commands or to the measurement output files and only minimal changes to the listing file with additional informational messages in cases of dual variability definitions. For example, the `*.mc0` files contain additional fields for random variables defined in the Gaussian style and new suffix keys IRV are introduced.

The content in a `*.mc0` file is similar to the Variation Block: the option settings are reported first, followed by the names of all requested models/devices/subcircuits, with their respective parameter names. Separators are used as follows:

- A single dot is a hierarchical separator between a subcircuit and an instance or device name.
- The special characters “:@:” separate model/device/subcircuit and parameter names.

Note:

- From the HSPICE release 2010.12-SP2 onward, the parameter name separator is changed to :@: from @.
- From the HSPICE release 2010.12-SP2 onward, the random number sample file suffix is changed to mc# from mc?#.

Syntax

- For normal parameters that use random variables directly, the syntax is:

```
RandomVariable:@:SubcircuitName.InstanceName:@:IRV
```

For example:

```
globw:@:globwidth:@:IRV , locwidth:@:x1.width:@:IRV
```

- For devices with models, the syntax is:

```
RandomVariable:@:ModelName:@:SubcircuitName.InstanceName  
:@:IRV
```

For example:

```
res_dev:@:x1.res_sub@x1.rab:@:IRV,  
res_dev@resistor:@:r2:@:IRV
```

- For model parameters, the syntax is:

```
RandomVariable:@:ModelName:@:SubcircuitName:@:IRV
```

For example:

```
par14@nch_mac.7:@:xmdut4:@:IRV
```

Example 1: Variation Duplicated in Traditional Format and Variation Block

```
.options ACCT OPTS LIST NOPAGE INGOLD=2 ALT999 PROBE POST=1 TNOM=25  
.options MODMONTE=1  
.model nch nmos LEVEL=53 VTH0='-0.4+vth0_nch*0.5'  
+ TOX='4E-9+4e-11*tox_nch' VERSION=3.2
```

Chapter 24: Monte Carlo Analysis—Variation Block Flow

Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo

```
.param vth0_nch = AGAUSS (0 , 1 , 1.0 )
.param tox_nch = AGAUSS (0 , 1 , 1.0 )
  m11  2 11  0  0  nch  W=1E-6  L=0.15E-6
  m12  2 11  0  0  nch  W=1E-6  L=0.15E-6
  m13  2 11  0  0  nch  W=1E-6  L=0.15E-6
  v01  2   0   1.5
  v02  11  0   0.0
.Variation
  Option Use_AGAUSS_Format=yes
  Option Sampling_Method=LHS
.Local_Variation
  nmos_nch vth0 = 0.5 //also define vth0 variation in VB
.End_Local_Variation
.End_Variation
.dc v02 0 2.0 0.1 sweep monte=2
.meas dc ids_11 find par('i(m11)*1E3') when v(11)=1.5
.meas dc ids_12 find par('i(m12)*1E3') when v(11)=1.5
.meas dc ids_13 find par('i(m13)*1E3') when v(11)=1.5
.end
```

Sample output in the *.lis file for this simulation is as follows:

```
*** monte carlo index =      3 ***

      MODEL PARAMETER MONTE CARLO DEFINITIONS
      0:m11
tox_nch
      nch              =  4.0722E-09
      0:m12
tox_nch
      nch              =  3.9584E-09
      0:m13
tox_nch
      nch              =  4.0462E-09

MONTE CARLO PARAMETER DEFINITIONS
vth0_nch
tox_nch

      ids_11=  1.6030E+00
      ids_12=  5.8869E-01
      ids_13=  8.1399E-01
```

Beginning with the 2011.09 release, the *.mc0 file includes the following characteristics using the \$ sign:

- The header section records lead with a \$ sign (which are read by HSPICE as comments).
- Title lines include an extra leading "\$", that is, \$\$.
- The external sampling option (as in a corner file) is recorded with a leading "\$", such as: `$.Option Sampling_Method = External_Block_Name = extern_data`
- Block definitions (as in a corner file) are recorded with a leading "\$" such as, `$.Data Extern_Data`
- Column names, random variable data (if requested), and the enddata record (as in the corner file) are also represented with a leading "\$".
- The *.mc file is written only in `.OPTION MEASFORM = 1 | 3 |` format for ease in exporting to Microsoft Excel. The default is 1. (See [.OPTION MEASFORM](#) in the *HSPICE Reference Manual: Commands and Control Options*.)

The *.mc0 file output for Example 1 shows the following results. Note that the independent random variable `vth0_nch` is missing due to the duplicated variation definition on `vth0.nch`.

```

$$$$SOURCE='HSPICE' VERSION='F-2011.09-BETA 32-BIT'
$ This file format is subject to change
$option ignore_global_variation = no
$option ignore_local_variation = no
$option ignore_interconnect_variation = no
$option ignore_spatial_variation = no
$option ignore_variation = no
$option sampling_method = lhs
$option normal_limit = 4.0000
$option replicates = 1
$option random_generator = MOA
$option stream = 1
$option output_sigma_val = 1.00
.TITLE '.options acct opts list nopage ingold=2 alt999 probe
post=1 tnom=25'
index      tox_nch:@:nch:@:~m11:@:IRV tox_nch:@:nch:@:~m12:@:IRV
           tox_nch:@:nch:@:~m13:@:IRV m11:@:vth0:@:ILN
           m11:@:nch:@:vth0:@:MLA m12:@:vth0:@:ILN
           m12:@:nch:@:vth0:@:MLA m13:@:vth0:@:ILN
           m13:@:nch:@:vth0:@:MLA

           status      alter#

```

Chapter 24: Monte Carlo Analysis—Variation Block Flow

Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo

1.0000	0.	0.	0.
	0.	0.	0.
	0.	0.	0.
	1.0	1.0000	
2.0000	-1.1333	0.2145	-0.6805
	1.3500	0.6750	-0.4423
	-0.2212	-0.5529	-0.2764
	1.0	1.0000	
3.0000	1.8038	-1.0394	1.1542
	-1.4199	-0.7100	1.2916
	0.6458	0.6412	0.3206
	1.0	1.0000	

Example 2: Subcircuit and Macro Models

The following example demonstrates the combined syntax used for subckt and macro models:

```
*four resistors
.param bias=1m
.param globw=AGAUSS(1u,0.1u,3)
.param globwidth=globw
.param locwidth= AGAUSS(0.1u,0.02u,3)
.param res_dev = AGAUSS(10,10,1)
.option modmonte=1 numdgt=8 ingold=2
i1 0 1 bias
i2 0 2 bias
i3 0 3 bias
i4 0 4 bias
i5 0 4 bias
i6 0 4 bias

X1 1 0 res1 width='globwidth+locwidth'
X2 2 0 res1 width='globwidth+locwidth'
X3 3 0 res1 width='globwidth+locwidth'
X4 4 0 res1 width='globwidth+locwidth'
r1 5 0 resistor w='3u+locwidth'
r2 6 0 resistor w='3u+locwidth'

.subckt res1 a b
rab a b res_sub w=width
.model res_sub R w=3u l='3u+res_dev*0.1u'
rsh='100+res_dev'
.ends res1
```


Chapter 24: Monte Carlo Analysis—Variation Block Flow
Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo

```
.model resistor R w=3u l='3u+res_dev*0.1u' rsh='100+res_dev'  
.Variation  
  Option Use_AGAUSS_Format=yes  
  
.End_Variation  
.op  
.dc bias 1m 1m 1m monte=4  
.print v(1) v(2) v(3) v(4)  
.measure dc v1 find v(1) at=1m  
.measure dc v2 find v(2) at=1m  
.measure dc v3 find v(3) at=1m  
.measure dc v4 find v(4) at=1m  
.option measdgt=6  
.end
```

The following sample is returned in the *.lis file for this statement:

```
*** monte carlo index =      2 ***  
      MODEL PARAMETER MONTE CARLO DEFINITIONS  
      0:r1  
res_dev  
  resistor      =  3.3475E-06  
res_dev  
  resistor      =  100.5  
      0:r2  
res_dev  
  resistor      =  3.9904E-06  
res_dev  
  resistor      =  133.9  
      1:rab  
res_dev  
  1:res_sub     =  4.9269E-06  
res_dev  
  1:res_sub     =  114.1  
      2:rab  
res_dev  
  2:res_sub     =  4.2027E-06  
res_dev  
  2:res_sub     =  124.7  
      3:rab  
res_dev  
  3:res_sub     =  3.4716E-06  
res_dev  
  3:res_sub     =  116.8  
      4:rab  
res_dev  
  4:res_sub     =  3.2135E-06  
res_dev  
  4:res_sub     =  99.87
```

Chapter 24: Monte Carlo Analysis—Variation Block Flow

Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo

MONTE CARLO PARAMETER DEFINITIONS

```
globw
globwidth      = 1.0285E-06
locwidth

      r1          = 3.0996E-06  r2          = 3.0984E-06
      1:width     = 1.1251E-06  2:width     = 1.1337E-06
      3:width     = 1.1295E-06  4:width     = 1.1364E-06
res_dev

x
      bias      voltage      voltage      voltage      voltage
                1            2            3            4
1.00000000e-03  5.1094738e-01  4.7630593e-01  3.6860296e-01
8.7529637e-01
y
v1= 5.109474e-01
v2= 4.763059e-01
v3= 3.686030e-01
v4= 8.752964e-01
```

The following sample output is returned for the subcircuit/macro file example:

```
*.ms0 file output
$ This file format is subject to change
$option ignore_global_variation = no
$option ignore_local_variation = no
$option ignore_interconnect_variation = no
$option ignore_spatial_variation = no
$option ignore_variation = no
$option sampling_method = srs
$option normal_limit = 4.0000000e+00
$option replicates = 1
$option random_generator = MOA
$option stream = 1
$option output_sigma_val = 1.00
.TITLE 'four resistors'
index      res_dev:@:resistor:@:r1:@:IRV
           res_dev:@:resistor:@:r1:@:IRV
           res_dev:@:resistor:@:r2:@:IRV
           res_dev:@:resistor:@:r2:@:IRV
           res_dev:@:x1.res_sub:@:x1.rab:@:IRV
           res_dev:@:x1.res_sub:@:x1.rab:@:IRV
           res_dev:@:x2.res_sub:@:x2.rab:@:IRV
           res_dev:@:x2.res_sub:@:x2.rab:@:IRV
           res_dev:@:x3.res_sub:@:x3.rab:@:IRV
           res_dev:@:x3.res_sub:@:x3.rab:@:IRV
```

Chapter 24: Monte Carlo Analysis—Variation Block Flow
 Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo

```
res_dev:@:x4.res_sub:@:x4.rab:@:IRV
res_dev:@:x4.res_sub:@:x4.rab:@:IRV
```

```
globw:@:globwidth:@:IRV
locwidth:@:r1:@:IRV
locwidth:@:r2:@:IRV
locwidth:@:x1.width:@:IRV
locwidth:@:x2.width:@:IRV
locwidth:@:x3.width:@:IRV
locwidth:@:x4.width:@:IRV
```

	status	alter#	
1.00000e+00	1.00000e+01	1.00000e+01	1.00000e+01
	1.00000e+01	1.00000e+01	1.00000e+01
	1.00000e+01	1.00000e+01	1.00000e+01
	1.00000e+01	1.00000e+01	1.00000e+01
	1.00000e-06	1.00000e-07	1.00000e-07
	1.00000e-07	1.00000e-07	1.00000e-07
	1.00000e-07		
	1.0	1.0000000e+00	
2.00000e+00	3.47526e+00	4.54263e-01	9.90365e+00
	3.38531e+01	1.92688e+01	1.40765e+01
	1.20268e+01	2.46670e+01	4.71569e+00
	1.67953e+01	2.13505e+00	-1.27138e-01
	1.02848e-06	9.96426e-08	9.84000e-08
	9.65996e-08	1.05238e-07	1.01065e-07
	1.07936e-07		
	1.0	1.0000000e+00	
3.00000e+00	2.04627e+01	1.45870e+01	1.16248e+01
	8.74212e+00	7.04076e+00	1.77743e+01
	-2.05886e+01	2.42788e+00	2.22087e+01
	2.74557e+01	1.93586e+01	1.76029e+01
	1.03292e-06	1.08333e-07	1.01669e-07
	1.06247e-07	1.04484e-07	9.72494e-08
	1.08387e-07		
	1.0	1.0000000e+00	
4.00000e+00	6.12805e+00	1.20906e+01	8.52608e+00
	4.93741e+00	1.60211e+01	1.09779e+01
	3.89823e+00	1.69714e+01	4.64658e+00
	-9.45094e-01	1.06839e+01	1.47630e+01
	9.93244e-07	1.02410e-07	9.43055e-08
	1.05275e-07	1.07705e-07	8.09711e-08
	1.09258e-07		
	1.0	1.0000000e+00	

Application Considerations

Due to the combinations of variation specified in the variation block and the value of `Normal_Limit`, variations that are applied may be too large. This can make some circuits display abnormal behavior and produce unrealistic results for certain samples. This, in turn, can distort the summary statistics reported by HSPICE at the end of the Monte Carlo simulation.

Troubleshooting Monte Carlo-VB Issues

Troubleshoot this known issues as described in the following section:

- [Independent Random Variable Assignments](#)

Independent Random Variable Assignments

Users may come across differences in the results following multiple Monte Carlo simulations without changing the `SEED` setting if they have made small changes to the netlist without actually changing the circuit. The following describes the way HSPICE assigns random values to the independent random variables to explain how those differences can arise.

Let two random variables be defined in case 1:

```
.Variation
  .Spatial_Variation
    Parameter a = N( )
    Parameter b = N( )
    Parameter Slope = 'a/50u'
    Parameter Pi = 3.14159265
    Parameter Angle = 'Pi*2*b'
R rmodel rsh=Perturb('Slope*sqrt(Get_E(x)* Get_E(x)+ Get_E(y)*
Get_E(y)) \\
*cos(Angle-atan(Get_E(y)/Get_E(x)) - (Get_E(x)<0?Pi:0))')
  .End_Spatial_Variation
.End_Variation
```

Random variables `a` and `b` are used to calculate variations in sheet resistivity as a function of a resistor's coordinates.

Let case 2 have four random variables defined:

```
.Variation
  .Spatial_Variation
    Parameter a = N( )
    Parameter b = N( )
    Parameter c = N( )
    Parameter d = N( )
    Parameter Slope = 'a/50u'
    Parameter Pi = 3.14159265
    Parameter Angle = 'Pi*2*b'
R rmodel rsh=Perturb('Slope*sqrt(Get_E(x) * Get_E(x) + Get_E(y) *
Get_E(y))\ \
    *cos(Angle-atan(Get_E(y)/Get_E(x)) - (Get_E(x)<0?Pi:0))')
  .End_Spatial_Variation
.End_Variation
```

Random variables a and b are used in the Variation Block; c and d are not used.

During the Monte Carlo sweep, a pseudo random number generator creates an array of random values (Random1, Random2, . . . , RandomN) and assigns them to each an independent random variable.

In case 1, the random number assignment is:

```
Monte=1 -- a=Random1 b=Random2
Monte=2 -- a=Random3 b=Random4
. . .
Monte=N -- a=Random2N-1 b=Random2N
```

In case 2, the random number assignment is:

```
Monte=1 -- a=Random1 b=Random2 c=Random3 d=Random4
Monte=2 -- a=Random5 b=Random6 c=Random7 d=Random8
. . .
Monte=N -- a=Random4N-3 b= Random4N-2 c=Random4N-1 d=Random4N
```

Here, although Random1 through RandomN are the same in both cases, the sequence of assignment to independent random variables differs. Hence, the individual samples of a (or b) differ between the two simulations. As a consequence, at low sample numbers the difference in the standard deviation (sigma) of the distributions of a (or b) might be quite large. For higher sample numbers, the differences get smaller, according to the general convergence

rate of Monte Carlo results of $\frac{1}{\sqrt{n}}$ where n is the number of samples.

Chapter 24: Monte Carlo Analysis—Variation Block Flow

Troubleshooting Monte Carlo-VB Issues

Because a pseudo random number generator is used in HSPICE, repeated simulations generate the same set of statistical results for a given set of independent random variables. The user can change the random number sequences at each run by defining in the Variation Block: `Option Stream = val` where *val* is an integer between one and twenty.

In the traditional Monte Carlo style, this is the similar setting to:

```
.option seed=val
```

When using Monte Carlo simulation, you should keep in mind that there is always uncertainty associated with this method in the relationship of one sample to the overall population.

References

- [1] V. Czitrom: One-Factor-at-a-Time Versus Designed Experiments. *The American Statistician*, pp.126-131, May 1999.
- [2] M.D. McKay, R.J. Beckman, and W.J. Conover: A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, pp. 239-245, 1979.
- [3] M. Stein: Large Sample Properties of Simulations Using Latin Hypercube Sampling. *Technometrics*, pp. 143-151, 1987.
- [4] I.M. Sobol: On the Systematic Search in a Hypercube. *SIAM J. Numerical Analysis*, pp. 790-793, 1979.
- [5] H. Niederreiter: *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.
- [6] A. Singhee and R.A. Rutenbar: Why Quasi-Monte Carlo is Better than Monte Carlo or Latin Hypercube Sampling for Statistical Circuit Analysis. *IEEE Trans. on Computer Aided Design of Circuits and Systems*, pp. 1763-1776, November, 2010.

Chapter 24: Monte Carlo Analysis—Variation Block Flow
References

Mismatch Analyses

Describes the use of DC and AC mismatch analyses in HSPICE.

DCMatch and ACMatch analyses are efficient techniques for computing the effects of variations on a circuit's DC or AC response. The variation definitions are taken from the Variation Block or from the traditional format (commonly known as “AGAUSS” format). Both methods are small signal analyses, similar to noise analysis. Unlike the traditional Monte Carlo analysis, these methods do not rely on sampling, and are therefore significantly faster. The Monte Carlo results converge to those from DCMatch or ACMatch analysis for a large number of samples, provided that the circuit characteristics are close to linear in the parameter perturbations. Matching analyses are supported with the Variation Block format and, starting with the 2010.12-SP2 release, the AGAUSS style.

DCMatch and ACMatch analyses are affected by the control options specified in the Variation Block (see [Control Options and Syntax](#) in the chapter [Variability Analysis Using the Variation Block](#)).

HSPICE ships many examples for your use. See [Variability Examples](#) for paths to DC and AC mismatch demo files.

These topics are covered in the following sections:

- [Mismatch](#)
- [DCMatch Analysis](#)
- [ACMatch Analysis](#)
- [Application Considerations](#)
- [Mismatch Compared to Monte Carlo Analysis](#)

Mismatch

Variations in materials and processing steps are the source of differences in the characteristics of identically designed devices in close proximity on the same integrated circuit. These are random time-independent variations by nature and are collectively called *mismatch*.

Mismatch is one of the key limiting factors in analog signal processing. It affects more and more circuit types as device dimensions and signal swings are reduced. Mismatch is a function of the geometry of the devices involved, their spatial relationship (distance and orientation), and their environment.

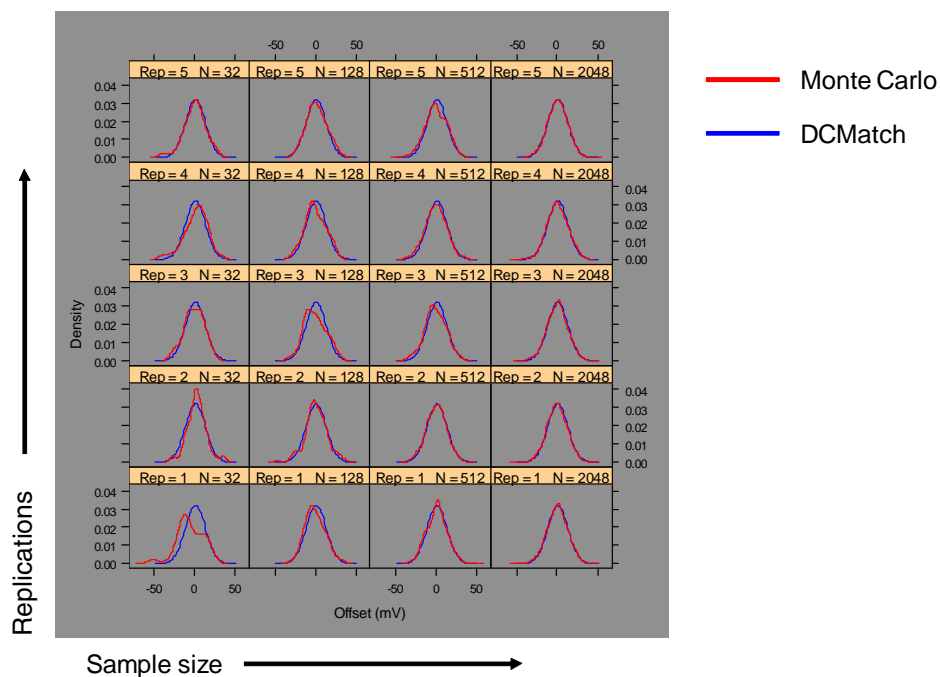


Figure 149 DCMatch and Monte Carlo Comparative Results

This chapter discusses the following mismatch analyses:

- [DCMatch Analysis](#)
- [ACMatch Analysis](#)

DCMatch Analysis

To observe the effects of variation on the DC response of a circuit, you can use a method called DC mismatch (DCMatch) analysis.

In DCMatch analysis, the combined effects of variations of all devices on a specified node voltage or branch current are determined. The primary purpose is to consider the effects of Local variations (that is, for devices in close proximity). DCMatch analysis also allows for identifying groups of matched devices (that is, devices that should be implemented on the layout according to special rules). A secondary set of results is calculated from the influences of Global and Spatial Variations, which is useful for investigating whether their effects on circuit response are much smaller than the effects of Local variations, when optimizing a design.

DCMatch analysis is based on the following dependencies and assumptions:

- Variations in device characteristics are modeled through variations in the underlying model parameters.
- Effects on a circuit's DC solution are small and can be modeled as a linear combination of the variations in the random variables.

In HSPICE, the variations in model parameters are defined in the Variation Block (see [Chapter 23, Variability Analysis Using the Variation Block](#)). Those definitions are used to calculate the variation in DC response. DCMatch analysis runs either from a default operating point or for each value of the independent variable in a DC sweep. The default output is in the form of tables containing the sorted contributions of the relevant devices to the total variation, as well as information on matched devices. In the current implementation, a heuristic algorithm makes a best guess effort to identify matched devices. This means that the results are suggestions only. In addition to the table, the total variation and contributions of selected devices can be output using `.PROBE` and `.MEASURE` commands.

Input Syntax

```
.DCMatch OUTVAR [THRESHOLD=T] [FILE=string] [INTERVAL=Int]
```

Parameter	Description
OUTVAR	One or more node voltages, voltage differences for a node pair, currents through an independent voltage source or a resistor.
THRESHOLD	Report devices with a relative variance contribution above Threshold in the summary table. <ul style="list-style-type: none"> ▪ T=0: reports results for all devices ▪ T<0: suppresses table output; however, individual results are still available through .PROBE or .MEASURE statements. The upper limit for T is 1, but at least 10 devices are reported, or all if there are less than 10. Default value is 0.01.
FILE	Valid file name for the output tables. Default is <code>basename.dm#</code> where “#” is the usual sequence number for HSPICE output files.
INTERVAL	Applies only if a DC sweep is specified. <i>Int</i> is a positive integer. A summary is printed at the first sweep point, then for each subsequent increment of <i>Int</i> , and then, if not already printed, at the final sweep point.

Note: If more than one DCMatch analysis is specified per simulation, only the last statement is used.

Example 1

In this example, HSPICE reports DCMatch variations on the voltage of node 9, the voltage difference between nodes 4 and 2, on the current through the source VCC, and the current through resistor `x1.r1`.

```
.DCMatch V(9) V(4,2) I(VCC) I(x1.r1)
```

Example 2

In this example, the variable `XVal` is being swept in the DC command from 1k to 9k in increments of 1k. DCMatch variations are calculated for the voltage on node `out`. Tables with DCMatch results are generated for the set `XVal={1K, 4K, 7K, 9K}`.

```
.DC XVal Start=1K Stop=9K Step=1K
.DCMatch V(out) Interval=3
```

DCMatch Table Output

For each output variable and sweep point, HSPICE generates a result record that includes setup information, total variations, and a table with the sorted contributions of the relevant devices. The individual entries are:

- Sweep or operating points for which the table is generated
- Name of the output variable
- DC value of this output variable
- Values used for DCMatch options
- Output sigma due to combined Global, Local, and Spatial variations

$$\sqrt{\sigma_{\text{global}}^2 + \sigma_{\text{local}}^2 + \sigma_{\text{spatial}}^2}$$

- Results for Global variations (similar to the specifics of Local Variation)
- Results for Local variations:
 - Number of devices that had no local variability specified
 - Output sigma due to Local variations
 - Number of devices with local variance contributions below the threshold value and not included in the table
 - Table with sorted device contributions

Contribution sigma (in volts or amperes). Values below 100 nV or 1 pA are rounded to zero to avoid reporting numerical noise.

- Contribution variance for i th parameter (in percent)

$$\frac{\sigma(i)^2}{\sum_1^n \sigma(k)^2} \times 100$$

The parameter “Threshold” applies to this column.

- Cumulative variance through i th parameter (in percent)

Chapter 25: Mismatch Analyses
DCMatch Analysis

$$\frac{\sum_1^i \alpha(k)^2}{\sum_1^n \alpha(k)^2} \times 100$$

- Results for Spatial variations are similar to the previous item, Local Variation.

Example: Simple Op-Amp and DCMatch Output Table Described

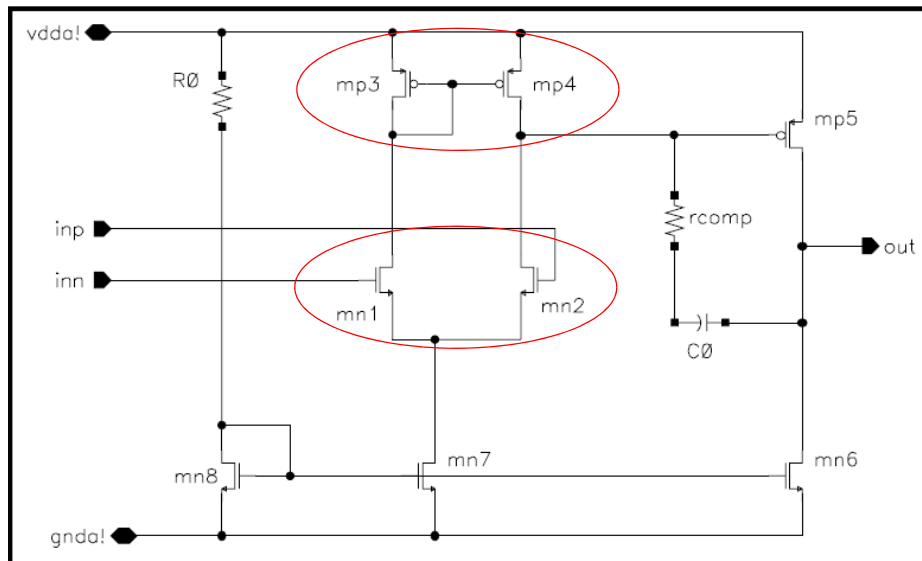


Figure 150 Simple Op-Amp

```

Output sigma due to global and local variations = 619.62uV
DCMATCH GLOBAL VARIATION
Output sigma due to global variations = 289.66uV
-----
Contribution   Contribution   Cumulative   Independent
Sigma (V)     Variance (%)   Variance (%) Variable
227.94u       61.92          61.92       snps20p:@:u0
139.48u       23.19          85.11       snps20p:@:vth0
109.93u       14.40          99.51       snps20n:@:u0
20.19u        485.62m       100.00      snps20n:@:vth0

DCMATCH LOCAL VARIATION
Output sigma due to local variations = 547.74uV
-----
Contribution   Contribution   Cumulative   Matched   Device
Sigma (V)     Variance (%)   Variance (%) pair       Name
297.91u       29.46          29.46        1         mn1
296.38u       29.16          58.61        1         mn2
252.37u       21.14          79.75        2         mp3
246.91u       20.23          99.99        2         mp4

```

Figure 151 Sample Output Table

The output table in [Figure 151](#) shows the major sections of a table created by DCmatch analysis in the following order:

- The DCmatch results from an operating point, for the node called “out”, which has an operating point voltage of 1.25 V.
- The standard deviation (sigma) of the variation on the specified output, due to combined global and local variations.
- A section with detailed results for the global variations. First is the output sigma due to global variations.
- The contribution of the different variables and device parameters, as specified in the Variation Block.

Note: The sum of squares of the contribution sigmas adds up to the square of the total output sigma.

- The results for the contribution and cumulative variance are reported as percent of the total variance (which is the square of the total output sigma). In the example shown, the variation in Vth0 for the model snps20p is the largest contributor to the variation of the output.

Chapter 25: Mismatch Analyses

DCMatch Analysis

- The detailed results for local variations, starting with the output sigma due to local variations.
- The contribution of the different devices. A column with matched pair information indicates that mn1 and mn2 are matched, as well as mp3 and mp4. The layout of these devices should be checked for conformity to established matching rules.

The table also includes a suggestion on matched devices that should be verified independently. Devices with the same number in the column “Matched pair” are likely to be matched. Their layout should be reviewed for conformity to established matching rules.

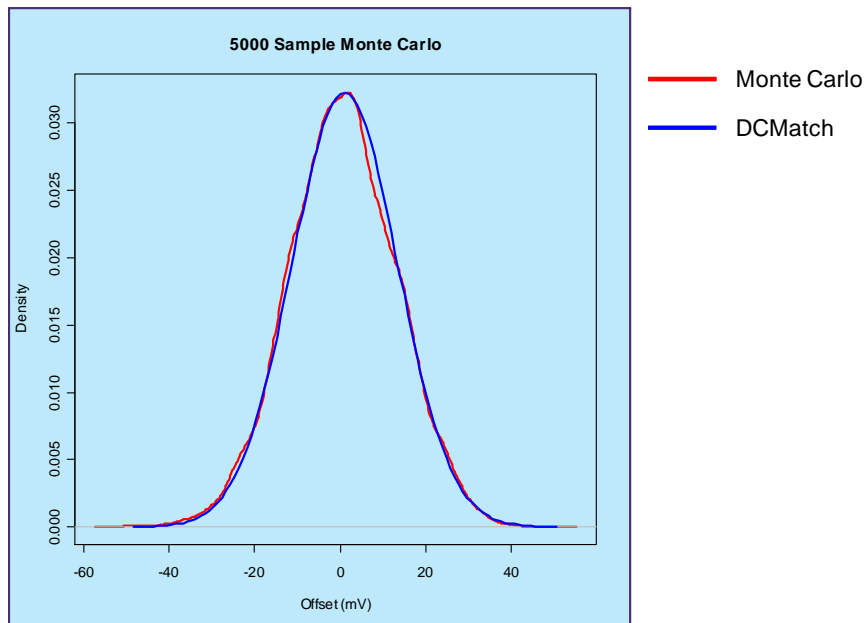


Figure 152 DCMatch and Monte Carlo

HSPICE Ignores Limit Distributions in DCMatch

HSPICE calculates the output sigma using the following equation:

$$\sigma(f) = \sqrt{\sigma(f)^2 \left(\frac{\partial f}{\partial x}\right)^2 + \sigma(f)^2 \left(\frac{\partial f}{\partial y}\right)^2}$$

In addition to this equation, DCMatch assumes that the metric is differentiable because it is calculus-based. The `Limit` function is often used to drive conditional statements. This makes the metric non-differentiable.

The current mismatch implementation ignores limit distributions in DCmatch.

The advantages of DCMatch are (a) it is fast, and (b) results are expected to be close to what one would get with Monte Carlo. In the case of the limit distribution, condition (b) is likely to be violated. To avoid listing improper results for DCMatch, HSPICE issues a warning message and ignores DC/ACMatch if variations have limit distributions.

Output Using .PROBE and .MEASURE Commands

Depending on the output variable specified on the `.DCMatch` command, results produced by DCMatch analysis can be saved by using `.PROBE` and `.MEASURE` commands (see syntax and examples that follow). If multiple output variables are specified, a result is produced for the last variable only. A DC sweep needs to be specified to produce these kinds of outputs; a single point sweep is sufficient.

The keywords available for saving specific results from DCMatch analysis are:

Table 79 Keyword descriptions from DCMatch Analysis

Keyword	Description
<code>DCM_Total</code>	Output sigma due to Global and Local variations.
<code>DCM_Global</code>	Output sigma due to Global variations.
<code>DCM_Global(par)</code>	Contribution of parameter “par” to output sigma due to Global variations. Here, 'par' can be an independent variable or a model parameter.
<code>DCM_Local</code>	Output sigma due to Local variations.
<code>DCM_Local(dev)</code>	Contribution of device “dev” to output sigma due to Local variations.
<code>DCM_Spatial</code>	Output sigma due to Spatial Variations.
<code>DCM_Spatial(var)</code>	Contribution of independent variable “var” to output sigma due to Spatial Variations.

Syntax for .PROBE Command for DCMatch

A `.PROBE` statement in conjunction with `.OPTION POST` creates a data file with waveforms that can be displayed in WaveView.

```
.PROBE DC DCM_Total  
.PROBE DC DCM_Global  
.PROBE DC DCM_Local  
.PROBE DC DCM_Global (VariableName)  
.PROBE DC DCM_Global (ModelType, ModelName, ParameterName)  
.PROBE DC DCM_Local (InstanceName)  
.PROBE DC DCM_Spatial  
.PROBE DC DCM_Spatial (VariableName)
```

This type of output is useful for plotting the effects of mismatch as a function of bias current, temperature, or a circuit parameter.

Examples

In the first example, the contribution of the variations on `vth0` (threshold) of the `nmos` devices with model `SNPS20N` is saved. In the second example, the contribution of device `mn1` in subcircuit `X8` is saved.

```
.Probe DCM_Global (nmos, SNPS20N, vth0)  
.Probe DCM_Local (X8.mn1)
```

Syntax for .MEASURE Command

With `.MEASURE` statements, HSPICE performs measurements on the simulation results and saves them in a file with an `*.ms#` extension.

```
.MEAS DC res1 max DCM_Total  
.MEAS DC res2 max DCM_Global  
.MEAS DC res3 max DCM_Local  
.MEAS DC res4 max DCM_Global (VariableName)  
.MEAS DC res5 max DCM_Global (ModelType, ModelName, ParameterName)  
.MEAS DC res6 max DCM_Local (InstanceName)  
.MEAS DC res7 find DCM_Local at=SweepValue  
.MEAS DC res8 find DCM_Local (InstanceName) at=SweepValue  
.MEAS DC res9 max DCM_Spatial  
.MEAS DC res10 find DCM_Spatial (VariableName) at=SweepValue
```

The keywords `DCM_total`, `DCM_global`, and `DCM_local` also support use of `AGAUSS` type macro models. For example:

```
.MEAS DC dcm_mn2 find DCM_local (xi82.xmn2.main) at=2
```

Example

In this example, the result `systoffset` reports the systematic offset of the amplifier, the result `matchoffset` reports the variation due to local mismatch, and the result `maxoffset` reports the maximum (3-sigma) offset of the amplifier.

```
.MEAS DC systoffset avg V(inp,inn)
.MEAS DC matchoffset avg DCM_Local
.MEAS DC maxoffset param='abs(systoffset)+3.0*matchoffset'
```

DCMatch Example Netlist

An example netlist for running DCMatch analysis using a classic 8-transistor CMOS operational amplifier is available in the HSPICE demo directory as `$installdir/demo/hspice/variability/opampdcm.sp`.

In this netlist, device sizes are set up as a function of a parameter `k`, which allows for investigating the effects of the Global and Local Variations as a function of device size. The following lines relate to DCMatch analysis:

Chapter 25: Mismatch Analyses

DCMatch Analysis

```
...
.param k=2
...
mn1 net031 inn net044 nmosbulk snps20N L='k*0.5u' W='k*3.5u' M=4
mn2 net18 inp net044 nmosbulk snps20N L='k*0.5u' W='k*3.5u' M=4
mp3 net031 net031 vdda pmosbulk snps20P L='k*0.5u' W='k*4.5u' M=4
mp4 net18 net031 vdda pmosbulk snps20P L='k*0.5u' W='k*4.5u' M=4
...
.Variation
  .Global_Variation
    Nmos snps20N vth0=0.07 u0=10 %
    Pmos snps20P vth0=0.08 u0=8 %
  .End_Global_Variation
  .Local_Variation
    Nmos snps20N vth0='1.234e-9/
sqrt(Get_E(W)*Get_E(L)*Get_E(M))'
    + u0='2.345e-6/sqrt(Get_E(W)*Get_E(L)*Get_E(M))' %
    Pmos snps20P vth0='1.234e-9/
sqrt(Get_E(W)*Get_E(L)*Get_E(M))'
    + u0='2.345e-6/sqrt(Get_E(W)*Get_E(L)*Get_E(M))' %
  .Element_Variation
    R r=10 %
  .End_Element_Variation
  .End_Local_Variation
.End_Variation
...
.DCMatch v(out)
.dc k start=1 stop=4 step=0.5
...
.meas DC systoffset find V(in_pos,in_neg) at=2
.meas DC dcmoffset find DCM_Local at=2
.meas DC maxoffset param='abs(systoffset)+3.0*dcmoffset'
.meas DC dcm_mn2 find DCM_Local(xi82.mn2) at=2
.meas DC gloffset find DCM_Global at=2
.option post
...
```

The DCMatch analysis produces four types of output from this netlist:

- Table from operating point with $k=2$ in the output listing
- Table from DC sweep for $k=1$ to 4 in file `opampdcm.dm0`
- Waveform for output variation as a function of k in file `opampdcm.sw0`
- In file `opampdcm.sw0` for $k=2$:
 - Values for systematic offset
 - Output sigma due to Local Variation

- 3-sigma amplifier offset
- Contribution of device mn2 to output sigma due to Local Variation
- Output sigma due to Global Variation

Note: When a netlist contains both `.OP` and `.DCMATCH` commands, the output is exported to the `*.dm#` file instead of the `*.lis` file beginning with HSPICE release D-2010.03-SP2.

ACMatch Analysis

In ACMatch analysis, the combined effects of variations of device characteristics on the frequency response of a circuit are determined. The variation definitions are taken from the Variation Block.

The main application for ACMatch analysis is in the simulation of circuits which are sensitive to parasitics or require matching of parasitics, for characteristics such as delays and power supply rejection.

ACMatch analysis takes the changes in frequency response due to variations in DC parameters (which affect operating point and low frequency response, as well as bias-dependent capacitors) and due to variations in AC parameters. Note that variation on the stimuli (voltage and current sources) can be specified on the DC and AC parameters, and both types are considered in the ACMatch analysis.

ACMatch analysis is similar to DCMatch analysis in that:

- It is efficient compared to Monte Carlo analysis because there is no sampling involved.
- Variations in component characteristics are modeled through variations in the underlying model parameters.
- Effects on a circuit's DC solution are small, and can be modeled as a linear combination of the variation in independent random variables. This is relevant for ACMatch analysis because the changes in the DC solution affect the circuit's AC characteristics.

ACMatch analysis is specified with an AC analysis, which defines the frequencies for which the circuit is analyzed; this can be at single or multiple sweep points. At least one measure or other output statement is required for this AC analysis, and subsequently ACMatch analysis, to run. The primary

output of ACMatch analysis is a table with the sorted parameter and device contributions.

Parasitic Capacitor Sensitivity

ACMatch allows for calculation of virtual parasitic capacitor sensitivities whose nominal values are “zero” in the original design and that are not specified in the netlist. Such an analysis is useful for high precision (differential) analog circuits and switched capacitor filters, because they are quite sensitive to layout parasitics, but their values are not known at the pre-layout stage.

In the following scenario, the parasitic capacity sensitivity feature can be useful.

The design of a network usually begins with a study in which all the parasitic elements are neglected. Later, if needed, the network is re-simulated with the parasitic elements and its behavior compared with the original design. Such analysis is useful for high precision (differential) analog circuits and switched capacitor filters, because they are quite sensitive to layout parasitics, but their values are not known at the pre-layout stage. Just as DC and ACMatch are useful for identifying critical devices that can then be re-sized or their layout constructed carefully, designers can evaluate similar feedback about parasitic components.

The Virtual Capacitance table calculates such parasitic capacitor sensitivities whose nominal values are 'zero' in the original design.

Input Syntax

```
.ACMatch OUTVAR [THRESHOLD=T] [FILE=string] [INTERVAL=Int]  
+ [Virtual_Sensitivity=Yes|No] [Sens_threshold=x]  
+ [Sens_Node=(nodei_name,nodej_name), ..., (nodem_name,noden_name)]
```

Parameter	Description
OUTVAR	Output Variable can be one or several output voltages, voltage differences, branch current through an independent voltage source, or currents through a resistor, a capacitor, or an inductor. An identifier of the AC quantity of interest is followed by the voltage/current specifier: <ul style="list-style-type: none"> ▪ M magnitude ▪ P phase ▪ R real part ▪ I imaginary part
THRESHOLD	Report devices with a relative variance contribution above Threshold in the summary table. <ul style="list-style-type: none"> ▪ T=0: reports results for all devices ▪ T<0: suppresses table output; however, individual results are still available through .PROBE or .MEASURE statements. The upper limit for T is 1, but at least 10 devices are reported, or all if there are less than 10. Default: 0.01.
FILE	Valid file name for the output tables. Default is <i>basename.am#</i> where “#” is the usual sequence number for HSPICE output files.
INTERVAL	Relates to the associated AC sweep. <i>Int</i> is a positive integer. A summary is printed at the first sweep point, then for each subsequent increment of <i>Int</i> , and then, if not already printed, at the final sweep point.
Virtual_Sensitivity	Invokes ACmatch computation and output of virtual sensitivity. Default: Yes
Sens_Threshold= <i>x</i>	Only nodes with sensitivity above <i>x</i> are reported. At least 10 sensitivities (or all) are displayed. This avoids generation of null output if you specify too large a value for <i>x</i> . Default: 1e-6
Sens_Node	Output all sensitivities associated with the requested nodes. The node name should appear in pairs. (See examples below.)

If more than one ACMatch analysis is specified per simulation, only the last statement is executed.

Examples

```
.ACMatch VM(out) VP(out) IM(x1.r1) IP(x1.r1) IM(c1) IP(c1)  
.AC dec 10 1k 10Meg interval=10
```

When using the virtual sensitivity option `Sens_Node`, multiple name pairs are supported with a comma between node names, and between node name pairs.

```
.ACmatch v(out) virtual_sens=yes  
+ sens_node= (out, xi82.net044),  
+ (0,out), (xi82.net044,xi82.net031) sens_threshold=1e-6
```

ACMatch Table Output

For each output variable and sweep point, HSPICE generates a result record in a file with default extension `.am#`. This file includes setup information, a main result with the total variations, and two tables. One table includes the sorted contributions of the relevant devices and parameters, and the other table includes virtual capacitance data.

The individual entries for the mismatch table include:

- Frequency sweep value
- Name of the output variable
- AC magnitude of this output variable
- ACMatch option values
- Number of devices which had no variability specified
- Output sigma values due to combined Global and Local Variations
- Result for Global Variations
- Contributions of parameters to Global Variations
- Results for Local Variations
- Contributions of devices to Local Variations

The entries in the different columns correspond to those described in the section on [DCMatch Table Output on page 839](#).

To avoid printing unreliable results due to precision issues, phase output is not available in the table if the associated magnitude of the same variation type is

less than 1 uV for voltage or 1 nA for current output. A warning is printed instead.

The entries for the virtual capacitance table include:

- The output value is printed with Re and Im components regardless of the request on the ACMatch command.
- The section header is a virtual capacitance table (per femto farad).
- The sensitivities are multiplied by 1e-15. This is the “natural” unit for measuring typical parasitic capacitors in nanometer technologies.

Example 1 (Parasitic Capacitance Sensitivity)

```

1 ***** HSPICE -- B-2008.09-TST 32-BIT (May 23 2008) sunos *****

***** AC mismatch analysis          tnom= 25.000 temp= 25.000
frequency = 1.00000D+03
=====
output = v(out) node voltage =      1.82kV threshold = 1.000E-02
perturbation = 2.00      interval = 1 virtual_sensitivity = yes
sensitivity_threshold = 1.000E-06
Output 1-sigma due to total variations = 148.43 V
ACMATCH LOCAL VARIATION
    15 Devices had no Local Variability specified
Output 1-sigma due to local variations = 148.43 V
    1 Devices with Local Contribution Variance larger than Threshold
-----
Contribution          Contribution          Cumulative          Device
1Sigma (V)            Variance (%)          Variance (%)        Name
148.43                100.00                100.00              xi82.ccomp
745.32m                2.52m                 100.00              c1
4.82m                  105.39n                100.00              c0
-----
Virtual capacitance table (per femto farad)
sens(real)            sens(imaginary)      (nodei, nodej)
196.06u                208.38u              (out, xi82.net058)
196.04u                208.40u              (out, xi82.net18)
-191.73u               -203.47u             (out, xi82.net031)
7.29u                  8.21u                (xi82.net031, xi82.net18)
7.31u                  8.19u                (xi82.net031, xi82.net058)
3.63u                  4.19u                (in_pos, xi82.net18)
3.64u                  4.18u                (in_pos, xi82.net058)
3.69u                  4.05u                (xi82.net18, xi82.net0148)
3.70u                  4.04u                (xi82.net0148, xi82.net058)
3.69u                  4.05u                (gnda, xi82.net18)
3.69u                  4.05u                (vdda, xi82.net18)

```

Chapter 25: Mismatch Analyses
ACMatch Analysis

```

3.69u          4.05u          (xi82.net18, 0)
3.69u          4.05u          (in_neg, xi82.net18)
3.70u          4.04u          (gnda, xi82.net058)
3.70u          4.04u          (vdda, xi82.net058)
3.70u          4.04u          (xi82.net058, 0)
3.70u          4.04u          (in_neg, xi82.net058)
3.64u          4.06u          (xi82.net044, xi82.net18)
3.65u          4.06u          (xi82.net044, xi82.net058)
1.38u          1.52u          (out, xi82.net044)
171.31n       202.01n       (out, 0)
-12.88n       17.84n        (xi82.net031, xi82.net044)
*** ACmatch end for this simulation***

```

Note that the sensitivities of the last two rows are smaller than the threshold. However, they are still printed because these nodes are selected with the option `sens_node = (0,out) (xi82.net044,xi82.net031)`.

Example 2 (Sensitivity Not Calculated)

```

frequency = 1.00000D+06
=====
output = v(out) node voltage = 1.87 V threshold = 1.000E-02
perturbation = 2.00 interval = 1 virtual sensitivity = no

```

Output 1-sigma due to Global and local Variations = 48.68mV

ACMatch GLOBAL VARIATION

10 Devices had no Global Variability specified

Output 1-sigma due to Global Variations = 46.97mV

```

-----
Contribution      Contribution      Cumulative      Independent
1Sigma (V)        Variance (%)     Variance (%)    Variable
38.89m            68.57           68.57           snps20n:@:vth0
15.78m            11.28           79.85           snps20p:@:vth0
15.08m            10.31           90.16           snps20n:@:tox
14.56m            9.61            99.77           snps20n:@:u0
1.80m             146.80m         99.91           snps20p:@:u0
1.38m             86.49m          100.00          snps20p:@:tox

```

ACMatch LOCAL VARIATION

6 Devices had no Local Variability specified

Output 1-sigma due to Local Variations = 12.79mV

7 Devices with Local Contribution Variance larger than Threshold

```

-----
Contribution      Contribution      Cumulative      Device
1Sigma (V)        Variance (%)     Variance (%)    Name
7.43m             33.77           33.77           xi82.mn7
6.26m             23.97           57.73           xi82.mp4
6.20m             23.53           81.26           xi82.mp3

```

4.87m	14.49	95.75	xi82.mn8
1.53m	1.43	97.18	xi82.mn2
1.49m	1.36	98.54	xi82.mn1
1.40m	1.20	99.74	r1
563.27u	193.90m	99.93	xi82.mp5
239.10u	34.94m	99.97	xi82.rcomp
184.24u	20.75m	99.99	xi82.mn6

Output from .PROBE and .MEASURE Commands for ACMatch

The syntax of .MEASURE and .PROBE commands for ACMatch analysis is similar to the syntax for DCMatch analysis.

Syntax for .PROBE Command

A .PROBE statement in conjunction with .OPTION POST creates a data file with waveforms that can be displayed in WaveView.

```
.PROBE AC ACM_Total
.PROBE AC ACM_Global
.PROBE AC ACM_Local
.PROBE AC ACM_Global(VariableName)
.PROBE AC ACM_Global(ModelType,ModelName,ParameterName)
.PROBE AC ACM_Local(InstanceName)
```

Syntax for .MEASURE Command

With .MEASURE statements, HSPICE performs measurements on the simulation results and saves them in a file with a *.ma# extension.

```
.MEAS AC res1 max ACM_Total
.MEAS AC res2 max ACM_Global
.MEAS AC res3 max ACM_Local
.MEAS AC res5 max ACM_Global(VariableName)
.MEAS AC res6 max ACM_Global(ModelType,ModelName,ParameterName)
.MEAS AC res7 max ACM_Local(InstanceName)
.MEAS AC res8 find ACM_Local at=SweepValue
.MEAS AC res9 find ACM_Local(InstanceName) at=SweepValue
```

Example

An example netlist for running ACMatch analysis using a classic 7-transistor CMOS operational amplifier is available in the HSPICE demo directory as \$installdir/demo/hspice/variability/opampacm.sp. The following lines relate to ACMatch analysis:

```
.Variation
.Global_Variation
    Nmos snps20N vth0=0.07 u0=10 % tox=3 %
    Pmos snps20P vth0=0.08 u0=8 % tox=3 %
```

Chapter 25: Mismatch Analyses

ACMatch Analysis

```

.End_Global_Variation
.Local_Variation
  Nmos snps20N vth0='1.234e-9 sqrt(Get_E(W)*Get_E(L)*Get_E(M))'
    + u0='2.345e-6/sqrt(Get_E(W)*Get_E(L)*Get_E(M))' %
    + tox='3.456e-6/sqrt(Get_E(W)*Get_E(L)*Get_E(M))' %
  Pmos snps20P vth0='1.234e-9/
    sqrt(Get_E(W)*Get_E(L)*Get_E(M))'
    + u0='2.345e-6/sqrt(Get_E(W)*Get_E(L)*Get_E(M))' %
    + tox='3.456e-6/sqrt(Get_E(W)*Get_E(L)*Get_E(M))' %
.Element_Variation
  R r=10 %
.End_Element_Variation
.End_Local_Variation
.End_Variation
.ACMatch v(out)
.ac dec 1 1k 10Meg
.meas ac res1 find acm_local at=1k

```

In this example, ACMatch analysis runs at 1 kHz, 10 kHz, 100 kHz, 1 MHz, and 10 MHz. After simulation, the results in `opampacm.am0` show the contributions of devices and parameters, and their different relative importance for the different frequencies. The measurement result is printed in `opampacm.ma0`.

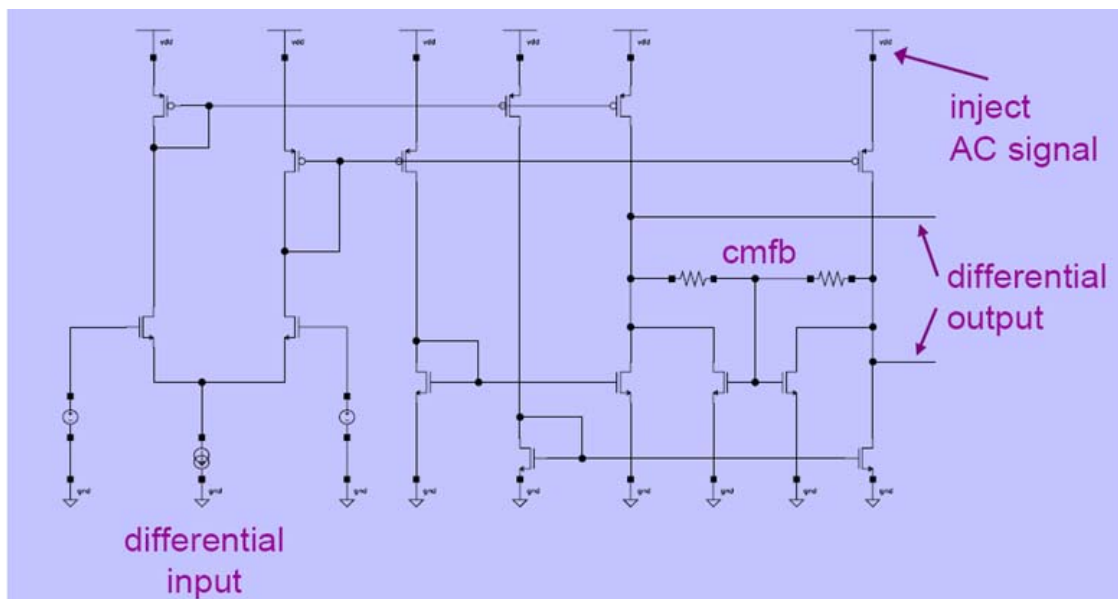


Figure 153 Fully Differential Amplifier: Calculate Effect of Variations on Power Supply Rejection and Feedthrough

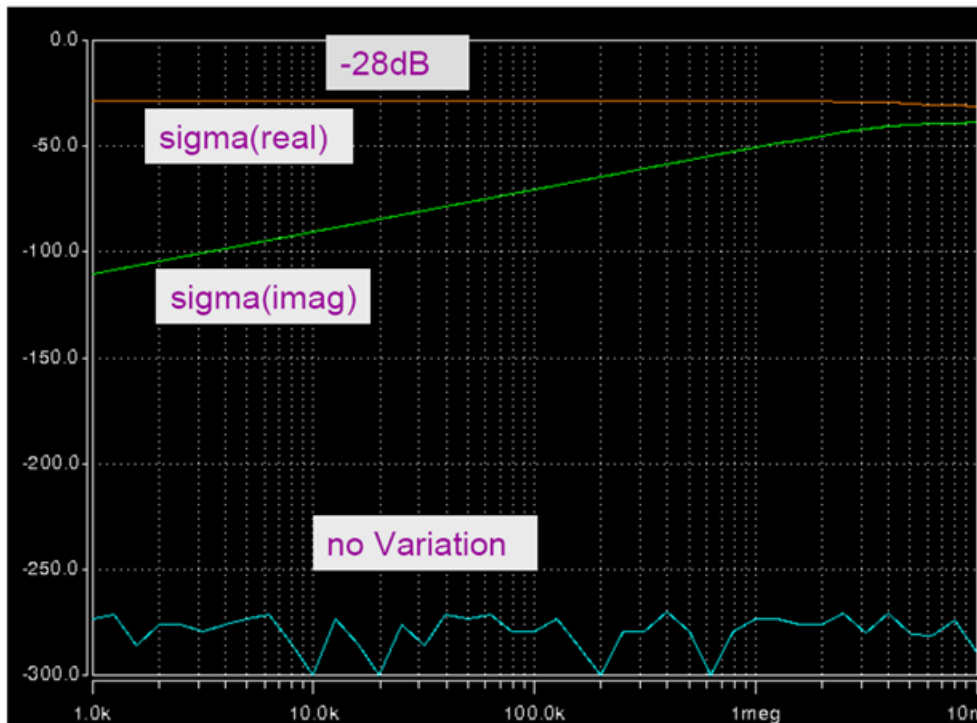


Figure 154 Power Supply Feedthrough

Application Considerations

ACMatch analysis results match those of a small signal Monte Carlo Analysis. Discrepancies arise with certain test setups, if the operating point in Monte Carlo analysis varies by a large amount. For example, the output of an amplifier might saturate at one of the supply rails for some samples, if it is configured for high gain at DC. If such conditions exist, and the amplifier is used with the same gain configuration in the real application, then they point to issues which need to be investigated with DCMatch analysis and resolved first. Otherwise, the DC gain configuration of the amplifier needs to be changed in the test setup.

Mismatch Compared to Monte Carlo Analysis

DCMatch and ACMatch analyses use calculus of probability instead of sampling. The following table shows a comparison of the two types.

Feature	Mismatch	Monte Carlo
Analysis type	DC or AC	AC, DC, Transient
Device/parameter contribution report	Yes	Yes, with the HSPICE data mining flow report
Relative run time	Fast	Slow
Accuracy	Good	Dependent on number of samples
Distributions	Normal preferred	Any
Variation result report	Global, Local, Spatial, and Interconnect separate	Global, Local, Spatial, and Interconnect combined

Supported Models with DCMatch and ACMatch

The following models are supported with DCMatch and ACMatch:

- **MOSFET:** LEVEL 49/53, 54, 66, 69
- **BJT:** 1,2, 6 and 8
- **R:** LEVEL 1
- **C:** LEVEL 1

References

- [1] M.Pelgrom, A.Duinmaijer, and A.Welbers: "Matching Properties of MOS Transistors", IEEE J. Solid-State Circuits, pp. 1433-1439, May 1989.
- [2] P.R.Kinget: "Device Mismatch and Tradeoffs in the Design of Analog Circuits", IEEE J. Solid-State Circuits, pp. 1212-1224, June 2005.

Chapter 25: Mismatch Analyses
References

Monte Carlo Data Mining

Describes the data mining capabilities of Monte Carlo results.

Running Monte Carlo simulations is expensive and the raw outputs may not provide sufficient insight into the circuit behavior or guidance on how the circuit could be improved. Usually, designers import the data into external tools for graphical and analytical analyses. The following describes techniques and files that provide post-processing capability within HSPICE itself.

The files described in this chapter are automatically generated with the Variation Block style description. Use `.Option Sampling_Method=SRS` (or one of the other choices) with the traditional AGAUSS style. See [Using Traditional Variation Format with Advanced Sampling Methods in Monte Carlo](#) in [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#).

For example cases and related `*.mpp0` demo files, see [Variability Examples](#).

The following sections discuss these topics:

- [Post-Processing of Monte Carlo Results](#)
- [Standalone Data Mining in HSPICE](#)

Post-Processing of Monte Carlo Results

The HSPICE data mining report, available in `*.mpp#`, `*.corner`, and `*.annotate`) files, provides extensive data including statistical summary, correlated contribution of IRVs on elements, design corner samples, and design environment back-annotation information.

HSPICE does post-processing on `*.ms0` (`*.ma0`, `*.mt0`) and `*.mc0` files, and generates output tables in a file with the suffix `*.mpp#`. After echoing the options used for the simulation, the output is displayed in a set of tables which

are grouped into two sections: a summary statistics section and a variable screening section.

The following sections discuss these topics:

- [Summary Statistics](#)
- [Variable Screening](#)
- [External Sampling — *.corner File](#)
- [Back-Annotation — *.annotate File](#)

Summary Statistics

The Summary Statistics section of the *.mpp0 file lists the number of sample, number of failures and a listing of responses with such a minor variation that they are dropped from further analysis. See the top section in [Figure 155 on page 861](#).

The subsections consist of:

- [Sample Moments and Robust Statistics](#)
- [Sample Quantiles and Quartiles](#)
- [Extremes and Corners](#)

Note: The *.mpp0 file is generated with only a statistical summary in the case of the traditional flow (Monte Carlo simulation without the `.OPTION SAMPLING_METHOD`).

A sample summary statistics section is shown in [Figure 155 on page 861](#).

```

Summary statistics
=====
Number of Samples: 5000
Number of Failures: 0
Responses with very small variation dropped from further analysis:
(variable = value)
in_fall_tran      = 4.000e-10
in_rise_tran      = 4.000e-10

Sample moments etc:
-----
(mad is a robust estimate of stdDev)

```

	mean	median	stdDev	mad	skewness	kurtosis
avg_cur	9.513e-04	9.542e-04	2.048e-05	1.652e-05	-1.0360	5.2973
fall_delay	2.708e-10	2.709e-10	2.451e-11	2.401e-11	0.1384	3.3073
out_fall_tran	2.628e-10	2.591e-10	1.931e-11	1.327e-11	1.6148	6.2205
out_rise_tran	5.108e-10	5.063e-10	5.205e-11	4.944e-11	0.5026	3.5413
rise_delay	7.058e-10	7.022e-10	6.484e-11	6.375e-11	0.3369	3.3121
total_cur	4.637e-13	4.513e-13	1.350e-13	1.299e-13	0.4760	3.5734
total_pwr	1.159e-12	1.128e-12	3.375e-13	3.248e-13	0.4760	3.5734

```

Quantiles:
-----
(Q50 is median, see above)

```

	Q01	Q05	Q25	Q75	Q95	Q99
avg_cur	8.854e-04	9.129e-04	9.421e-04	9.645e-04	9.777e-04	9.933e-04
fall_delay	2.154e-10	2.315e-10	2.546e-10	2.870e-10	3.105e-10	3.314e-10
out_fall_tran	2.332e-10	2.400e-10	2.510e-10	2.692e-10	3.092e-10	3.304e-10
out_rise_tran	4.055e-10	4.329e-10	4.749e-10	5.420e-10	6.035e-10	6.504e-10
rise_delay	5.668e-10	6.052e-10	6.611e-10	7.470e-10	8.179e-10	8.735e-10
total_cur	1.884e-13	2.565e-13	3.708e-13	5.465e-13	6.932e-13	8.316e-13
total_pwr	4.709e-13	6.412e-13	9.270e-13	1.366e-12	1.733e-12	2.079e-12

```

Extremes and corners:
-----

```

	min	max	minIndex	maxIndex
avg_cur	8.445e-04	1.017e-03	3744	1560
fall_delay	1.834e-10	3.825e-10	2755	2539
out_fall_tran	2.222e-10	3.415e-10	2026	4524
out_rise_tran	3.399e-10	7.689e-10	1588	1694
rise_delay	5.001e-10	1.011e-09	1588	1694
total_cur	7.749e-14	1.106e-12	4537	1588
total_pwr	1.937e-13	2.765e-12	4537	1588

Figure 155 Summary statistics section of *.mpp0 file

Sample Moments and Robust Statistics

The section called Sample Moments and Robust Statistics includes the sample mean, standard deviation, skewness, and kurtosis (Figure 156 on page 862).

The skewness and kurtosis are the third and fourth central moments of the data. Skewness is zero for symmetric distributions (like the normal distribution) and the normal distribution has a kurtosis of three. Significant departures from (0, 3) for the skewness and kurtosis indicate non-symmetry and departure from normality for the distribution.

The median and MAD statistics are also reported in this section. These are the robust estimates of the mean and standard deviation, respectively. Unlike the mean and standard deviation, the median and mad are insensitive to outliers in the data.

Sample moments etc:

(mad is a robust estimate of stdDev)

	mean	median	stdDev	mad	skewness	kurtosis
leakpwr	2.467e-05	2.443e-05	2.268e-06	2.090e-06	0.5364	3.5106
systoffset1	1.448e-03	1.433e-03	1.129e-03	1.103e-03	4.546e-02	2.8158
systoffset2	1.112e-03	1.104e-03	5.483e-04	5.345e-04	5.342e-02	2.8395
systoffset3	9.241e-04	9.175e-04	5.625e-04	3.566e-04	6.209e-02	2.8618
systoffset4	7.872e-04	7.818e-04	2.718e-04	2.675e-04	7.349e-02	2.8873

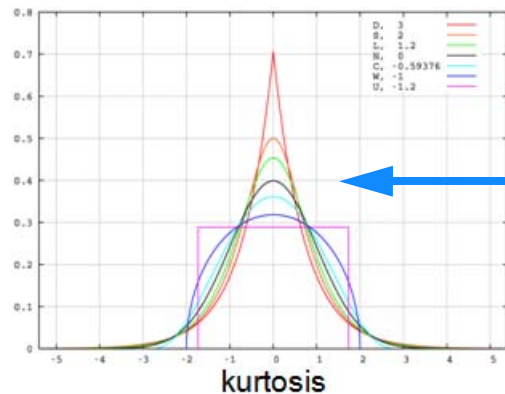
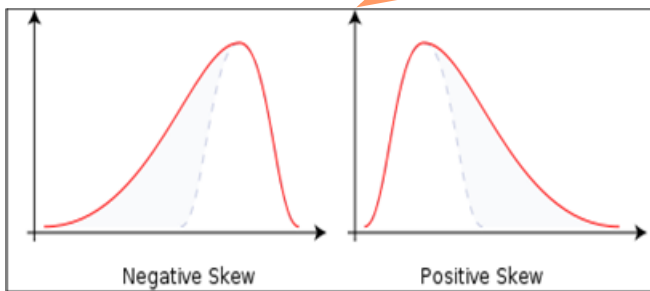


Figure 156 Sample moments section of the *.mpp0 file

Confidence Interval Levels

(F-2011.09-SP1) The Sample Moments section of the *.mpp0 file includes the confidence interval (CI) percentage on the top line of the section and as the first column in the Summary Statistics portion of the file.

A confidence interval is an interval in which a measurement or trial falls corresponding to a given probability. *Bootstrapping* is a computer-based method for assigning measures of accuracy to sample estimates. The process is as follows:

After running Monte Carlo with 20 samples, HSPICE produces a measurement output variable 'systoffset1' with a set of values:

```
systoffset1 =  
1.400e-03  1.254e-03  7.500e-04  4.476e-03  2.584e-03  9.569e-04  
6.650e-04  -1.410e-04  3.142e-03  1.753e-03  -1.695e-04  7.377e-04  
3.634e-03  -6.850e-04  1.632e-03  1.690e-03  6.391e-04  -4.123e-04  
2.571e-03  2.506e-03
```

The sample moments based on above samples are:

	mean	median	stdDev	mad	skewness
kurtosis	systoffset1	1.449e-03	1.327e-03	1.383e-03	1.384e-03
4.142e-01	2.397e+00				

To calculate the confidence intervals of above moments, HSPICE resamples `systoffset1`'s 20 results 500 times.

For each resample, HSPICE calculates the sample:

- Mean
- Median
- stdDev
- MAD
- Skewness
- Kurtosis

Generate a histogram of these 500 moments, then pick out the 2.5th and 97.5th percentiles. (This is the same as sorting the 500 moments and selecting the 12th~13th avg and 488th sorted moments. These percentiles do actually estimate the percentiles of the true distribution of moments.)

Take mean value for example: after the bootstrapping resampling method, the 95% confidence interval of the mean is (9.060e-04, 2.041e-03), which means that you can be fairly confident that the true `systoffset1` mean value is between (9.060e-04, 2.041e-03).

Current limitations:

- Resampling size is a fixed value (500 samples).
- Confidence interval value is a fixed value(95%).

See [Figure 157 on page 864](#) for listing of Confidence intervals (CI) in Summary Statistics section of the `*.mnp0` file.

Chapter 26: Monte Carlo Data Mining
 Post-Processing of Monte Carlo Results

```

Summary statistics
=====
Number of Samples: 20
Number of Failures: 0
Sample moments etc:
-----
CI: 95% Bootstrap confidence intervals based on 500 re-samples

      CI      mean      median      stdDev      mad      skewness      kurtosis
-----
2.5%  9.060e-04  6.978e-04  9.524e-04  6.133e-04 -3.725e-01  1.612e+00
systoffset1  1.449e-03  1.327e-03  1.383e-03  1.384e-03  4.142e-01  2.397e+00

97.5% 2.041e-03  2.129e-03  1.727e-03  2.160e-03  1.050e+00  3.972e+00

Quantiles:
-----
      CI      Q01      Q05      Q25      Q75      Q95      Q99
-----
2.5% -6.850e-04 -4.259e-04 -2.302e-04  1.348e-03  2.510e-03  3.036e-03
systoffset1 -6.332e-04 -4.259e-04  6.585e-04  2.522e-03  3.676e-03  4.316e-03

97.5% -1.587e-05  7.341e-04  1.180e-03  3.265e-03  3.676e-03  4.476e-03

User Specified Quantiles:
-----
      CI      Q00.1      Q10      Q90      Q99.9
-----
2.5% -6.850e-04 -4.396e-04  1.779e-03  3.131e-03
systoffset1 -6.798e-04 -1.938e-04  3.191e-03  4.460e-03
97.5% -1.541e-04  7.304e-04  3.718e-03  4.476e-03
  
```

Figure 157 Confidence intervals recorded in Summary Statistics

Sample Quantiles and Quartiles

The sample quantiles and quartiles are displayed next ([Figure 158 on page 865](#)). These measures are also robust to outliers and provide better visibility into the distributional behavior in the tails. Quantiles are points taken at regular intervals from the cumulative distribution function (CDF) of a random variable. They are found by counting the number of samples that fall to the left of the particular point in the empirical cumulative distribution function. Thus 25 percent of the samples fall below the Q25 quartile.

Chapter 26: Monte Carlo Data Mining

Post-Processing of Monte Carlo Results

```
.Variation
Option Other_Percentile = percentile
.end_variation
.data percentile
    q
    0.001
    0.1
    0.25
    0.75
    0.9
    0.999
.enddata
```

This option allows you to help to see how much impact there is from trailing data points, or to count samples near the absolute minimum for a sample set. You can set Q to gauge the true population of values near or around the absolute minimum.

The data value range is $(0 < x < 1.0)$. HSPICE issues a warning message if any values exceed the range and the data mining feature ignores these values. Because a data block can have more than one column, only the first column is processed when you use `Option Other_Percentile`.

The following rows are added in the `*.mpp0` file for user-defined percentiles; the confidence intervals are calculated and displayed, as well:


```

.....
User Specified Quantiles:
-----
                Q(0.1%)   Q(10.0%)   Q(25.0%)   Q(75.0%)
Q(90.0%)   Q(99.9%)
                2.5%
2.578e-05  2.750e-05   1.689e-05  2.177e-05  2.295e-05
3.214e-05
leakpwr                1.992e-05  2.196e-05  2.311e-05
2.592e-05  2.780e-05
3.266e-05
                97.5%
2.609e-05  2.804e-05   2.010e-05  2.211e-05  2.322e-05
3.266e-05
                2.5%
2.114e-03  2.805e-03   -1.669e-03 -8.952e-05  5.768e-04
4.402e-03
systoffset1                -1.620e-03  4.004e-05  7.043e-04
2.200e-03  2.914e-03
4.643e-03
                97.5%
2.286e-03  3.045e-03   -1.323e-03  1.139e-04  7.738e-04
4.643e-03
.....

```

Extremes and Corners

The minimum and maximum values for the measures, together with the sweep indices at which they occur, are given next (Figure 159 on page 867).

```

Extremes and corners:
-----
                min           max           minIndex   maxIndex
leakpwr                1.689e-05  3.409e-05   729         790
systoffset1           -1.669e-03  5.049e-03   648         969
systoffset2           -4.006e-04  2.865e-03   648         969
systoffset3           -7.319e-05  2.075e-03   648         969
systoffset4                4.285e-05  1.689e-03   648         661

```

Figure 159 Extremes and corners section of the *.mpp0 file

Variable Screening

The table in this section is similar to the DC and ACMatch tables shown in [Chapter 25, Mismatch Analyses](#). The contribution of each independent variable to the response variability is computed. HSPICE uses correlation to measure the variability contribution.

See [Table 80 on page 871](#) for a listing and description of the column headings.

Variable Screening: Response and Important Elements/Models

By default, HSPICE uses the Pearson correlation coefficient for screening. See http://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient. The Pearson correlation is a suitable measure when the responses are approximately linearly related to the dominant sources of variation. The more general Spearman rank correlation coefficient (see http://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient), which only requires monotone behavior, can also be used by specifying the Variation Block command `Option Screening_Method=Spearman`. The contributions from the independent variables are then aggregated to the element level and sorted by importance. Where possible, matched groups are identified. Careful layout techniques like common centroid can be used to reduce the influence of spatial variation and thermal/voltage gradients for matched groups. See [Figure 160 on page 869](#).

Note: In cases where all variation contributions are smaller than 0.05%, at least 10 element variations are reported.

HSPICE issues a warning noting that the results are below the cutoff point:

```
**warning** not all element variation contributions are above  
0.05%
```

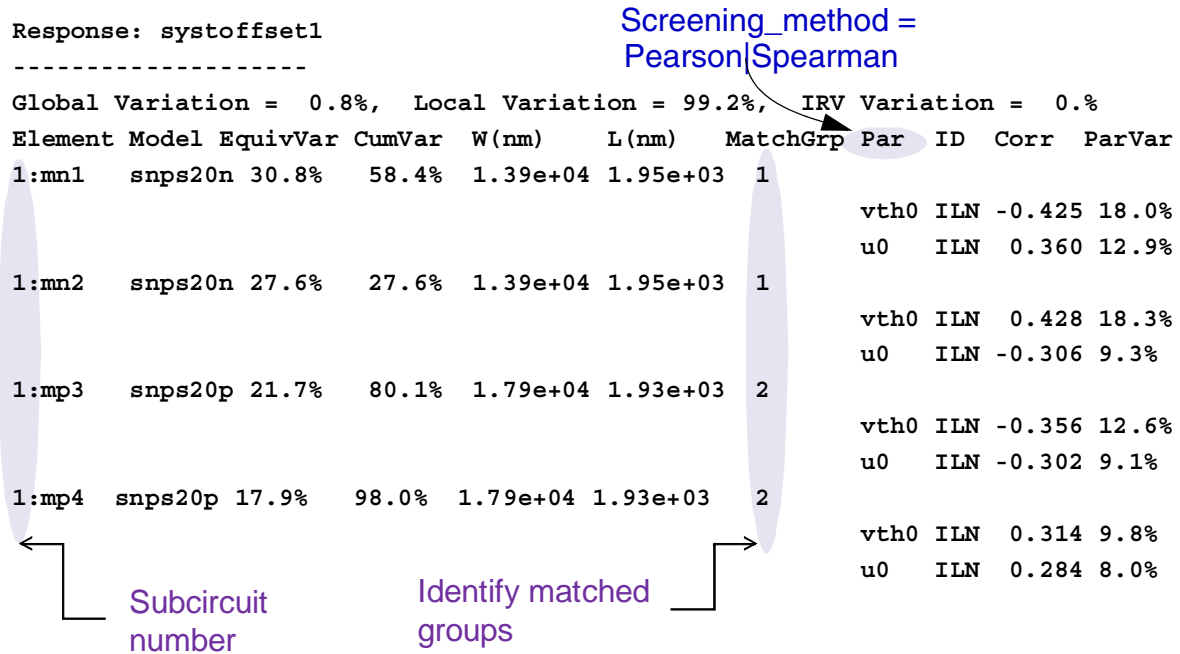


Figure 160 Response subsection of the Variable Screening portion of the *.mpp0 file

Important elements or models

The information in this section helps you focus on corner cases and extracts data on subcircuit encoding and selective simulation.

```
Important elements or models
=====
1:r0 snps20n 1:mn2 1:mn1 1:mp3 1:mp4

subcircuit encoding
=====
Key Subckt
1: xi82

Selective simulation -----
To simulate corners only:
Use: ... Monte = list(648, 729, 790, 969)
```

Simulate these "corners" to validate Robustness after device size changes

Figure 161 Information to simulate a *corner file

Response Correlation Matrix

HSPICE computes the correlation between responses and removes highly correlated responses to lessen output file size.

```
Response Correlation matrix (%)
=====
      leakpwr systoffset1 systoffset2 systoffset3 systoffset4
leakpwr      100.0      99.1      99.2      99.2      99.2
systoffset1   99.1      100.0     100.0     100.0     100.0
systoffset2   99.2      100.0     100.0     100.0     100.0
systoffset3   99.2      100.0     100.0     100.0     100.0
systoffset4   99.2      100.0     100.0     100.0     100.0

Delete highly correlated variables:
=====
Deleting following responses:
systoffset2 systoffset3 systoffset4
```

Compute correlation between responses, delete highly correlated responses to avoid output clutter

Figure 162 Response Correlation matrix

Variable Screening Table Headings

Table 80 Table Headings for Variable Screening in the *.mmp0 File

Column Heading	Description
Element	Name, if applicable.
Model	Name of model.
EquivVar	Normalized variance associated with an element or model; it is the sum of all local/global variation terms associated; uses normalized values for ease of comprehension. (Elements with EquivVar smaller than 0.05 percent are not reported.)
CumVar	The elements and models ordered by their contribution and shown as cumulative contributions.
W	Width of element in nanometers.
L	Length of element in nanometers.
MatchGrp	Mainly for analog and SOC designs. The influence of spatial variation, including thermal and bias gradients, is reduced if devices with the same MatchGrp index are close together and in the same orientation on the layout, preferably common-centroid. With the HSPICE F-2011.09 release, the *.mmp0 file reports for all instance parameters for matched group MOSFETs for the following models: PSP/BSIM/BSIMSOI and instance parameters W and L for all other models. See Figure 163 on page 872 .
Par	Parameters associated with the raw correlations.
ID	Form of variation (local, element, global, and so forth.)
ParVar	Contents are related to the square of the Corr column and gives the equivalent normalized variance associated with the Corr column. The sum of this column is 100 percent if all sources of variation are displayed. The values in the Corr column are aggregated to the element level for local variation and to the model level for global variation. This drives the values in the left half of the table. (Parameters with ParVar smaller than 0.01 percent are not reported.) By default, HSPICE uses the Pearson correlation coefficient to calculate the ParVar items (you can optionally use the Spearman method, controlled by Variation Block Option <code>Screening_Method</code>).

*Table 80 Table Headings for Variable Screening (Continued) in the *.mpp0 File*

Column Heading	Description
	<p>The Pearson equations are:</p> <p>Pearson correlation coefficient (default)</p> $\rho_{x,y} = \frac{\text{cov}(X, Y)}{\sigma_x \sigma_y} = \frac{E[(X - \mu_x)(Y - \mu_y)]}{\sigma_x \sigma_y}$ <p>Spearman correlation coefficient, $d_i = x_i - y_i$</p> $\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$
Corr	<p>The raw correlations are shown in the Corr column. These are associated with parameters in the Par column. Note: If the netlist has only one or more than 50 measure statements, no response correlation matrix is reported.</p>

Instance parameters listing for elements in match groups

[Figure 163 on page 872](#) breaks out instance parameters for elements in match groups.

```

Instance parameters for elements in match groups
=====
The following are elements defined with PSP/BSIM4/BSIMSOI models.
1.mn1 l=1E-06 w=7E-06 pd=5.5E-07 ps=4.8E-07 ad=1.89E-14 as=1.4175E-14
1.mn2 l=1E-06 w=7E-06 pd=5.5E-07 ps=5.5E-07 ad=1.89E-14 as=1.89E-14
1.mn6 l=3E-06 w=1.8E-05 pd=7.5E-07 ps=8.5E-07 ad=2.835E-14 as=3.51E-14
1.mn8 l=3E-06 w=1.8E-05 pd=0.39 ps=5.4E-07 ad=9.45E-15 as=1.54E-14
1.mp3 l=1E-06 w=9E-06 pd=4.8E-07 ps=5.5E-07 ad=1.4175E-14 as=1.89E-14
1.mp4 l=1E-06 w=9E-06 pd=5.4E-07 ps=4.7E-07 ad=1.82E-14 as=1.365E-14

Instance parameters for elements in match groups
=====
Element          W (nm)      L (nm)
1.mn6             1.8e+04     3.0e+03
1.mn8             1.8e+04     3.0e+03
1.mp3             9.0e+03     9.3e+02
1.mp4             9.0e+03     9.3e+02
  
```

Figure 163 Instance parameters for PSP/BSIM4/BSIMSOI models (top) and all other models — W and L instance parameters, only (bottom)

External Sampling — *.corner File

Designers sometimes use Monte Carlo to generate random test corners to evaluate design robustness. Yield or distribution estimation is not the primary objective. This is somewhat similar to random test vectors in digital design.

However, Monte Carlo, especially transient Monte Carlo, is computationally expensive. Some customers run DC Monte Carlo to find corners, and reuse these corners in transient Monte Carlo. Traditionally, this requires two steps. First, run DC Monte Carlo, and then run transient Monte Carlo with the corners extracted from the DC analysis.

The Monte Carlo results may lead to design changes, for example, by increasing device sizes to reduce the impact of local variation. The data mining techniques help in reducing the cost of additional Monte Carlo runs that may be done after the design changes. Such changes fall into two classes.

If only device sizes are changed, and the circuit structure remains the same, then the validation run could simply recompute values at the same sample points that created the extreme values for the responses. These indices are listed in the statistical summary portion of the *.mpp0 file and can be used in the list form on the Monte Carlo command. For example, the circuit with the *.mpp0 file section shown in [Figure 161 on page 869](#) would be run with:

```
.TRAN 0.1n 10n sweep Monte=list(648, 729, 790, 969)
```

The above method fails if elements are added or deleted from the design because the association of the random numbers to the independent variables changes. To support such use, HSPICE creates a *.corner file that can be imported in a subsequent MC simulation using the external sampling option: `Option Sampling_Method=External` with the Variation Block style representation or with `.option sampling_method = external` with traditional AGAUSS style.

Selective Simulation (*.corner)

External Sampling

=====

```
.Option Sampling_Method = External Block_Name = Corners
.Data Corners
index snps20n:@:vth0:@:IGN snps20n:@:u0:@:IGN
648 -0.198 -0.346
729 2.3876 0.584
790 -2.1946 -0.7195
969 -0.1663 -0.2071
.Enddata
```

Facilitates quick robustness validation following minor topology changes

For external sampling, list all IRV values at corner sampling indices

Figure 164 External Sampling

A *.corner file can be imported into a Microsoft Excel format by specifying .OPTION MEASFORM = 1 or 3 (default is 1).

HSPICE uses the same random numbers as in the prior simulation for the elements that are common between the old and new circuits and assigns new random numbers, with appropriate distributions, to any additional elements. Random numbers associated with deleted elements are ignored.

Character separators in *.corner files are indicated by the :@: (colon—"at" sign-colon) combination. This combination avoids confusion with hierarchy separators which use a single @ character.

Corner Analysis - DC Monte Carlo/Transient Analysis

Beginning with G-2012.06, the HSPICE corner analysis solution combines a DC Monte Carlo run followed by a transient Monte Carlo trial with the corners extracted from the DC analysis. Corners are defined as the Monte Carlo samples that lead to extreme values of the measures. There are two "corners" for any measure, the *min* and the *max*.

Rather than focus on only the min/max values, HSPICE enables selection of outlying values from the tails. These are specified as a fraction of the total sample size, for example, 0.1 percent. The more general corner definition is preferable to reduce design risk.

The following sections discuss these topics:

- [DC Monte Carlo Syntax](#)
- [Syntax for Transient Monte Carlo](#)
- [Limitation](#)
- [Example](#)

DC Monte Carlo Syntax

```
.DC var1 type np start1 stop1  
+[SWEEP MONTE=MCcommand] [corner_percentile=val]
```

where: `corner_percentile=val` specifies the percentiles used to find corners. The value is a non-negative number in the range (0.0~0.5). For example, if `value=0.1`, then HSPICE sorts the measure results: the points below the 10th percentile and those above the 90th percentile are chosen as corners. If the value = 0.0, then HSPICE uses the maximum and minimum values as corners. The default is 0.0. The following chart explains the corner percentiles for a response variable that has a Gaussian distribution:

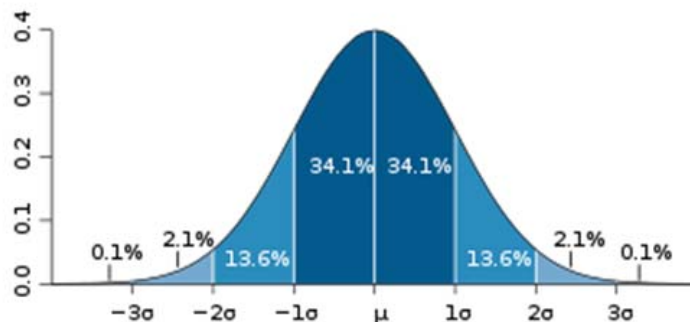


Figure 165 Bell-curve to determine corners

- The dark blue zone at the center represents observations within one standard deviation on either side of the mean, which accounts for 68.2 percent of the population.
- Two standard deviations from the mean (dark and medium blue) account for 95.4 percent, and 3-sigma (dark, medium, and light blue) for about 99.7 percent.
- The sample points with 0.1 percent `corner_percentile` represent extreme points which have variations larger than 3.1-sigma.

If multiple DC measures use the `corner_percentile` option, the union of all the corners is used in the next transient simulation.

When HSPICE finds corners during DC analysis:

- HSPICE only generates `*.ms#` files to locate the corners during data mining. Other output files (`*.sw0`, `*.pa0`, and so forth) are not generated.
- The `*.lis` file is simplified and no Monte Carlo-related information is printed.
- Data mining files (`*.mpp0` and `*.annotate`), which are relatively small, are created. Large files such as `*.mc0` are suppressed.

During DC Monte Carlo, a `*.corner` file is also generated with content as follows:

```
*=====
* corner percentile = value
*Selective simulation for extreme corners:
*Monte= list ( corner_number1, corner_number2, ..., corner_numberN)
*Number of samples = K
```

See [External Sampling — *.corner File on page 873](#) for more information.

Syntax for Transient Monte Carlo

```
.TRAN tstep1 tstop1 [tstep2 tstop2 ...tstepN tstopN]
+ [START=val] [UIC] [SWEEP MONTE=dc_corner]
```

where: `dc_corner` is a keyword for Monte Carlo simulation only. With this option, HSPICE reuses the corners generated in DC Monte Carlo and runs transient analysis with these corners as random values.

Limitation

DC Monte Carlo does not currently support the CDPL distributed mode.

Example

The following is a complete netlist to illustrate running a transient simulation with DC corners.

```
* run Tran simulation with DC corners
*.option mcfast
.options POST OPTS LIST MODMONTE=1 numdgt=10 measdgt=10
.model nch nmos LEVEL=54 TOXM=4E-9 wint=wint_nch VERSION=4.4
.param wint_nch = agauss ( 3e-7 , 1e-7 , 3.0 )

m11 2 11 0 0 nch W=1E-6 L=0.15E-6 dtemp=10
m12 2 11 0 0 nch W=1E-6 L=0.15E-6 dtemp=10
m13 2 11 0 0 nch W=1E-6 L=0.15E-6 dtemp=10
v01 2 0 1.5
v02 11 0 0.0 pulse (0 1 0 1n 1n 10n 20n)

.dc v02 0 2.0 0.1 sweep monte=500 corner_percentile=0.001
.meas dc ids_1 find par('i(m11)*1E3') at = 1.5
.meas dc ids_2 find par('i(m12)*1E3') at =1.5
.meas dc ids_3 find par('i(m13)*1E3') at =1.5

.tran 1n 1u sweep monte= dc_corner
.meas tran i1 find i(m11) at=1u
.meas tran i2 find i(m12) at=1u
.meas tran i3 find i(m13) at=1u

.end
```

Back-Annotation — *.annotate File

HSPICE provides a file that contains the key local variability contributors and matched pairs for display on the schematic in CustomDesigner. A simple tabular layout is provided for information exchange. For each response, there is the full hierarchic element name, the variability it contributes, and the Match Group value.

Standalone Data Mining in HSPICE

Sometimes, users create additional “measurements” outside HSPICE and wish these to be included in the data mining. This requires the new measurements to be formatted in additional files.

Chapter 26: Monte Carlo Data Mining

Standalone Data Mining in HSPICE

You can enter the following on the command line to have HSPICE skip netlist readin, errchk, and simulation, and do standalone data mining:

```
hspice -datamining -i datamining.cfg [-o outname]
```

The configuration file content includes:

- *comments/description
- *Required records
- .sampleFile input.mc0
- .measFile input .mt0 input .mt0A input.mt0B ...
- .Option Screening_Method = Pearson | Spearman

Note: The .sampleFile and .measFile keywords are case-sensitive.

DC Sensitivity Analysis and Variation Block

Describes enhanced sensitivity analysis in HSPICE focusing on DC simulation using Variation Block.

DC sensitivity analysis allows you to do the following tasks:

- Compute sensitivity of a model parameter.
- Compute parameter sensitivity of many more models than the traditional HSPICE `.SENS` command.
- Generate sensitivity for `.PROBE` and `.MEASURE` output commands.
- Generate sensitivity for DC sweeps.

DC sensitivity analysis is supported in conjunction with a single DC sweep, or DC sweep and Monte Carlo.

Note: HSPICE ships numerous examples for your use. See [Listing of Demonstration Input Files](#) for paths to demo files.

This chapter covers the following topics:

- [Sensitivity Block Using the Variation Block Construct](#)
- [Input Syntax](#)

Sensitivity Block Using the Variation Block Construct

The Variation Block serves as the default sensitivity block. In other words, the DC sensitivity analysis calculates the sensitivity for model and element parameters, which are specified in the Variation block.

Sensitivity to Local Variations

Sensitivity is reported by varying one parameter on one device at a time. So it is identified by device and parameter:

$$\text{SENS_Local}(p_1, k)$$

where, k is the device and p_1 is the parameter.

Sensitivity to Global Variations

Sensitivity is reported for varying one parameter simultaneously on all devices. So it is identified by model type, model, and parameter according to this equation:

$$\text{SENS_Global}(T, M, p_1) = \text{SUM}_k \text{SENS_Local}(p_1, k)$$

where k loops over all devices with the same model type T and model M . p_1 is the parameter.

Input Syntax

```
.DCSENS Output_Variable [File=string] [Perturbation=x]  
+ [Interval=SweepValue] [Threshold=x] [GroupByDevice=0|1]
```

Here, `Output_Variable` is the response with respect to the parameters designated in the Sensitivity Block. It can be a node voltage or branch current in the circuit.

Argument	Description
Output_Variable	Response with regard to the parameters designated in Sensitivity Block. Similar to the <code>.DCMATCH</code> command, the <code>Output_Variable</code> can be a node voltage or a branch current in the circuit.
File=string	Valid file name for the output tables. Default= <code>basename.ds#</code> where “#” is a number in the style of <code>ds0</code> , <code>ds1</code> , and so on. If multiple <code>dcswep</code> commands are specified in the netlist, then the sensitivity analysis table results for each <code>dcswep</code> are listed in <code>*.ds#</code> files. If <code>.OPTION OPFILE</code> is specified, sensitivity result tables on operating points are listed in <code>*.dp#</code> files. Otherwise, these tables are in the <code>*.lis</code> file.

Argument	Description
Perturbation= <i>x</i>	Perturbations of <i>x</i> standard deviation used in computing the finite difference approximations to device derivatives. The valid range for the parameter is 0.0001 to 1.0; default =0.05.
Interval=SweepValue	Positive integers for SweepValue. This option only applies to one-dimensional sweeps. A summary is printed at the first sweep point, and for each subsequent increment of SweepValue. The Interval key is ignored with a warning if a sweep is not being carried out. The option only controls the printed summary table. The analysis may be carried out at additional sweep values if required by other forms of output such as <code>.PROBE</code> and <code>.MEASURE</code> commands.
Threshold= <i>x</i>	Minimum value for reporting of absolute sensitivity. Only devices with absolute sensitivity value above <i>x</i> are reported. Results for all devices are displayed if Threshold=0 is set; default=10u.
GroupByDevice = 0 1	<ul style="list-style-type: none"> ▪ 0: (Default) Alternate mode of generating sensitivity result tables ▪ 1: Table form follows the <code>.SENS</code> command output.

Chapter 27: DC Sensitivity Analysis and Variation Block
Input Syntax

Part 5: Optimization

This Part contains the following chapters/topics.

- [Chapter 28, Exploration Block](#)
- [Chapter 29, Optimization](#)
- [Chapter 30, Timing Analysis Using Bisection](#)

Exploration Block

Describes the use of the Exploration Block in HSPICE.

The Exploration Block addresses the need to study the behavior and sensitivities of circuits to come up with an optimum design. During this early design phase, you may want to explore ranges of device sizes for a given circuit topology. The Exploration Block feature allows you to describe a set of experiments with different geometries, without changes to the original netlist.

The Exploration Block is closely related to the Variation Block with external sampling (see [Chapter 23, Variability Analysis Using the Variation Block](#)).

HSPICE ships hundreds of examples for your use; see [Chapter 31, Running Demonstration Files](#) for paths to demo files.

Topics:

- [Exploration Block Functions](#)
- [Usage Guidelines](#)
- [Flow Using an External Exploration Tool](#)
- [Exploration Block Structure](#)
- [Export File Syntax](#)
- [Execution of Exploration in HSPICE](#)
- [Exploration Data Block Syntax](#)
- [Exploration and Variation Block Interactions](#)
- [Limitations](#)

Exploration Block Functions

The Exploration Block extracts the parameters suitable for exploration from a netlist, and thus it eliminates parsing by the Exploration tool. The parameters are presented in a normalized format. This solution eliminates the exploration tool's need to rewrite the netlist with new parameter values. Use of the Exploration Block returns only the updated values of the parameters which need to be changed in the course of a set of exploration simulations. HSPICE finds all the places where they need to be applied.

The Exploration Block contains a section with options and constraints, and it may include a data block with instructions on how to change certain parameters on individual devices or device groups. The data block must be created outside the simulator, based on information from the simulator and considerations specific to the particular design, possibly from an optimization program.

Usage Guidelines

To accommodate time restriction, exploration needs to be applied in an organized manner, with the smallest number of unrelated variables. A good approach for best results (partial and full matching) is to consider that integrated circuits are built with hierarchy, and that known relationships exist between devices. In essence, your knowledge about the circuit is encapsulated in the way exploration is carried out. Experience with optimization tools has shown that exploration methodology is crucial for your success, but is often difficult to set up correctly.

Multiple Instantiations of the Same Cell or Subcircuit

In a typical design process, a large circuit is assembled from cells out of existing libraries. Each cell has different descriptions, for different applications: subcircuit, layout, behavioral model, etc. The circuit netlist describes how the cells are connected with other cells, and it contains a description of their content. Exploration of HSPICE is currently restricted to the hierarchical mode only.

In the hierarchical mode, exploration is cell-oriented, meaning that all instantiations of a particular cell are affected the same way. With this usage

model, if you wish to explore separate instantiations of the same cell in a different manner, then you need to create new cell names with their content definitions repeated, before exploration can start. This renaming needs to be done anyhow for the final design, if you accept new device sizes coming out of the exploration because a basic rule of a circuit description is that multiple cell definitions with same name (and possibly different content) are not allowed.

Specifying Relationships between Devices

The following relationships between different devices can be specified in the Exploration Block to force matching:

```
device1Property1=expression of device2Property1
```

Such expressions reduce the number of variables for exploration because derived properties are processed inside of HSPICE. These relationship rules will be applied to all the devices subject to exploration. Therefore even if no change is requested from the Exploration tool, HSPICE executes these rules. So, if for example the lengths of devices `opamp.mn1` and `opamp.mn2` are different in the netlist, they will be the same in a simulation which contains an Exploration Block with the rule that they should be the same.

A simplified syntax expresses relationships of a whole set of device properties.

Examples:

```
length(opamp.mn1)=length(opamp.mn2)
```

```
opamp.mp4=opamp.mp3
```

mp4 will be identical to mp3, in all properties

```
bias.mn5=2*bias.mn6
```

mn5 consists of 2 devices in parallel, which are identical to mn6

Specifying Relationships between Device Properties

The following relationships can be specified to cover appropriate scaling of secondary properties on the same device:

```
deviceProperty2=expression of deviceProperty1.
```

Example

```
ad='120n*W' as='120n*W' ps='240n+W' pd='240n+W'
```

Subcircuits and Elements Supported for Exploration

The exploration feature is primarily designed for design work on integrated circuits in CMOS technology. Exploration is supported for the following subcircuits and elements:

- Independent sources: DC value
- MOS devices: W, L, M, dtemp
- Resistors: R or W, L, M, dtemp
- Capacitors: C or W, L, M, dtemp

When designing circuits, the multiplicity factor M is always a positive integer, but the Exploration tool can request arbitrary positive values.

To preserve relationships which have been previously defined through expressions, exploration can only be applied to parameters which are defined with numerical values.

Example 1

```
m1 out in1 vdd vdd pch w=wp l=100n m=3
```

Exploration can be applied to element parameters l and m , but not to w directly.

Example 2

```
subckt nand in1 in2 out wp=100n wn=50n len=100n
m1 out in1 vdd vdd pch w=wp l=len
m2 out in2 vdd vdd pch w=wp l=len
m3 out in1 mid gnd nch w=wn l=len
m4 mid in2 gnd gnd nch w=wn l=len
.ends nand
```

Exploration can be applied to subcircuit parameters w_p , w_n , and l_{en} . The application envisioned here is for leaf cells with programmable layout: separate width and common length of pmos and nmos devices.

Example 3

```
.subckt onebit in1 in2 carry-in out carry-out
x1 in1 in2 7 nand
x2 in1 7 8 nand wp=100n wn=100n
x3 in2 7 9 nand wp=300n wn=150n
.ends onebit
subckt nand in1 in2 out wp=200n wn=100n
m1 out in1 vdd vdd pch w=wp l=100n
m2 out in2 vdd vdd pch w=wp l=100n
m3 out in1 mid gnd nch w=wn l=100n
m4 mid in2 gnd gnd nch w=wn l=100n
.ends nand
```

The subcircuit named `onebit` can be used for exploration because it instantiates other subcircuits using parameters with numerical values: `wn` and `wp` of `nand` gates `x2` and `x3`. The subcircuit named `nand` can be used for exploration on the default values `wn` and `wp` (exploration only affects instantiation `x1` because `x2` and `x3` parameters override the default values). The devices `m1` to `m4` can be used for exploration on their length but not on the width. This preserves the imposed relationship of equal width for `m1` and `m2`, and for `m3` and `m4`.

Exploration supports variation in temperature, in addition to element and subcircuit parameters. Encrypted sections of a netlist are not available for exploration.

Flow Using an External Exploration Tool

The design flow consists of:

- An information extraction and export phase in HSPICE
- A definition phase for the Exploration Block outside of HSPICE
- An exploration phase in HSPICE

Information Extraction and Export Phase

HSPICE creates an output file with the Export Block. The file's first section contains the names of the variables suitable for exploration, parameters, element and subcircuit parameters, along with appropriate identifiers, which

include subcircuit name, device instance names, models and properties. In the second section, the corresponding values are listed.

From [Example 3 on page 889](#):

- Subcircuit `onebit` with properties `wn` and `wp` for instantiations `x2` and `x3`
- Subcircuit `nand` with properties `wp` and `wn` (only useful if an instantiation exists where `wp` and `wn` are not defined, as in `onebit.x1`)
- Devices `m1` to `m4` with property `l`

In the export phase, HSPICE runs a simulation from the originally supplied netlist, ignoring any Exploration Block content other than options.

Definition Phase (Outside HSPICE)

You must create or adapt the external utility described in the following items; it is not provided by HSPICE.

- An external utility reads the files created by HSPICE with the device information and any constraints.
- Supplemental information to the external utility consists of technology and details of the experiment.
- The utility creates a set of experiments and formulates them as a data block, with some or all variables contained in the Export Block, and one or more sets of exploration data.
- The utility submits the netlist with the Exploration Block to HSPICE.

Exploration Phase

These items are done in the exploration phase:

- HSPICE applies the content of the data block, calculates the secondary parameters and constraints, and runs a set of simulations with the updated device geometries as specified in the Exploration Block. HSPICE produces measurement results and a file with all the parameter values used for each exploration simulation.
- The external utility analyzes and combines the simulation results.
- Based on the results, the utility might specify another set of experiments, with a new set of simulations, and run through these steps until some predefined goal is reached.

Refer to the [Figure 166](#), and notice the flow difference before and after adding Exploration Block.

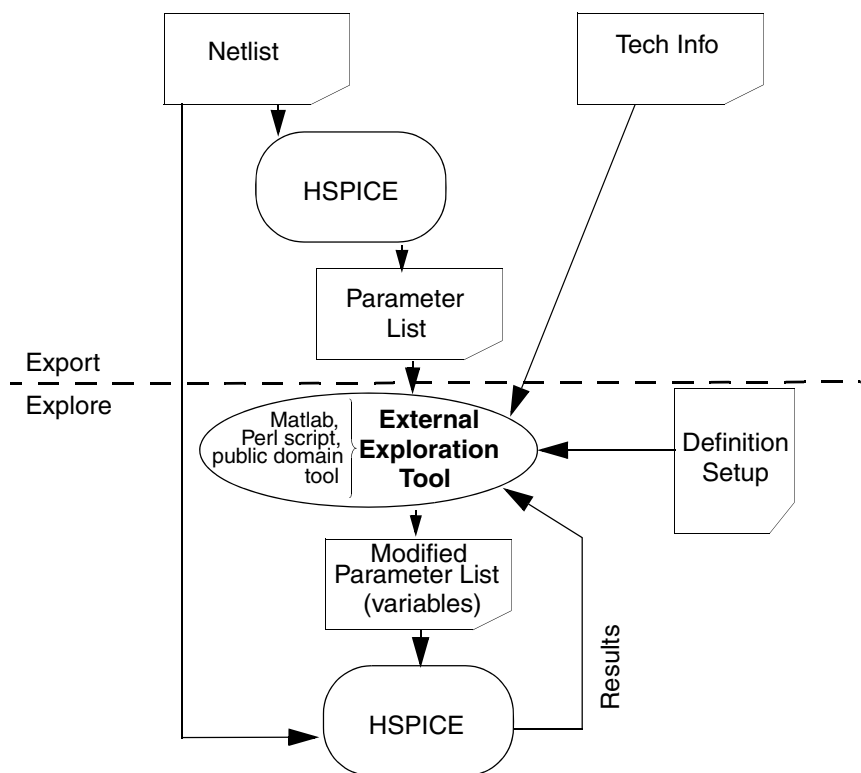


Figure 166 Exploration Block Flow

Exploration Block Structure

Because the Exploration Block is closely related to the Variation Block, the internal structure is similar, in particular when compared with external sampling. The major differences include:

- The Variation Block specifies variation on the model parameters, whereas the Exploration Block deals with the values as defined in the netlist.
- Perturbations specified in the Variation Block are applied in a flat manner, whereas those from the Exploration Block apply to subcircuits.
- The Exploration Block includes:
 - options
 - parameters
 - relationships between devices
 - relationships between properties
 - area calculation
 - data block characteristics
- A period is used as separator between a subcircuit name and an element name. For example: `opamp.rbias` refers to the resistor **rbias** instantiated in subcircuit **opamp**. A period is also used as separator between subcircuit names, if one subcircuit is defined within another. For example: `opamp.bias.rbias` refers to the resistor **rbias** in subcircuit **bias**, nested within subcircuit **opamp**.

Syntax

The following syntax shows the parts and sequence of an Exploration Block:

```
.Design_Exploration
  Options
    Parameter Parameter_Name = value
    Parameter Parameter_Name = expression
  .Data BlockName
    Index      Name      Name, ...
  ...
  .EndData
.End_Design_Exploration
```

Control Options

The options below are described in the tables that follow.

- Option Explore_only|Do_not_explore
- Option Export
- Option Exploration_method
- Option Ignore_exploration
- Option Secondary_param

If you want to explore only certain cells or subcircuits use:

Option	Description
Option Explore_only Subckts= SubcktList	This command is executed hierarchically — the specified subcircuits and all instantiated subcircuits and elements underneath are affected. Thus, if an inverter with name INV1 is placed in a digital control block called DIGITAL and in an analog block ANALOG, and Option Explore_only Subckts = ANALOG, then the perturbations only affect the INV1 in the analog block. You must create a new inverter INV1analog, with the new device sizes.
Option Do_not_explore Subckts= SubcktList	Excludes listed subcircuits.

The two modes of exploration are distinguished by setting either:

Option	Description
Option Export=yes	Exports extraction data and runs a simulation with the original netlist
Option Export=no	(Default) Runs a simulation with Exploration data

Chapter 28: Exploration Block

Exploration Block Structure

If you specify `Option Export=yes` followed by `Option Mex_File_Only=yes` HSPICE generates a `*.mex` file without running any simulations.

Option	Description
<code>Option Mex_File_Only=yes</code>	Generates a <code>*.mex</code> file without running any simulations.
<code>Option Mex_File_Only=no</code>	(Default) No <code>*.mex</code> file is generated.

The perturbation types are selected by setting either:

Option	Description
<code>Option Exploration_method=external Block_name=Block_name</code>	The <code>Block_name</code> is the same as the name specified in the <code>.DATA</code> block; HSPICE sweeps the row content with the EXCommand (see the section EXCommand Option: Export Data Block Action).
<code>Option Ignore_exploration=yes no</code>	(Default=no) HSPICE ignores the content in the <code>design_exploration</code> block, when <code>Ignore_exploration=yes</code> .
<code>Option Secondary_param=yes no</code>	(Default = no) If <code>Secondary_param= yes</code> , HSPICE exports the MOSFET secondary instance parameters to a <code>*.mex</code> file (created when <code>option export=yes</code>), and also permits the secondary parameters to be imported as a column header in the <code>.DATA</code> block (<code>option export=no</code>). See the example, following this section.

Example: Option secondary_param=yes

```
.design_exploration
option secondary_param=yes
parameter asad = '1.5e-7'
option exploration_method=external block_name=dat2
*nmos snps20n design_area='(Get_E(L)+1u)*(Get_E(W)+0.8u)'
pmos snps20p as = 'asad*Get_E(W)' ad = '2*asad*Get_E(W)'
.data dat2
    index vdd@p k@p
    opamp.rcomp@r@e
    opamp.r0@r@e      r1@r@e r0@r@e
    opamp.ccomp@c@e c0@c@e c1@c@e
    opamp.mn1@snps20n@m@e
    opamp.mn1@snps20n@ad@e
    opamp.mn2@snps20n@m@e
    opamp.mn2@snps20n@ad@e
    opamp.mp3@snps20p@m@e
    opamp.mp4@snps20p@m@e
    opamp.mp5@snps20p@l@e      opamp.mp5@snps20p@w@e
    opamp.mp5@snps20p@m@e      opamp.mn8@snps20n@l@e
    opamp.mn8@snps20n@w@e      opamp.mn8@snps20n@m@e
    opamp.mn7@snps20n@l@e      opamp.mn7@snps20n@w@e
    opamp.mn7@snps20n@m@e      opamp.mn6@snps20n@l@e
    opamp.mn6@snps20n@w@e      opamp.mn6@snps20n@m@e
    v2@v@e
1.000      2.5000      2.0000      7.000e+03 1.000e+06 1.000e+06
1.000e+07 9.000e-13 1.000e-03 5.000e-12
4.0000      1.000e-08 4.0000      1.000e-08 4.0000
4.0000      4.000e-07 1.000e-05 3.0000      6.000e-06
3.600e-05 10.0000      6.000e-06 3.600e-05 4.0000
6.000e-06 3.600e-05 6.0000      0.
.enddata
.End_Design_Exploration
```

Notice that column header `opamp.mn1@snps20n@ad@e` can be recognized by HSPICE only if `Option Secondary_param=yes`. In the netlist for the opamp (not shown above), only the devices `mn1` and `mn2` have secondary element parameter `AD` defined.

The supported MOSFET secondary parameters are: `AS`, `AD`, `PS`, `PD`, `NRD`, `NRS`, `RDC`, `RSC`.

Parameters Section

Parameters can be defined here, which are used in subsequent definitions within the Exploration Block. The name space is separate from the netlist.

Parameters specified with numerical values are exported; derived parameters are not exported and are not available for exploration.

Secondary Element Parameters

To calculate secondary element parameters on a single device:

```
Element subcircuitName.ElementName parameterName=  
    'expression of parameterName'
```

The element parameters here include AD, AS, PD, PS, NRS, and NRD on the left side and expressions of L and W of the same element on the right hand side. For example:

```
Element opamp.mn1 AD='1e-7*Get_E(W)'
```

This relationship is enforced on all instantiations of subcircuit opamp (unless specifically excluded from exploration). Also, the property AD of opamp.mn1 is not exported, and it is not available for exploration.

Secondary Device Parameters

Expressions for calculating the values of secondary device parameters for all devices with a certain model can be defined. Default values for AD, AS, PD, PS, NRS and NRD are often specific for devices which share the same model, as a function of W and L.

```
ModelType ModelName instanceParameterName='expression of  
parameterName'
```

For example:

```
nmos snps65n as='asad*Get_E(W)' ad='asad*Get_E(W)'
```

This directive means that all nmos devices subject to exploration, with model snps65n, and have AS and/or AD specified, have their source and drain areas re-calculated by this equation prior to simulation. If the secondary parameter is not specified on the device, then it is not added.

Note that HSPICE simulation results can change when such a definition is added to the Exploration Block, if the original values for AS and AD are different from the values calculated using the expression. While the secondary parameters are not exported, they are available for exploration when defined in the data block (expressions are not supported):

```
Opamp.mn1 AD=1e-12
```

Same-Circuit Parameters

To force relationship between parameters of the same subcircuit, use the syntax

```
Subckt subcircuitName parameterName='expression of  
parameterName'
```

Note that this function supports only relationship within the same subcircuit.

Device Relationships

Relationships between element properties exist, which must be respected when changing device size. To reduce the amount of time required by the Exploration Tool to calculate these dependencies, such relationships can be defined directly in the Exploration Block.

To force a relationship between two different elements, use the syntax:

```
Element subcircuitName.ElementName parameterName=  
'expression of  
Get_E([subcircuitName.]ElementName@parameterName)'
```

The element parameter names here include W and L for NMOS and PMOS devices. The subcircuit name on the right side of the definition is optional, if it is the same as the one on the left side.

```
Element opamp.mn1 l='Get_E(mn2@l)'  
Element inv4.mp1 w='2*Get_E(inv2.mp1@w)'
```

These relationships are enforced on all instantiations of subcircuits opamp and inv4 (unless specifically excluded from exploration).

Property Relationships

Properties L of opamp.mn1 and W of inv4.mp1 are not exported, and are not available for exploration.

Derived Device Properties

Derived device properties, as defined in the Exploration Block, are not exported. While specifying device relationship in a direct way is not supported, you can do this through parameter transformations.

Element Relationship: This relationship is enforced on all instantiations of subcircuit opamp (unless specifically excluded from exploration). Also, the

property AD of `opamp.mn1` is not exported, and it is not available for exploration. The rules for the element relationship are:

1. If Element relationship equations are defined, then no related element parameters be exported in `*.mex` file, and such parameters do not appear in the `.DATA` block, as well.
2. If such parameters appear in a `.DATA` block, a warning message such as `**warning** unsupported statement c0@c@e will be ignored, but will be reported in the list file (HSPICE ignores this column in the .DATA block)`.
3. Considering the implementation complex, the 0809 release does not support netlist-defined subcircuits.
4. Such relationship equations can change element parameters no matter such element Parameters are defined through numbers or through expressions in the netlist.

Examples

Example 1:

```
Element opamp.mn1@L = `Get_E(opamp.mn2@L) '
```

Only `opamp.mn2@L` from the netlist is exported; here, the device property of `opamp.mn1@L` is not exported.

Example 2:

```
nmos nch ad='120n*Get_E(W) '  
+ as='120n*Get_E(W) '  
+ ps='240n+Get_E(W) '  
+ pd='240n+Get_E(W) '
```

The properties AD, AS, PS, and PD are not exported.

Get_E() Function In Exploration Block

The Exploration Block deals strictly with netlist values, not final values. So the `Get_E()` function returns the value that the user specifies in the netlist, and BEFORE adding any scale, `xw` and `wint` calculations. This is distinct from the `Get_E()` in Variation Block, which returns the effective values.

`Get_E()` is used to return a specific instance parameter value supported in the Exploration Block. The syntax is:

```
Get_E(SubcircuitName.InstanceName@ParameterName)
```


This function allows you to conveniently specify the relationship between instances. For example:

```
.design_exploration
  element op5.mn1 l='Get_E(op5.mn2@l) '
  element op5.mn1 w='Get_E(op5.mn2@w) '
.end_design_exploration
```

Parameters Defined Outside the Exploration Block

The parameters defined outside the Exploration Block can be referenced using the syntax:

```
Get_P(parameterName)
```

Area Measurement

While an exact area is only available after layout in integrated circuit design, following certain rules can provide a good estimate.

The complete measurement consists of three steps:

1. Calculate area of each device, according to model specific expressions
2. Calculate total area of top circuit or specified subcircuit.
3. Make results available to built-in measurement processor for output.

The calculation is performed as part of the operating point for AC and TRAN, but executed for each step of a DC transfer characteristics. This allows for reporting area at a certain value of a design parameter, which affects circuit area. However, area is not recalculated if it changes during an AC or transient sweep.

Syntax for device area calculation:

```
Modeltype Modelname design_area = expression
```

Example

```
nmos nch design_area='(Get_E(L)+1u)*(Get_E(W)+0.8u) '
```

The measurement syntax allows for reporting the area of the whole circuit or a subcircuit, and has the following structure:

```
.measure analysisType measName Function
+ DESIGN_AREA(HierarchicalName DeviceName)
```

Where, `analysisType` is DC|AC|Tran, Function can be min, max, find-at.

For example:

```
.measure top_area max design_area(x1)
```

Rules for Area Measurement using `design_area` Keyword

The following rules apply for area measurement for typical cases.

1. Area computation only supports resistors, capacitors, and MOSFETs, currently. There is no geometry parameter for L, so this element is ignored in area computation.
2. The `design_area` keyword only supports those instances in which W/L is defined explicitly in netlist.
3. Compute total circuit area:

```
.measure |dc|ac|tran output_name1 find design_area at=val
```

Hierarchically based, compute the sum area of subcircuit x1:

```
.measure |dc|ac|tran output_name2 find design_area(x1) at=val
```

Compute area of x1.mn1

```
.measure |dc|ac|tran output_name3 find design_area(x1.mn1)
      at=val
```

For the equations defining area inside `.design_exploration` block, only model specific expressions are currently supported:

```
Modeltype Modelname design_area = expression
```

4. The priority of computing one device area is
 - a. For resistors and capacitors:
 - (i) Expressions defined inside `exploration_block`
 - (ii) $W*L*M$ (W and L is defined as instance parameter)
 - (iii) $W_{model}*L_{model} *M$ (W_{model} and L_{model} are the model geometry values)
 - (iv) Otherwise, their area is zero.
 - b. For MOSFETs:
 - (i) Expressions defined inside `exploration_block`
 - (ii) $W*L*M$ (W and L are defined as instance parameters)

- (iii) `Wdefault*Ldefault *Mdefault` (`Wdefault` and `Ldefault` are the default geometry values for MOSFET)
 - (iv) `M` is the multiplier parameter.
5. If 'scale' is defined, then `design_area = area*scale*scale`
Such measurement works with the `.DC |.AC |.TRAN` command, whether `.design_exploration` block is defined or not.

Specifying Constraints

While working through the device relationships, designers may want to specify constraints in the Exploration Block. The IF-ELSE structure can be used to constrain the circuit topology in HSPICE only.

Each device subject to exploration is checked to verify whether the condition applies, and then the specified action or actions are executed.

Syntax

```
if (condition1){  
  statement_block1  
}
```

The following statement block is optional, and can be repeated multiple times:

```
elseif (condition2)  
{statement_block2  
}
```

The following statement block is optional, and cannot be repeated:

```
else{  
  statement_block3  
}
```

Guidelines for using IF-ELSE Blocks

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

- In an IF, ELSEIF, or ELSE statement block, you can reset certain elements geometries or issue a warning, or an error, to cause the simulator to abort. This example checks all MOSFETs that use model `nch`, if the length `l` is larger than `10u`, then a warning message is issued and the command resets it to `10u`.

```
If(nmos@nch@l>1e-5) {  
  Reset_parameter l=10u  
  Warning  
}
```

Chapter 28: Exploration Block

Exploration Block Structure

- The following functions can be used in specifying constraints:
 - < Relational operator (less than)
 - <= Relational operator (less than or equal to)
 - > Relational operator (greater than)
 - >= Relational operator (greater than or equal to)
 - && Logical AND
 - || Logical OR
- Supported CONDITION clauses include:

```
ModelType@ModelName@ModelParameter  
Subckt@SubcktName@SubcktParameter
```

When logical operators (&& and ||) are used, the ModelType, ModelName, Subckt, SubcktName within the condition clauses must be the same. Otherwise, an error message is issued and HSPICE aborts.

For example:

```
If (nmos@snps20n@w<2e-7 &&nmos@snps20n@l>10e-6) {  
    action1  
    action2  
}  
elseif (subckt@opamp@k>3 || subckt@opamp@n<2) {  
    action  
}
```

- You can include an unlimited number of ELSE-IF statements within an IF-ELSE block. If one element is found to satisfy the IF condition, HSPICE executes the action statements in the IF constraint for this element, and ignores any later ELSEIF or ELSE blocks.
- Commands for action statements are: `reset_parameter`, `warning`, and `abort`.
 - The `reset_parameter` statement can be specified several times in a constraint block. It can be used to limit or round off model or subcircuit parameters. For example:

```
If (nmos@nch@l>1e-5) {  
    reset_parameter l=10u  
    reset_parameter w=5u  
    ...  
    warning  
}
```

- An `abort` keyword would print a message to indicate that the condition was satisfied and terminate the simulation.
- The `warning` keyword does not abort the simulation, but generally remedies the condition. In the example below, a warning message is generated when the condition is met.

```
Warning: Condition "nmos@nch@l > 10u" encountered for  
device "DeviceName".
```

- Since the `ELSE` constraint block does not contain any `CONDITION` clauses, its action statements are executed only if no element satisfies the `CONDITION` in '`IF`' constraint.

The Processing of Netlist Parameters

As shown in the section [“Flow Using an External Exploration Tool,”](#) you can have a special case of passing parameter values down one level of hierarchy. In a general case, when HSPICE finds a parameter definition with numerical value (`.param paramName=value`), it is exported with its name and value in the appropriate section. Parameters which are defined with other parameters instead of numerical values, or expressions of other parameters and numerical values, are not included in the Export file. This preserves relationships between devices, which have been set up by the designer in the original netlist.

Example of `diffpair` in netlist:

```
.subckt diffamp in1 in2 out lpair=2u wpair=2u mpair=4  
  mn1 d1 in1 s b modelName l=lpair w=wpair m=mpair  
  mn2 d2 in2 s b modelName l=lpair w=wpair m=mpair  
  ....  
.ends diffamp
```

The subcircuit `diffamp` and its parameters `lpair`, `wpair`, and `mpair` will be in the Export file along with their local values. The devices `mn1` and `mn2` are not available for exploration.

Export File Syntax

HSPICE writes the extracted data from the circuit to a file with the extension `.mex?` with syntax similar to the `*.mcx?` file, which lists the perturbations created from the Variation Block content. The option settings are

reported first, followed by the names of all requested subcircuits and devices with their respective parameter names.

Separators are used as follows:

- A single period is used as a hierarchy separator between a subcircuit and an instance or device name, and as a separator between one or more subcircuit names, if their definitions are nested
- The @ character is used as a separator between model and parameter names.
- Additionally, identifiers are appended as follows to identify the proper owner if an element and a nested subcircuit have the same name:
 - E for element parameters
 - P for global parameters, and parameters used in subcircuit definitions and instantiations

Syntax Structure

The following constructs are provided:

- For primitives (R,C, without model):

[SubcircuitName.] InstanceName@ParamName@E

Example:

`Opamp.rbias@r@E`

- For devices with model (in NMOS, PMOS)

[SubcircuitName.] InstanceName@ModelType@ModelName@ParamName@E

Example:

`Opamp.mn1@snps65n@L@E`

- For standalone parameters:

[SubcircuitName@] =ParamName@P

Example:

`(.param factor) Opamp@factor@P`

- For parameters declared on subcircuit definition line:

[*SubcircuitName*@] *ParamName*@P

Example:

nand@wp@P

- For parameters appended to subcircuit instantiation:

[*SubcircuitName*.] *InstanceName*@*ParamName*@P

Example:

onebit.x2@wp@P

Whenever the optional *SubcircuitName* is not specified, the top level is assumed (implicit definition). For nested subcircuits, several *SubcircuitName* entries separated with a period are used.

Example Export File

```
onebit.x2@wp@P          onebit.x2@wn@P
onebit.x3@wp@P          onebit.x3@wn@P
opamp.mn1@snps65n@L@E  opamp.bias.rbias@r@E
diffamp@lpair@P        diffamp@wpair@P        diffamp@mpair@P
index1 value1 value2 value3 value4 value5 etc
```

If option `export=yes` is set, then the output file contains a single data set with the original design values from the netlist. If option `export=no` (or default), then one data set is written per exploration step with all the parameters suitable for exploration, not only the ones which were changed through an Exploration Data Block (see [Exploration Block Structure](#)).

Execution of Exploration in HSPICE

Exploration is considered a second sweep. The following syntax of the sweep with the data block command is used with `EXCommand` using the keyword `explore`, otherwise it has the same syntax as `MCCCommand` for Monte Carlo.

```
.DC|.TRAN|.AC analysisDetail sweep EXCommand
```

The sample number is optional (and ignored if specified) when data export is requested. The following table shows the tasks performed by the simulator with the different combinations of `EXCommand`, option `Export`, and Data Block definition (valid meaning here: defined and having at least one set). Simulation with relationships means that the relationships described in the section [Device Relationships](#) are enforced.

EXCommand Option: Export Data Block Action

EXCommand	Option Export	Data Block	Action
Ignored	Yes	ignored	Export and run simulation with original netlist
explore	No or undefined	valid	Simulate all sets with relationships in Data Block
explore=5	No or undefined	valid	Simulate sets 1 to 5 from Data Block, with relationships
explore list=3	No or undefined	valid	Simulate set 3 from Data Block, with relationships
Ignored	No or undefined	not defined	Simulate with relationships enabled

Exploration Data Block Syntax

The exploration tool output is returned back into HSPICE as a data block, which is referenced in the Exploration Block as “Exploration_Data”. Data blocks can contain, for example, information such as the variable `temper`, to enable temperature sweeps.

The content is as follows:

```
variableName1 variableName2 variableName3
index1 value11 value12 value13
index2 value21 value22 value23
.....
```

`index` is an integer, monotonically increasing.

It is sufficient here to include only the cumulative set of parameters which change for the exploration run. Parameter names and values not specified here are left at their original values.

Exploration and Variation Block Interactions

When an Exploration Block and Variation Block are both present, HSPICE can currently handle sweeps in up to two dimensions, but the following rules apply:

- There is a `monte` command only: Exploration Block is ignored
- There is an `explore` command only: Variation Block is ignored
- There is neither `monte` nor `explore` command: Variation and Exploration blocks ignored
- When there is both a `monte` and `explore` command and there is/are:
 - Single Monte Carlo sample specified: execute the Exploration Block content, according to the `explore` command and option settings.
 - Several Monte Carlo samples specified, single Exploration request: execute the requested Exploration Data set, with specified Monte Carlo samples.
 - Several Monte Carlo samples and several `explore` requests: abort, with appropriate message.
- For multiple Exploration Blocks:
 - Options are cumulative, the last definition prevails.
 - Only one named data block can be executed.

Limitations

The following feature is not implemented in this release.

Netlist Export

At the end of the exploration procedure, a valid netlist needs to be generated which reflects the final choices for the device sizes, in order to be able to drive a layout tool and run a successful LVS (layout versus schematic) verification.

Chapter 28: Exploration Block
Limitations

Optimization

Describes optimization in HSPICE for optimizing electrical yield.

HSPICE ships numerous examples for your use; and [Device Optimization Examples](#) for paths to demo files. See also [Cell Characterization Examples](#) for `.MODEL opt passfail` and `bisection` methods.

These topics are covered in the following sections:

- [Overview](#)
- [Optimization Statements](#)
- [Optimization Examples](#)

Overview

Optimization automatically generates model parameters and component values from a set of electrical specifications or measured data. When you define an optimization program and a circuit topology, HSPICE automatically selects the design components and model parameters to meet your DC, AC, and transient electrical specifications.

The circuit-result targets are part of the `.MEASURE` command structure and you use a `.MODEL` statement to set up the optimization.

Note: HSPICE uses post-processing output to compute the `.MEASURE` statements. If you set `INTERP=1` to reduce the post-processing output, the measurement results might contain interpolation errors. See the [HSPICE Reference Manual: Commands and Control Options](#) for more information about these options.

HSPICE employs an incremental optimization technique. This technique solves the DC parameters first, then the AC parameters, and finally the transient parameters. A set of optimizer measurement functions not only makes transistor optimization easy, but significantly improves cell and circuit optimization.

To perform optimization, create an input netlist file that specifies:

- Minimum and maximum parameter and component limits.
- Variable parameters and components.
- An initial estimate of the selected parameter and component values.
- Circuit performance goals or a model-versus-data error function.

If you provide the input netlist file, optimization specifications, component limits, and initial guess, then the optimizer reiterates the circuit simulation until it either meets the target electrical specification, or finds an optimized solution.

For improved optimization, reduced simulation time, and increased likelihood of a convergent solution, the initial estimate of component values should produce a circuit whose specifications are near those of the original target. This reduces the number of times the optimizer reselects component values and resimulates the circuit.

Optimization Control

How much time an optimization requires before it completes depends on:

- Number of iterations allowed.
- Relative input tolerance.
- Output tolerance.
- Gradient tolerance.

The default values are satisfactory for most applications. Generally, 10 to 30 iterations are sufficient to obtain accurate optimizations.

Simulation Accuracy

For optimization, set the simulator with tighter convergence options than normal. The following are suggested options.

For DC MOS model optimizations:

```
absmos=1e-8  
relmos=1e-5  
relv=1e-4
```

For DC JFET, BJT, and diode model optimizations:

```
absi=1e-10  
reli=1e-5  
relv=1e-4
```

For transient optimizations:

```
relv=1e-4  
relvar=1e-2
```

Curve Fit Optimization

Use optimization to curve-fit DC, AC, or transient data:

1. Use the `.DATA` statement to store the numeric data for curves in the data file as in-line data.
2. Use the `.PARAM xxx=OPTxxx` statement to specify the variable circuit components and the parameter values for the netlist. The optimization analysis statements use the `DATA` keyword to call the in-line data.
3. Use the `.MEASURE` statement to compare the simulation result to the values in the data file. In this statement, use the `ERR1` keyword to control the comparison.

If the calculated value is not within the error tolerances specified in the optimization model, HSPICE selects a new set of component values. HSPICE then simulates the circuit again and repeats this process until it obtains the closest fit to the curve or until the set of error tolerances is satisfied.

Goal Optimization

Goal optimization differs from curve-fit optimization because it usually optimizes only a particular electrical specification, such as rise time or power dissipation.

To specify goal optimizations, do the following:

1. Use the `GOAL` keyword.

2. In the `.MEASURE` statement, select a relational operator where `GOAL` is the target electrical specification to measure.

For example, you can choose a relational operator in multiple-constraint optimizations when the absolute accuracy of some criteria is less important than for others.

Timing Analysis

To analyze circuit timing violation, HSPICE uses a binary search algorithm. This algorithm generate a set of operational parameters, which produce a failure in the required behavior of the circuit. When a circuit timing failure occurs, you can identify a timing constraint, which can lead to a design guideline. Typical types of timing constraint violations include:

- Data setup time before a clock.
- Data hold time after a clock.
- Minimum pulse width required to allow a signal to propagate to the output.
- Maximum toggle frequency of the component(s).

Bisection Optimization finds the value of an input variable (target value) associated with a goal value for an output variable. To relate them, you can use various types of input and output variables, such as voltage, current, delay time, or gain, and a transfer function.

You can use the bisection feature in either a `PASSFAIL` mode or a bisection mode. In each case, the process is largely the same.

Optimization Statements

Optimization requires several statements:

- `.MODEL modname OPT ...`
- `.PARAM parameter=OPTxxx (init, min, max)`

Use `.PARAM` statements to define initial, lower, and upper bounds.

- A `.DC`, `.AC`, or `.TRAN` analysis statement, with:

`MODEL=modname`

`OPTIMIZE=OPTxxx`

RESULTS=*measurename*

Use .PRINT and .PROBE output statements, with the .DC, .AC, or .TRAN analysis statements.

Only use an analysis statement with the OPTIMIZE keyword for optimization. To generate output for the optimized circuit, specify another analysis statement (.DC, .AC, or .TRAN), and the output statements.

- .MEASURE *measurename* ... [GOAL=*rel* [*val*]]

Include a space on either side of the relational operator: = <space> <space>

For a description of the types of .MEASURE statements that you can use in optimization, see [Chapter 12, Simulation Output](#).

The proper specification order is:

1. Analysis statement with OPTIMIZE.
2. .MEASURE statements specifying optimization goals or error functions.
3. Ordinary analysis statement.
4. Output statements.

Optimizing Analysis (.DC, .TRAN, .AC)

The following syntax optimizes HSPICE simulation for a DC, AC, and Transient analysis.

```
.DC [DATA=filename] SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
.AC [DATA=filename] SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
.TRAN [DATA=filename] SWEEP OPTIMIZE=OPTxxx
+ RESULTS=ierr1 ... ierrn MODEL=optmod
```

Argument	Description
DATA	Specifies an in-line file of parameter data to use in optimization.
MODEL	The optimization reference name, which you also specify in the .MODEL optimization statement.

Argument	Description
OPTIMIZE	Indicates that the analysis is for optimization. Specifies the parameter reference name used in the .PARAM optimization statement. In a .PARAM optimization statements, if OPTIMIZE selects the parameter reference name, then the associated parameters vary during an optimization analysis.
RESULTS	The measurement reference name. You also specify this name in the .MEASURE optimization statement. RESULTS passes the analysis data to the .MEASURE optimization statement.

Optimization Examples

This section contains examples of HSPICE optimizations (for HSPICE RF optimization, see “Optimization” in the *HSPICE RF Manual*):

- [MOS Level 3 Model DC Optimization](#)
- [MOS Level 13 Model DC Optimization](#)
- [RC Network Optimization](#)
- [Optimizing CMOS Tristate Buffer](#)
- [BJT S-parameters Optimization](#)
- [BJT Model DC Optimization](#)
- [Optimizing GaAsFET Model DC](#)
- [Optimizing MOS Op-amp](#)

MOS Level 3 Model DC Optimization

This example shows an optimization of I-V data to a Level 3 MOS model. The data consists of gate curves (ids versus vgs) and drain curves (ids versus vds).

This example optimizes the Level 3 parameters:

- VTO
- GAMMA
- UO

- VMAX
- THETA
- KAPPA

After optimization, HSPICE compares the model to the data for the gate, and then to the drain curves. `.OPTION POST` generates waveform files for comparing the model to the data.

Input Netlist File for Level 3 Model DC Optimization

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/devopt/ml3opt.sp
```

The HSPICE input netlist shows:

- Using `.OPTION` to tighten tolerances, which increases the accuracy of the simulation. Use this method for I-V optimization.
- `.MODEL optmod OPT itropt=30` limits the number of iterations to 30.
- The circuit is one transistor. The `VDS`, `VGS`, and `VBS` parameter names, match names used in the data statements.
- `.PARAM` statements specify `XL`, `XW`, `TOX`, and `RSH` process variation parameters, as constants. The device characterizes these measured parameters.
- The model references parameters. In `GAMMA= GAMMA`, the left side is a Level 3 model parameter name; the right side is a `.PARAM` parameter name.
- The long `.PARAM` statement specifies initial, min and max values for the optimized parameters. Optimization initializes `UO` at 480, and maintains it within the range 400 to 1000.
- The first `.DC` statement indicates that:
 - Data is in the in-line `.DATA` all block, which contains merged gate and drain curve data.
 - Parameters that you declared as `OPT1` (in this example, all optimized parameters) are optimized.
 - The `COMP1` error function matches the name of a `.MEASURE` statement.
 - The `OPTMOD` model sets the iteration limit.

- The `.MEASURE` statement specifies least-squares relative error. HSPICE divides the difference between data `par(ids)` and model `i(m1)` by the larger of:
 - the absolute value of `par(ids)`, or
 - `minval=10e-6` If you use `minval`, low current data does not dominate the error.
- Use the remaining `.DC` and `.PRINT` statements for print-back after optimization. You can place them anywhere in the netlist input file because parsing the file correctly assigns them.
- The `.PARAM VDS=0 VGS=0 VBS=0 IDS=0` statements declare these data column names as parameters.

The `.DATA` statements contain data for `IDS` versus `VDS`, `VGS` and `VBS`. Select data that matches the model parameters to optimize.

Example

To optimize `GAMMA`, use data with back bias (`VBS= -2` in this case). To optimize `KAPPA`, the saturation region must contain data. In this example, the all data set contains:

- Gate curves: `vds=0.1 vbs=0,-2 vgs=1 to 5` in steps of 0.25.
- Drain curves: `vbs=0 vgs=2,3,4,5 vds=0.25 to 5` in steps of 0.25.

[Figure 167 on page 917](#) shows the results.

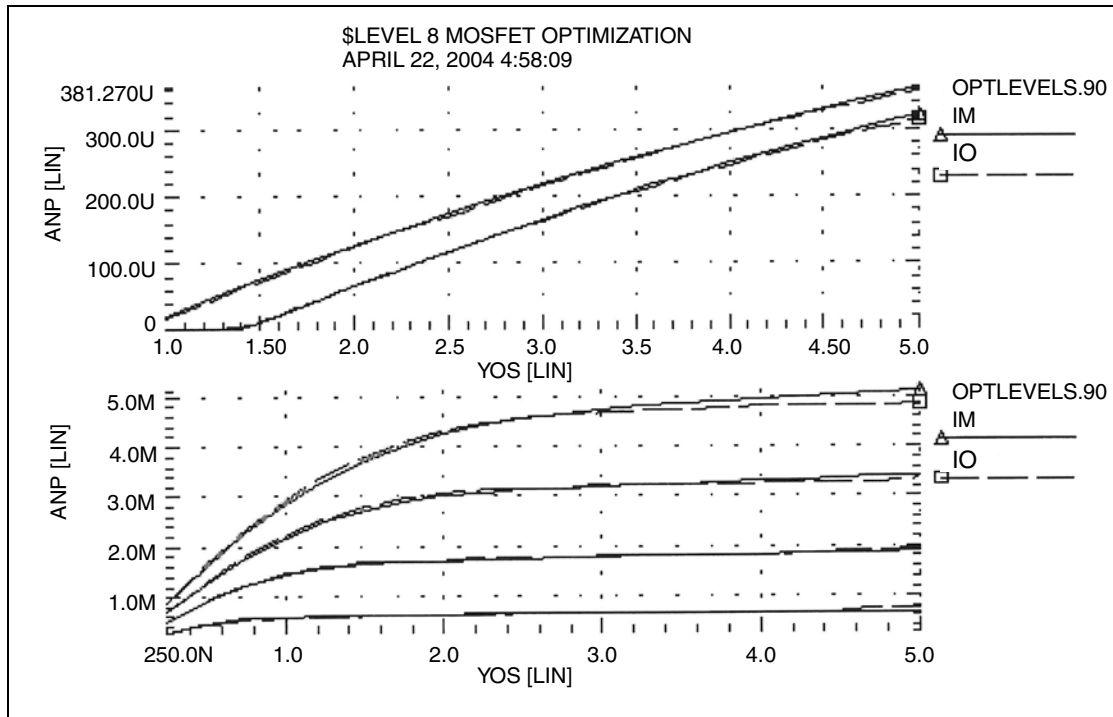


Figure 167 Level 3 MOSFET Optimization

MOS Level 13 Model DC Optimization

This example shows I-V data optimization to a Level 13 MOS model. The data consists of gate curves (i_{ds} versus v_{gs}) and drain curves (i_{ds} versus v_{ds}). This example demonstrates two-stage optimization.

1. HSPICE optimizes the v_{fb0} , k_1 , μ_{z} , x_{2m} , and u_{00} Level 13 parameters to the gate data.
2. HSPICE optimizes the μ_{S} , x_{3MS} , and U_1 Level 13 parameters, and the ALPHA impact ionization parameter to the drain data.

After optimization, HSPICE compares the model to the data. The `POST` option generates waveform files to compare the model to the data. [Figure 168 on page 918](#) shows the results.

DC Optimization Input Netlist File for Level 13 Model

You can find the sample netlist for this example in the following directory:
\$installdir/demo/hspice/mos/ml13opt.sp.

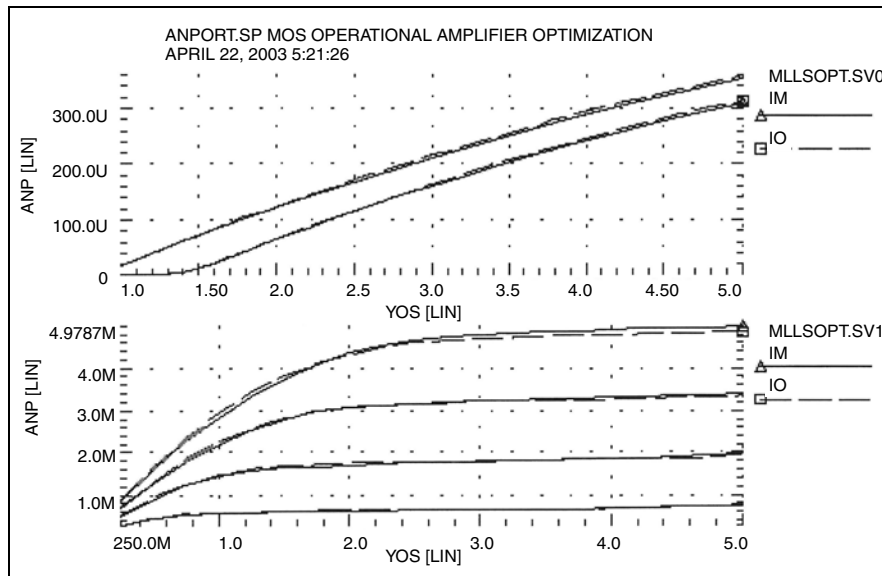


Figure 168 Level 13 MOSFET Optimization

RC Network Optimization

For a full demonstration example of optimizing speed or power for an RC circuit, follow the path to *rcopt.sp* in [Circuit Optimization Examples on page 993](#).

The following example optimizes the power dissipation and time constant for an RC network. The circuit is a parallel resistor and capacitor. Design targets are:

- 1 s time constant.
- 50 mW rms power dissipation through the resistor.

The HSPICE strategy is:

- RC1 .MEASURE calculates the RC time constant, where the GOAL of .3679 V corresponds to 1 s time constant e-rc.
- RC2 .MEASURE calculates the rms power, where the GOAL is 50 mW.
- OPT_{RC} identifies RX and CX as optimization parameters, and sets their starting, minimum, and maximum values.

Network optimization uses these HSPICE features:

- Measure voltages and report times that are subject to a goal.
- Measure device power dissipation subject to a goal.
- Measure statements replace the tabular or plot output.
- Parameters used as element values.
- Parameter optimizing function.
- Transient analysis with SWEEP optimizing.

Optimization Results

```
RESIDUAL SUM OF SQUARES      = 4.291583E-16
NORM OF THE GRADIENT         = 5.083346E-04
MARQUARDT SCALING PARAMETER  = 2.297208E-04
NO. OF FUNCTION EVALUATIONS = 20
NO. OF ITERATIONS           = 9
```

Residual Sum of Squares: The residual sum of squares is a measure of the total error. The smaller this value, the more accurate the optimization results.

$$\text{residual sum of squares} = \sum_{i=1}^{ne} E_i^2$$

In this equation, E is the error function, and ne is the number of error functions.

Norm of the Gradient: The norm of the gradient is another measure of the total error. The smaller this value is the more accurate the optimization results are. The following equations calculates the G gradient:

$$G_j = \sum_{i=1}^{ne} E_i \cdot (\Delta E_i / \Delta P_j)$$

Chapter 29: Optimization

Optimization Examples

$$\text{norm of the gradient} = 2 \cdot \sqrt{\sum_{j=1}^{np} G_j^2}$$

In this equation, P is the parameter, and np is the number of parameters to optimize.

Marquardt Scaling Parameter: The Levenburg-Marquardt algorithm uses this parameter to find the actual solution for the optimizing parameters. The search direction is a combination of the Steepest Descent method and the Gauss-Newton method.

The optimizer initially uses the Steepest Descent method as the fastest approach to the solution. It then uses the Gauss-Newton method to find the solution. During this process, the Marquardt Scaling Parameter becomes very small, but starts to increase again if the solution starts to deviate. If this happens, the optimizer chooses between the two methods to work toward the solution again.

If the optimizer does not attain the optimal solution, it prints both an error message, and a large Marquardt Scaling Parameter value.

Number of Function Evaluations: This is the number of analyses (for example, finite difference or central difference) needed to find a minimum of the function.

Number of Iterations: This is the number of iterations needed to find the optimized or actual solution.

Optimized Parameters OPTRC

```
.param rx= 7.4823 $ 55.6965 5.7945m  
.param cx=133.9934m $ 44.3035 5.1872m
```

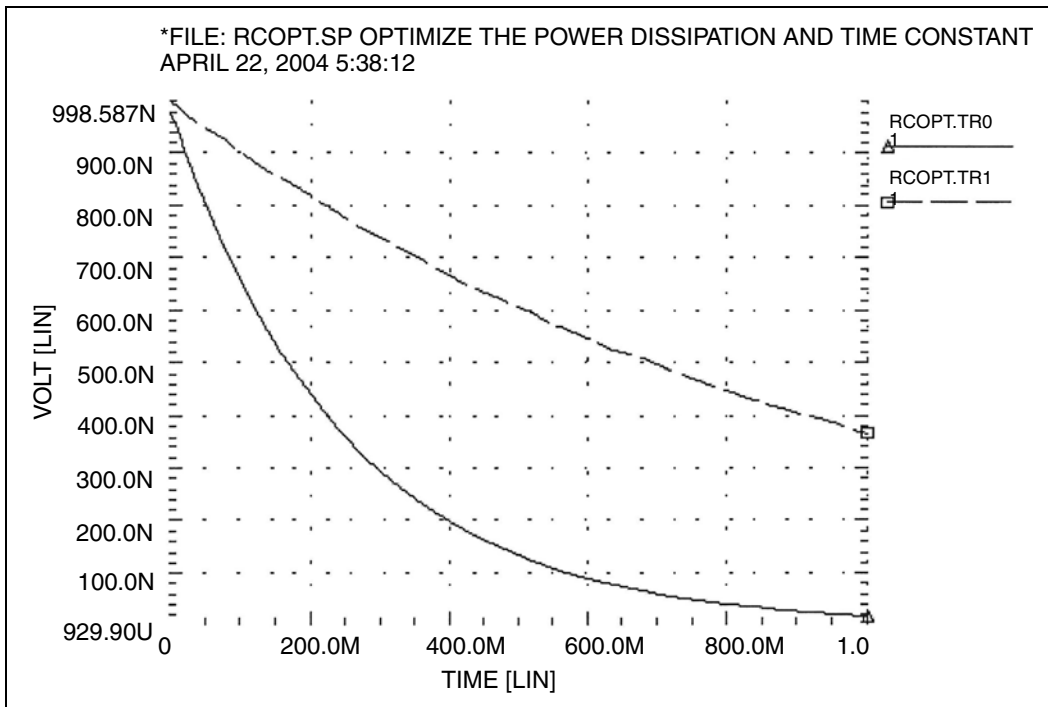


Figure 169 Power Dissipation and Time Constant (VOLT) RCOPT.TR0=Before Optimization, RCOPT.TR1=Optimized Result

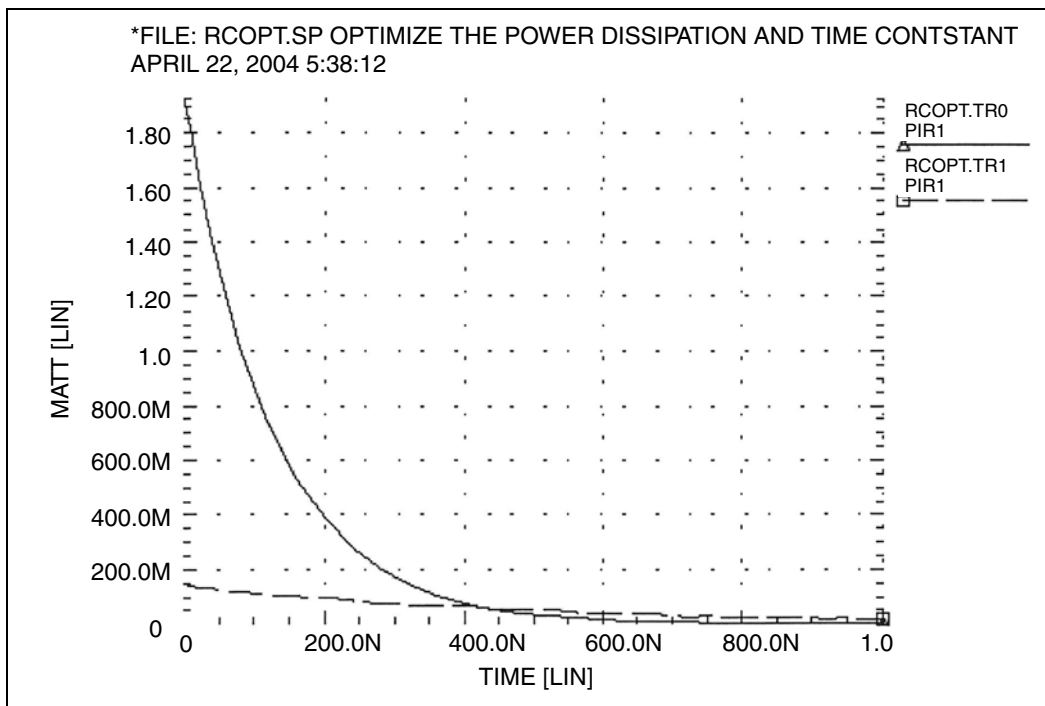


Figure 170 Power Dissipation and Time Constant (WATT) RCOPT.TR0=Before Optimization, RCOPT.TR1=Optimized Result

Optimizing CMOS Tristate Buffer

The example circuit is an inverting CMOS tristate buffer. The design targets are:

- Rising edge delay of 5 ns (input 50 percent voltage to output 50 percent voltage).
- Falling edge delay of 5 ns (input 50 percent voltage to output 50 percent voltage).
- RMS power dissipation should be as low as possible.
- Output load consists of:
 - pad capacitance
 - leadframe inductance
 - 50 pF capacitive load

The HSPICE strategy is:

- Simultaneously optimize both the rising and falling delay buffer.
- Set up the internal power supplies, and the tristate enable as global nodes.
- Optimize all device widths except:
 - Initial inverter (assumed to be standard size).
 - Tristate inverter and part of the tristate control (optimizing is not sensitive to this path).
- Perform an initial transient analysis for plotting purposes. Then optimize and perform a final transient analysis for plotting.
- To use a weighted RMS power measure, specify unrealistically low power goals. Then use MINVAL to attenuate the error.

Input Netlist File to Optimize a CMOS Tristate Buffer

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/apps/trist_buf_opt.sp
```

Chapter 29: Optimization
 Optimization Examples

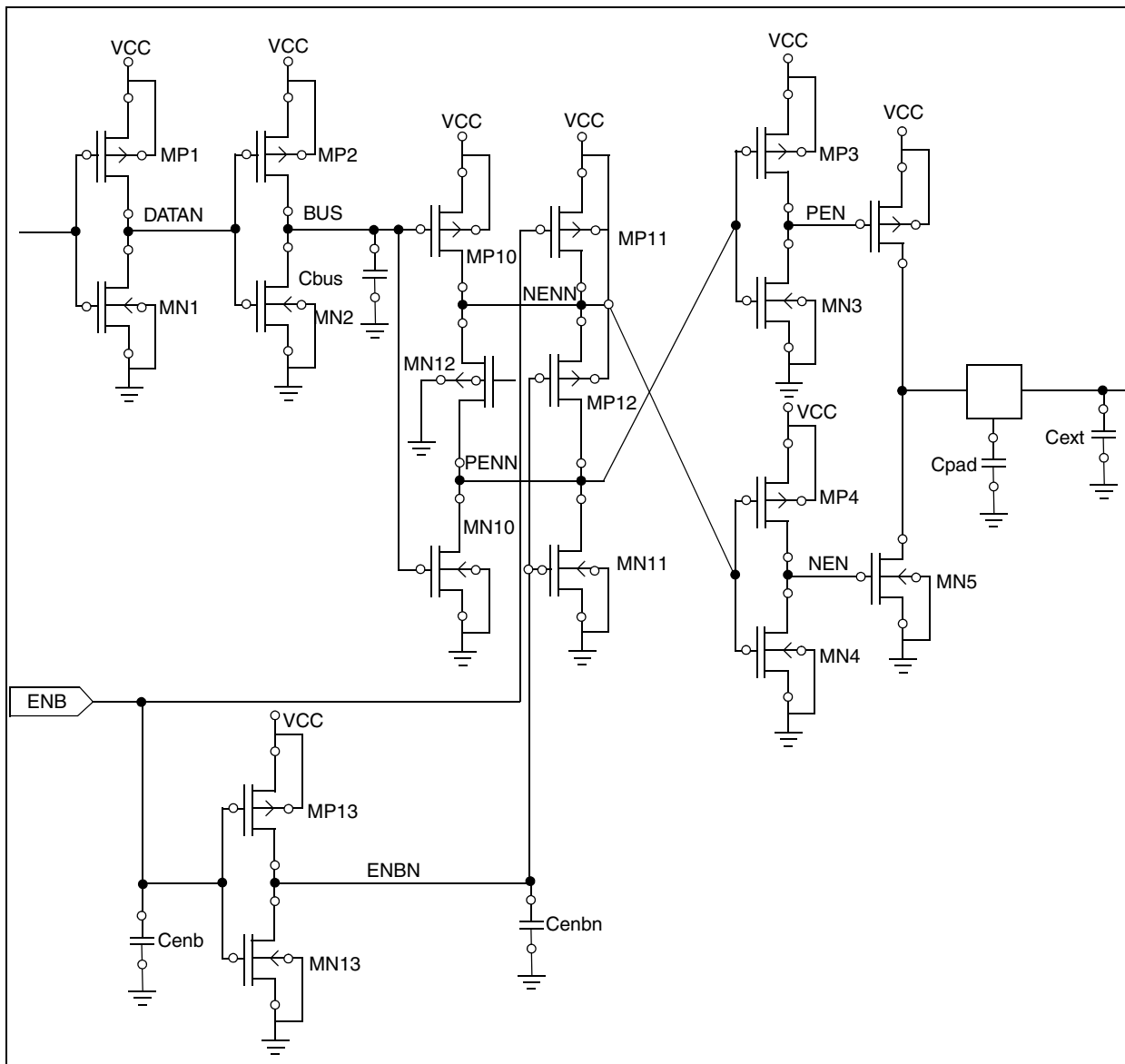


Figure 171 Tristate Buffer Optimization Circuit

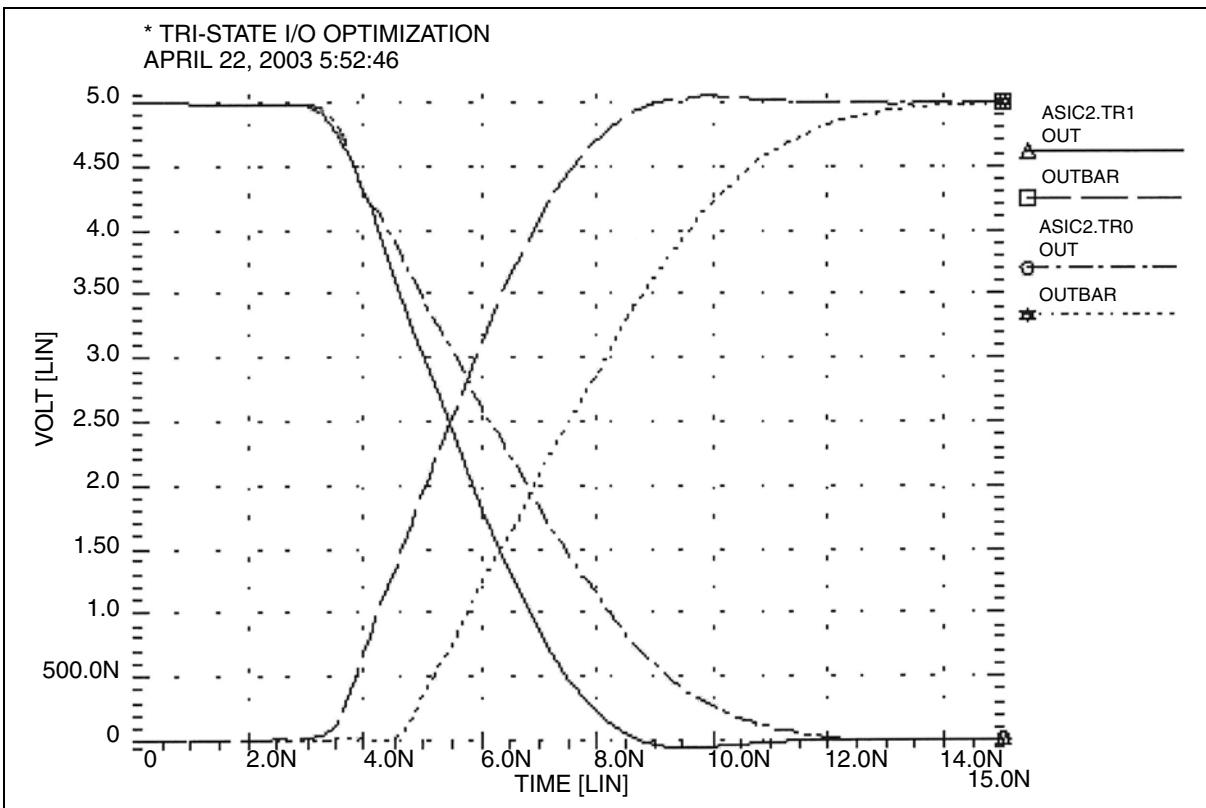


Figure 172 Tristate Input/Output Optimization ACIC2B.TR0 = Before Optimization, ACIC2B.TR1=Optimized Result

BJT S-parameters Optimization

The following example optimizes the S-parameters to match those specified for a set of measurements. The .DATA measured in-line data statement contains these measured S-parameters as a function of frequency. The model parameters of the microwave transistor (LBB, LCC, LEE, TF, CBE, CBC, RB, RE, RC, and IS) vary. As a result, the measured S-parameters (in the .DATA statement) match the calculated S-parameters from the simulation results.

This optimization uses a 2n6604 microwave transistor, and an equivalent circuit that consists of a BJT, with parasitic resistances and inductances. The BJT is biased at a 10 mA collector current (0.1 mA base current at DC bias and $bf=100$).

Key HSPICE Features Used

- .NET command to simulate network analyzer action.
- .AC optimization.
- Optimized element and model parameters.
- Optimizing, compares measured S-parameters to calculated parameters.
- S-parameters used in magnitude and phase (real and imaginary available).
- Weighting of data-driven frequency versus S Parameter table. Used for the phase domain.

Input Netlist File for Optimizing BJT S-parameters

BJT Equivalent Circuit Input

Use the bjtopt.sp netlist file located in your $\$installdir/demo/hspice/devopt$ directory for optimizing BJT S-parameters.

Optimization Results

```
RESIDUAL SUM OF SQUARES      =5.142639e-02
NORM OF THE GRADIENT         =6.068882e-02
MARQUARDT SCALING PARAMETER=0.340303
CO. OF FUNCTION EVALUATIONS=170
NO. OF ITERATIONS           =35
```

The maximum number of iterations (25) was exceeded. However, the results probably are accurate. Increase ITROPT accordingly.

```
Optimized Parameters OPT1- Final Values
***OPTIMIZED PARAMETERS OPT1 SENS %NORM-SEN
.PARAM LBB = 1.5834N $ 27.3566X 2.4368
.PARAM LCC = 2.1334N $ 12.5835X 1.5138
.PARAM LEE =723.0995P $254.2312X 12.3262
.PARAM TF =12.7611P $ 7.4344G 10.0532
.PARAM CBE =620.5195F $ 23.0855G 1.5300
.PARAM CBC = 1.0263P $346.0167G 44.5016
.PARAM RB = 2.0582 $ 12.8257M 2.3084
.PARAM RE =869.8714M $ 66.8123M 4.5597
.PARAM RC =54.2262 $ 3.1427M 20.7359
.PARAM IS =99.9900P $ 3.6533X 34.4463M
```

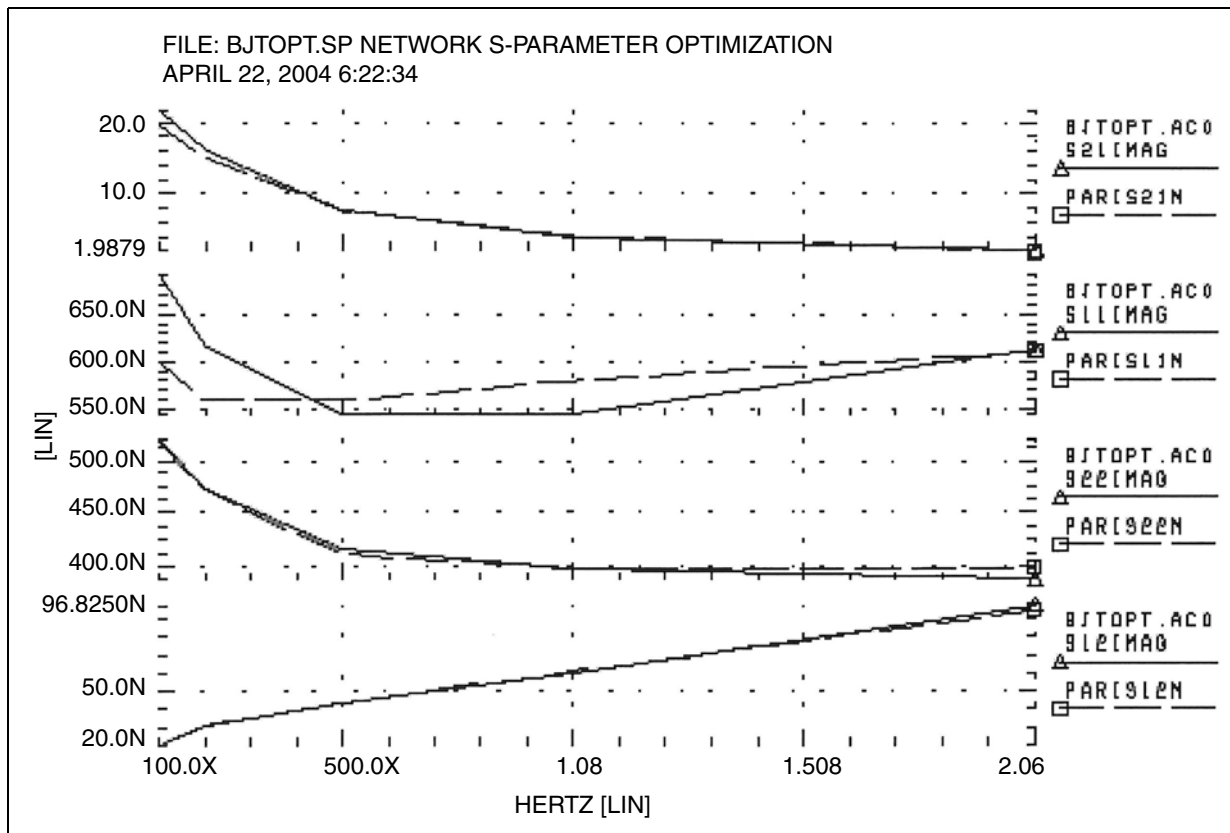


Figure 173 BJT-S Parameter Optimization

BJT Model DC Optimization

The goal is to match forward and reverse Gummel plots obtained from a HP4145 semiconductor analyzer by using the HSPICE `LEVEL=1` Gummel-Poon BJT model. Because Gummel plots are at low base currents, HSPICE does not optimize the base resistance. HSPICE also does not optimize forward and reverse Early voltages (`VAF` and `VAR`) because simulation does not measure VCE data.

Chapter 29: Optimization

Optimization Examples

The key feature in this optimization is incremental optimization.

1. HSPICE first optimizes the forward-Gummel data points.
2. HSPICE updates forward-optimized parameters into the model. After updating, you cannot change these parameters.
3. HSPICE next optimizes the reverse-Gummel data points.

BJT Model DC Optimization Input Netlist File

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/devopt/opt_bjt.sp
```

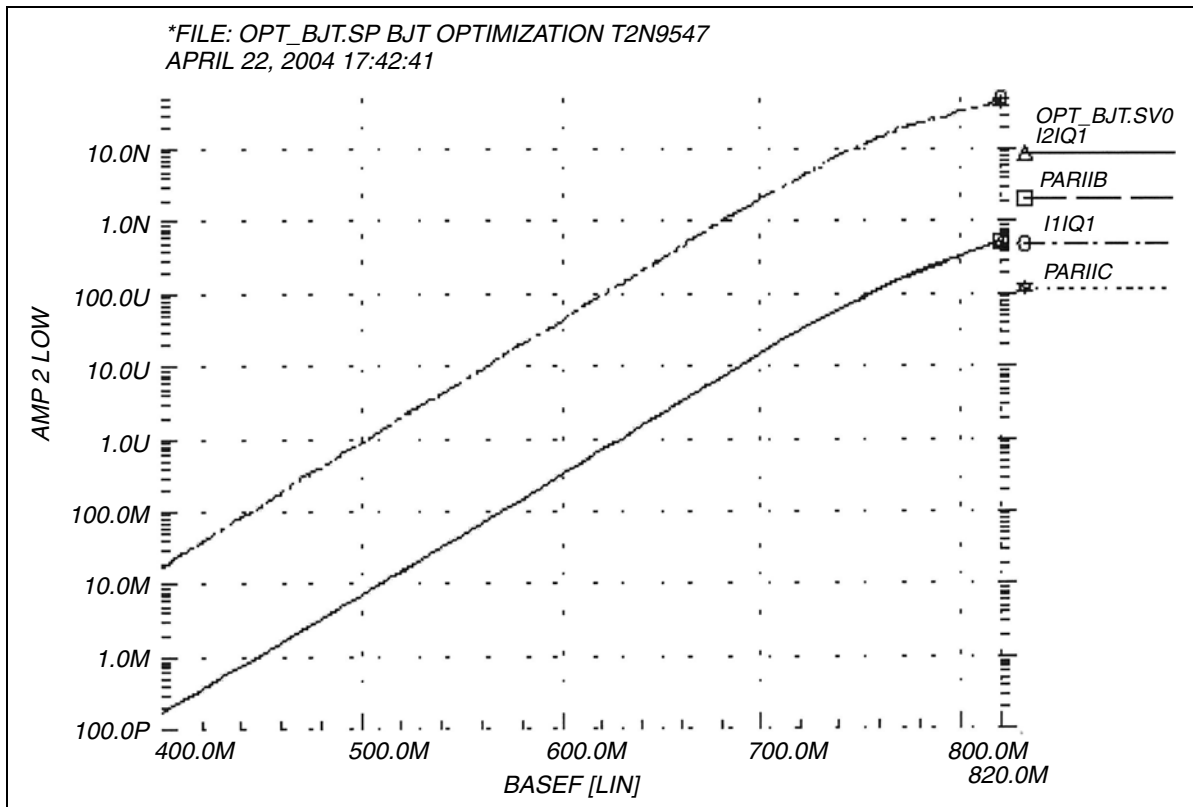


Figure 174 BJT Optimization Forward Gummel Plots

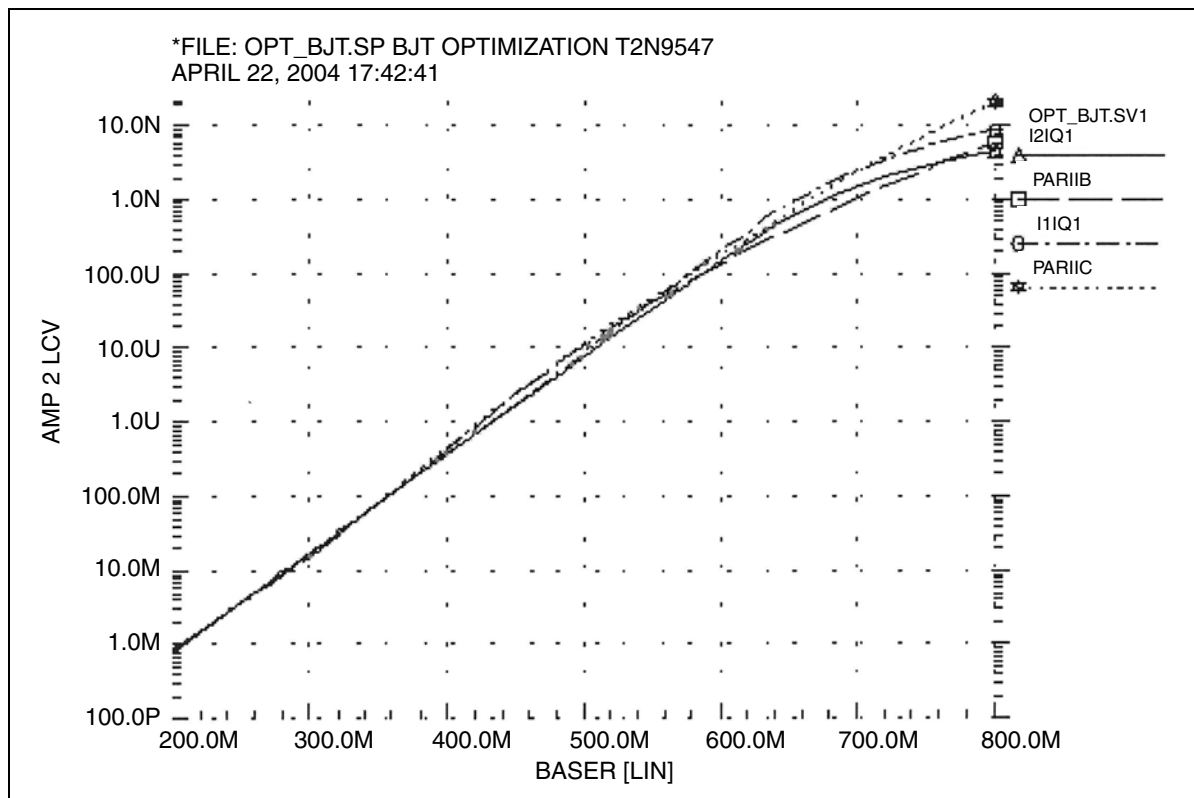


Figure 175 BJT Optimization Reverse Gummel Plots

Optimizing GaAsFET Model DC

This example circuit is a high-performance, GaAsFET transistor. The design target is to match HP4145 DC measured data to the HSPICE LEVEL=3 JFET model.

The HSPICE strategy is:

- .MEASURE IDSERR is an ERR1 type function. It provides linear attenuation of the error results starting at 20 mA. This function ignores all currents below 1 mA. The high-current fit is the most important for this model.
- The OPT1 function simultaneously optimizes all DC parameters.

Chapter 29: Optimization

Optimization Examples

- The .DATA statement merges TD1.dat and TD2.dat data files.
- The graph plot model sets the MONO=1 parameter to remove the retrace lines from the family of curves.

GaAsFET Model DC Optimization Input Netlist File

You can find the sample netlist for this example in the following directory:
\$installdir/demo/hspice/devopt/jopt.sp

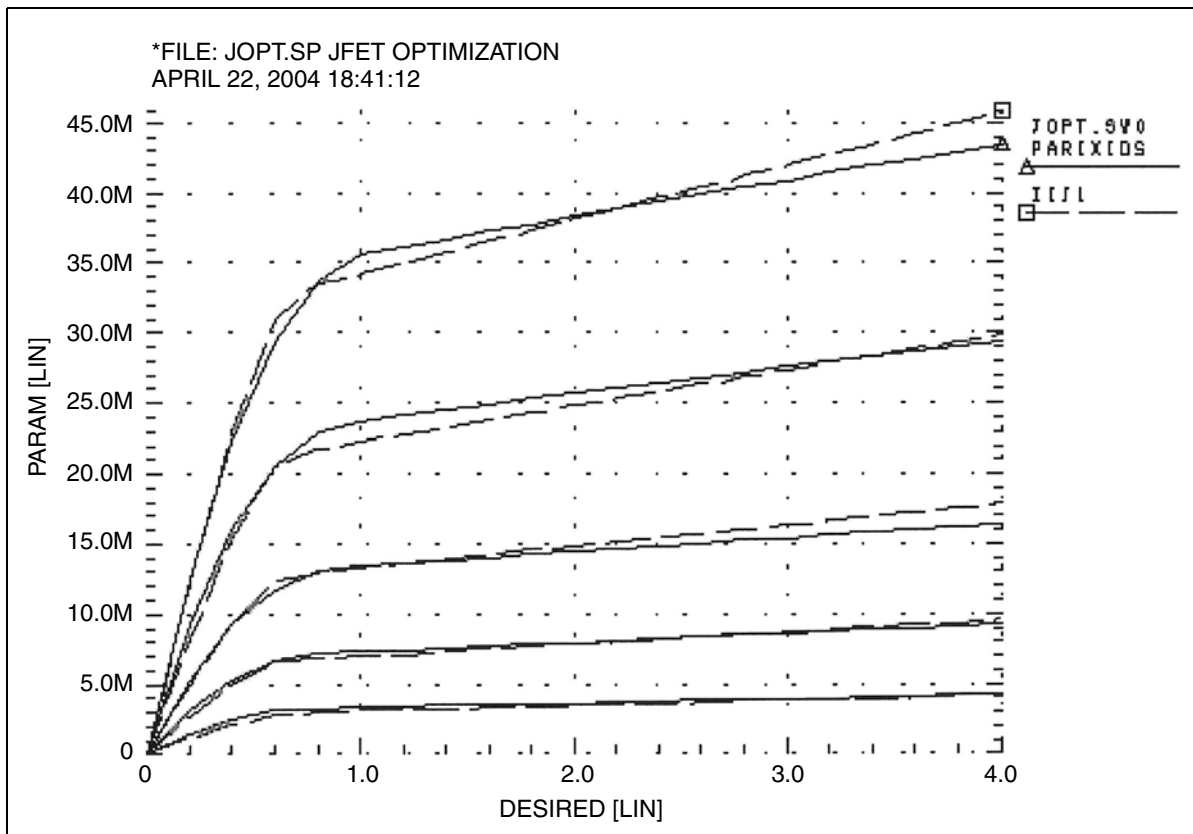


Figure 176 JFET Optimization

Optimizing MOS Op-amp

The design goals for the MOS operational amplifier are:

- Minimize the gate area (and therefore the total cell area).
- Minimize the power dissipation.
- Open-loop transient step response of 100 ns for rising and falling edges.

The HSPICE strategy is:

- Simultaneously optimize two amplifier cells for rising and falling edges.
- Total power is power for two cells.
- The optimization transient analysis must be longer to allow for a range of values in intermediate results.
- All transistor widths and lengths are optimized.
- Calculate the transistor area algebraically use a voltage value and minimize the resulting voltage.
- The transistor area measure statement uses `MINVAL`, which assigns less weight to the area minimization.
- Optimizes the bias voltage.

Example: MOS Op-amp Optimization Input Netlist File

You can find the sample netlist for this example in the following directory:
\$installdir/demo/hspice/ciropt/ampopt.sp

Chapter 29: Optimization
Optimization Examples

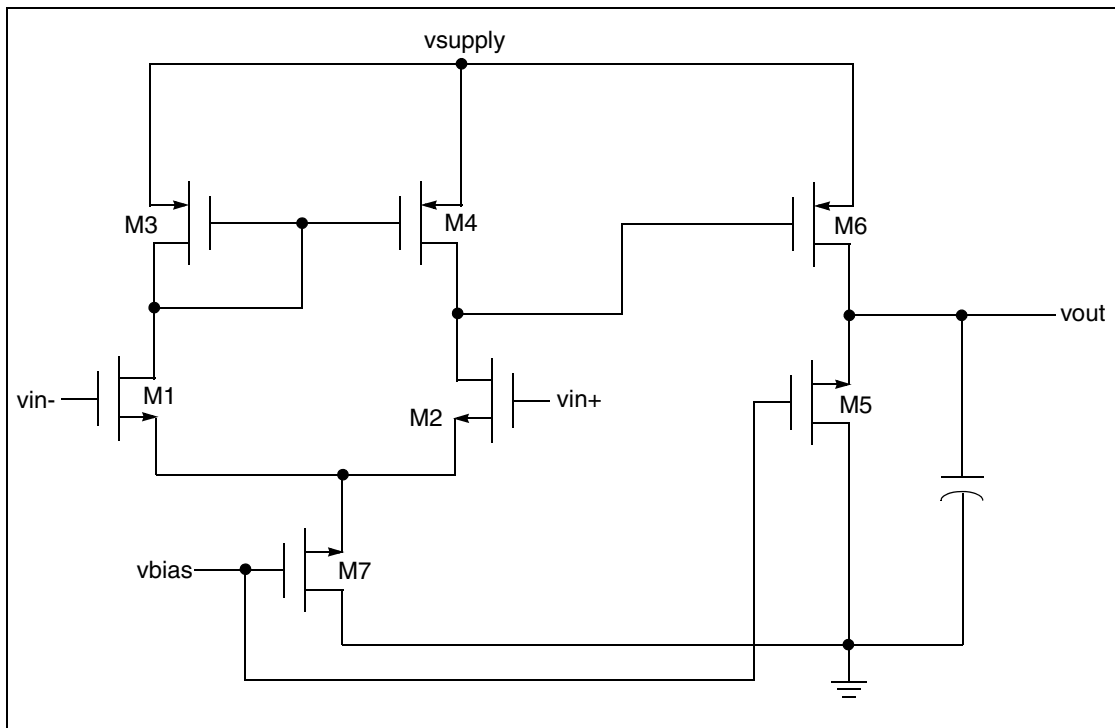


Figure 177 CMOS Op-amp

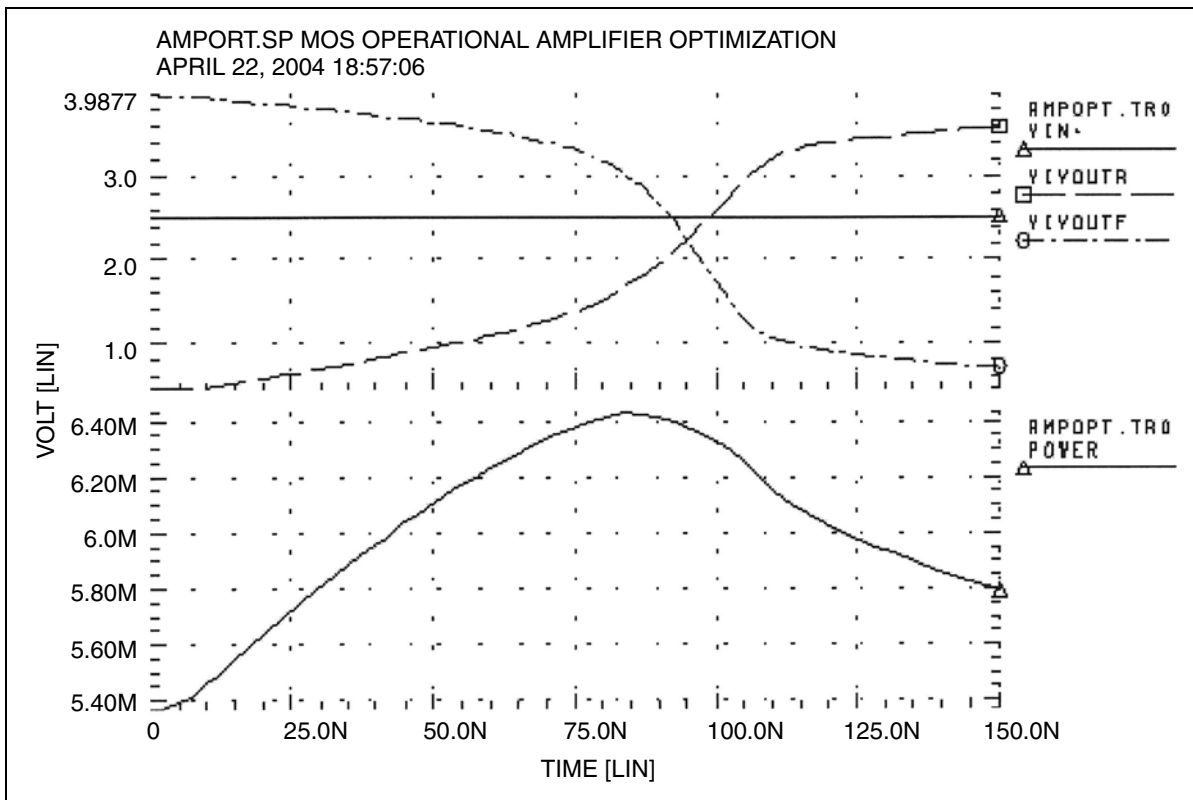


Figure 178 Operational Amplifier Optimization

Chapter 29: Optimization
Optimization Examples

Timing Analysis Using Bisection

Describes how to use the bisection function in timing optimization.

To analyze circuit timing violations, a typical methodology is to generate a set of operational parameters that produce a failure in the required behavior of the circuit. When a circuit timing failure occurs, you can identify a timing constraint, which can lead to a design guideline. You must perform an iterative analysis to define the violation specification.

Typical types of timing constraint violations include:

- Data setup time, before the clock
- Data hold time, after the clock
- Minimum pulse width required for a signal to propagate to the output
- Maximum toggle frequency of the component(s)

HSPICE ships numerous examples for your use; see [Bisection-Timing Analysis Examples](#) for paths to demo files.

For more information about optimization, see [Chapter 29, Optimization](#) in this user guide.

For information on Monte Carlo, see [Chapter 22, Monte Carlo—Traditional Flow Statistical Analysis](#) and [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#).

This chapter discusses the following topics.

- [Overview of Bisection](#)
- [Bisection Methodology](#)
- [Using Bisection](#)
- [Setup Time Analysis](#)
- [Minimum Pulse Width Analysis](#)

- [Pushout Bisection Methodology](#)
- [Using RELOUT and RELIN to Affect HSPICE Bisection Optimization](#)
- [Using Bisection with Monte Carlo Analysis](#)
- [Bisection Optimization Application Example](#)

Overview of Bisection

Before bisection methods were developed, engineers built external drivers to submit multiple parameterized simulations to SPICE-type simulators. Each simulation explored a region of the operating envelope for the circuit. To provide part of the analysis, the driver also post-processed the simulation results, to deduce the limiting conditions.

If you characterize small circuits this way analysis times are relatively small, compared with the overall job time. This method is inefficient, due to overhead of submitting the job, reading and checking the netlist, and setting up the matrix. The newer bisection methods increase efficiency when you analyze timing violations, to find the causes of timing failure. Bisection optimization is an efficient cell-characterization method, in Synopsys HSPICE.

For a full demo example of finding early, optimal, and late setup times of a DFF, follow the path to `fig26_4.sp` in [Bisection-Timing Analysis Examples](#) in this user guide.

The bisection methodology saves time in three ways:

- Reduces multiple jobs to a single characterization job.
- Removes post-processing requirements.
- Uses accuracy-driven iterations.

[Figure 179 on page 937](#) shows a typical analysis of setup-time constraints. Clock and data input waveforms drive a cell. Two input transitions (rise and fall) occur at times T_1 and T_2 . The result is an output transition, when $V(\text{out})$ changes from low to high. The following relationship between the $T_1(\text{data})$ and $T_2(\text{clock})$ times must be true for the $V(\text{out})$ transition to occur: $T_2 > (T_1 + \text{setup time})$.

Characterization or violation analysis determines the setup time. To do this, HSPICE keeps T_2 fixed and repeats the simulation with different T_1 values. It then observes which T_1 values produce an output transition and which do not.

Before bisection, users had to run tight sweeps of the delay between the data setup and clock edge, and look for the value at which no transition occurs. To do this, you swept a value that specifies how far the data edge precedes a fixed clock edge. This method is time consuming, and is accurate only if the sweep step is very small. Linear search methods cannot accurately determine the setup time value, unless you use extremely small steps from T_1 to T_2 to simulate the circuit at each point, and monitor the outcome.

For example, even if you know that the desired transition occurs during a particular 5 ns period, you might need to run 50 simulations to search for the setup time to within 0.1 ns over that 5 ns period. But the error in the result can be as large as 0.05 ns.

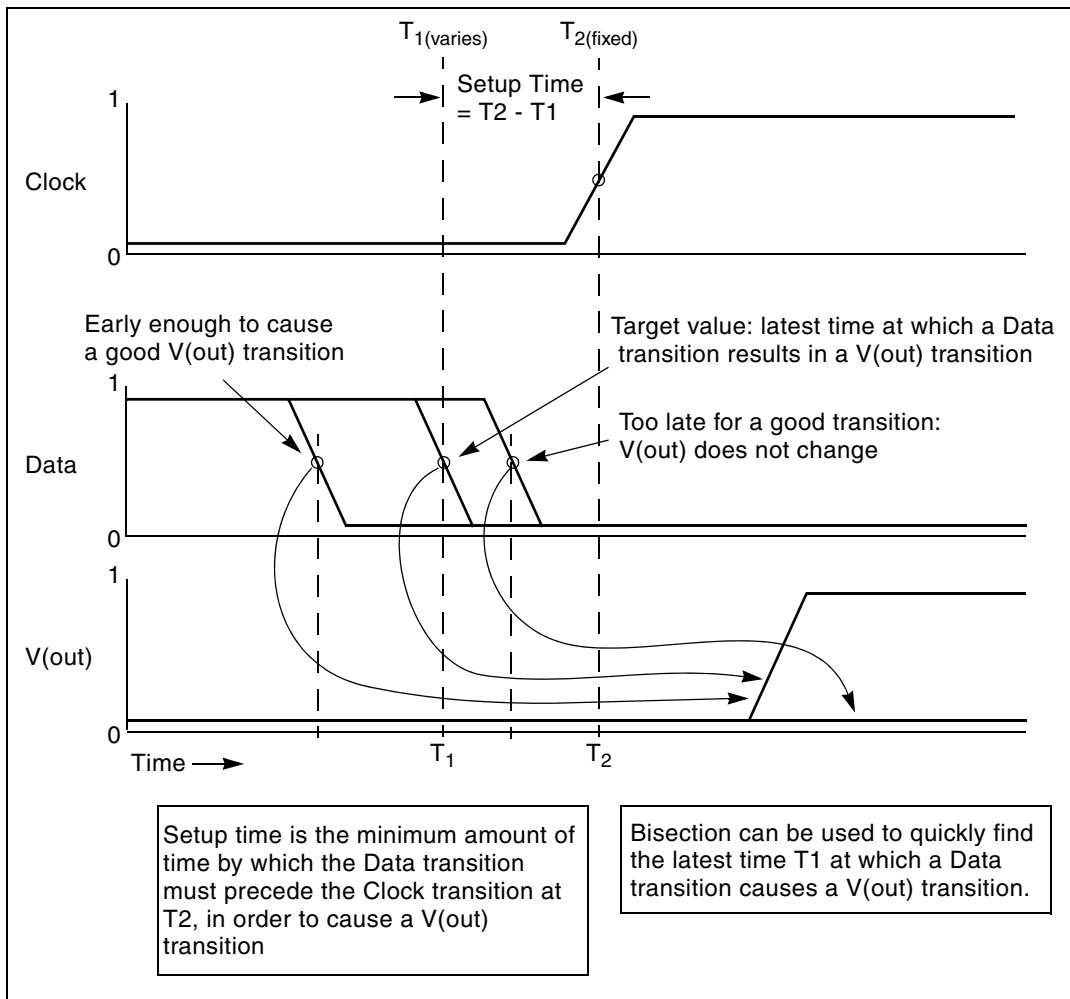


Figure 179 Determining Setup Time with Bisection Violation Analysis

The bisection feature greatly reduces the amount of work and computational time required to find an accurate solution for this type of problem. The following sections show examples of using this feature to identify timing violations for the setup, hold, and minimum clock pulse width.

Bisection Methodology

Bisection is an optimization method that uses a binary search method, to find the value of an input variable (target value). This variable is associated with a goal value of an output variable.

The type of the input and output variables can be voltage, current, delay time, or gain, related by some transfer function. In general, use a binary search to locate the goal value of the output variable within a search range of the input variable. Then, iteratively halve that range to rapidly converge on the target value. At each iteration, HSPICE compares the measured value of the output variable with the goal value. Both the PASSFAIL method and the bisection method use bisection (see [Using Bisection](#)).

The bisection procedure consists of two measurement and optimization steps, when solving the timing violation problem:

- Detecting whether the output transition occurred.
- Automatically varying the input parameter (T_1 in [Figure 179 on page 937](#)) to find the value for which the transition barely occurs.

Measurement

Use the `MAX` measurement function to detect the success or failure of an output transition. For a low-to-high output transition, a `MAX` measurement produces zero on failure, or approximately the V_{dd} supply voltage on success. This measurement, using a goal of V_{dd} (minus a suitable small value to ensure a solution), is sufficient to drive the optimization.

Optimization

The bisection method is straightforward if you specify a single measurement with a goal, and known upper and lower boundary values for the input

parameter. The characterization engineer must specify acceptable upper and lower boundary values.

Using Bisection

Before you can use bisection, you must specify the following:

- A pair of values, for the upper and lower boundaries of the input variables. To find a solution, one of these values must result in an output variable $>|\text{goal value}|$ and the other must result in $<|\text{goal value}|$.
- A goal value. If there is no goal keyword in the statement, the goal value will not be defaulted to zero, and HSPICE considers the measure result as a relative error expression.
- Error tolerance value. The bisection process stops when the difference between successive test values is \leq error tolerance. If the other criteria are also met, see the following steps.
- Related variables. Use a monotonic transfer function to relate variables where a steadily progressing time (increase or decrease) results in a single occurrence of the goal value at the target input variable value.

HSPICE includes the error tolerance in a relation, used as a process-termination criterion.

[Figure 180 on page 944](#) shows an example of the binary search process that the bisection algorithm uses. This example is the pass/fail type, and is appropriate for a setup-time analysis that tests for the presence of an output transition. In the example depicted in [Figure 179 on page 937](#) note that:

1. A long setup time $TS (= T2 - T1)$ results in a V_{OUT} transition (a pass).
2. A too-short setup time (where the latch has not stabilized the input data, before the clock transition) results in a fail.

Explanation: For example, you might define a pass time value as any setup time, TS , that produces a V_{OUT} output minimum high logic output level of 2.7V, which is the goal value.

3. The target value is a setup time that just produces the V_{OUT} value of 2.7V. Finding the exact value is impractical, if not impossible, so you need to specify an error tolerance to calculate a solution arbitrarily close to the target value.

4. The bisection algorithm performs tests for each specified boundary value to determine the direction in which to pursue the target value, after the first bisection. In this example shown in [Figure 179 on page 937](#), the upper boundary has a pass value and the lower boundary has a fail value.
5. To start the binary search you specify the lower and upper boundaries. The program tests the point midway between the lower and upper boundaries (see [Figure 180 on page 944](#)).
 - If the initial value passes the test, the target value must be less than the tested value (in this example). The bisection algorithm moves the upper search limit to the value that it just tested.
 - If the test fails, the target value must be greater than the tested value. Bisection moves the lower limit to the value that it just tested.
6. The algorithm tests a value midway between the new limits.
7. The search continues in this manner, moving one limit or the other to the last midpoint, and testing the value midway between the new limits.
8. The process stops when the difference between the latest test values is less than or equal to the error tolerance that you specified. To normalize this value, multiply by the initial boundary range.

For more information about using the `.MODEL` statement for bisection, see [.MODEL](#) in the *HSPICE Reference Manual: Commands and Control Options*.

For the path to a full demo file example of a DFF bisection search for setup time, `dff_top.sp`, see [Bisection-Timing Analysis Examples](#) in this user guide.

Examining the Command Syntax

The following syntax is used for bisection:

```
.MODEL OptModelName OPT METHOD=BISECTION ...
```

-or-

```
.MODEL OptModelName OPT METHOD=PASSFAIL .....
```

`OptModelName` is the model to be used. Refer to the [Optimization Examples](#) in Chapter 24 of this user guide for name information on specifying optimization models in HSPICE. The `METHOD` keyword specifies which optimization method to use. The `OPT` keyword indicates that optimization is to be performed.

For bisection, the method can be one of the following:

- BISECTION

When the difference between the two latest test input values is within the error tolerance and the latest measured value exceeds the goal, bisection has succeeded and then ends. This process reports the optimized parameter that corresponded to the test value that satisfies this error tolerance and this goal (passes). See [Bisection Optimization Application Example on page 961](#).

- PASSFAIL

When the difference between the last two optimization parameter test values is < the error tolerance and the associated goal measurement fails for one of the values and passes for the other, bisection has succeeded and then ends. The process reports the optimization parameter test value associated with the last passing measurement. “Pass” is defined as a condition in which the associated goal measurement can produce a valid result. “Fail” is defined as a condition in which the associated goal measurement is unable to produce a valid result. For example, if the measurement is of TRIG/TARG form, and the TARG event is not found, then this optimization parameter test value is deemed a failure. When using PUSHOUT bisection, the definition of a failure is modified to also include any goal measurement result that is valid and > the push-out specification.

You can also monitor multiple measurement results and find the parameter value at which all measurements begin to succeed. For example:

```
.tran 1p 100n sweep optimize=opt1 result=delq,delqn model=optmod
```

The parameters are passed in a normal optimization specification:

```
.PARAM ParamName=OptParFun (Initial, Lower, Upper)
```

In the BISECTION method, the measure results for *Lower* and *Upper* limits of *ParamName* must be on opposite sides of the goal value in the `.MEASURE` statement. In the PASSFAIL method, the measure must pass for one limit and fail for the other limit. The process ignores the value of the *Initial* field. The error tolerance is a parameter in the model which is being optimized. Using the BISECTION method, a bisectional search is applied to multiple parameters. The logical relationship of these parameters is based on 'AND'. In the PASSFAIL method, a bisectional search is applied to only one parameter.

When the OPTLST option is set (`.OPTION OPTLST=1`), the process outputs the following information for the BISECTION method:

```
bisec-opt iter = num_iterations xlo = low_val xhi = high_val  
x = result_low_val xnew = result_high_val err = error_tolerance
```

The x is the old parameter value and x_{new} is the new parameter value.

When `.OPTION OPTLST=1`, the process outputs the following information for the `PASSFAIL` method:

```
bisec-opt iter = num_iterations xlo = low_val xhi = high_val x =  
result_low_val xnew = result_high_val measfail = 1
```

In this syntax, `measfail=0` for a test failure for the x value.

Performing Transient Analyses with Bisections

When performing transient analysis bisection with the `.TRAN` statement, use the following syntax:

```
.TRAN TranStep TranTime SWEEP OPTIMIZE=OptParFun  
+ RESULTS=MeasureNames MODEL=OptModelName
```

When performing a transient analysis bisection with the `.MEASURE` statement, use the following syntax:

```
.MEASURE TRAN MeasureName MeasureClause GOAL=GOALMIN/GOALMAX
```

Setup Time Analysis

This example uses a bisectional search to find the minimum setup time for a D flip-flop. The circuit for this example is `dff_top.sp`, which is located in directory `$installdir/demo/hspice/bisec`.

[Figure 180 on page 944](#) and [Figure 181 on page 945](#) show the results of this demo. HSPICE does not directly optimize the setup time, but extracts it from its relationship with the `DelayTime` parameter (the time before the data signal), which is the parameter to optimize.

Input Listing

The following portion of the input listing shows how `.TRAN` analysis, the `DelayTime` parameter, and `.MEASURE` statements are used in bisection:

```
* DFF_top Bisection Search for Setup Time  
* PWL Stimulus  
v28 data gnd PWL  
+ 0s 5v  
+ 1n 5v  
+ 2n 0v  
+ Td = "DelayTime" $ Offsets Data from time by DelayTime
```

```
v27 clock gnd PWL
+ 0s 0v
+ 3n 0v
+ 4n 5v
* Specify DelayTime as the search parameter and provide
* the lower and upper limits.
.PARAM DelayTime= Opt1 ( 0.0n, 0.0n, 5.0n )
* Transient simulation with Bisection Optimization
.TRAN 0.1n 8n 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 SetupTimeTrig v(Data)Val = 'v(Vdd)/2'
+ Fall = 1
+ Targ v(Clock)Val = 'v(Vdd)/2'
+ Rise = 1
* Optimization Model
.MODEL OptMod Opt
+ Method = Bisection
.OPTION Post Brief NoMod
```

Chapter 30: Timing Analysis Using Bisection

Setup Time Analysis

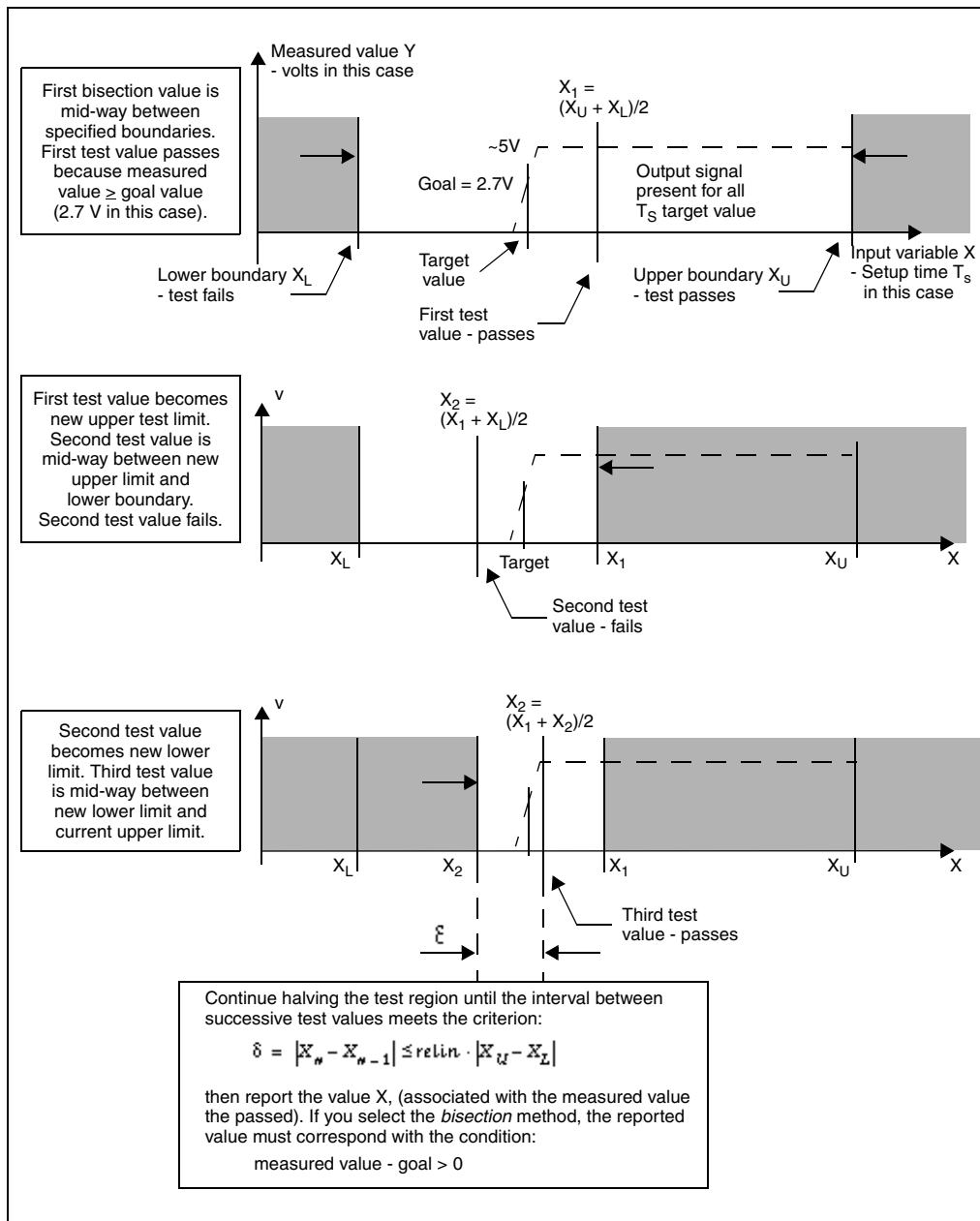


Figure 180 Bisection Example for Three Iterations

Results

The upper plot in [Figure 181 on page 945](#) shows the relationship between the clock and the data pulses that determine the setup time. The bottom plot is the output transition.



Figure 181 Transition at Minimum Setup Time

Find the actual value for the setup time in the “Optimization Results” section of the output listing file:

```

optimization completed, the condition
relin = 1.0000E-03 is satisfied
**** optimized parameters opt1
.PARAM DelayTime = 1.7822n
...
maxvout = 5.0001E+00 at= 4.8984E-09
from      = .0000E+00      to= 8.0000E-09
setuptime= 2.1777E-10 targ= 3.5000E-09 trig= 3.2822E-09
    
```

This listing file excerpt shows that the optimal value for the setup time is 0.21777 ns.

The upper plot in [Figure 182](#) shows examples of early and late data transitions, and the transition at the minimum setup time. The bottom plot shows how the timing of the data transition affects the output transition. The following analysis statement produces these results:

```

* Sweep 3 values for DelayTime   Early   Optim   Late
.TRAN 0.1n 8n Sweep DelayTime Poi 3   0.0n   1.7822   5.0n
    
```

Chapter 30: Timing Analysis Using Bisection

Minimum Pulse Width Analysis

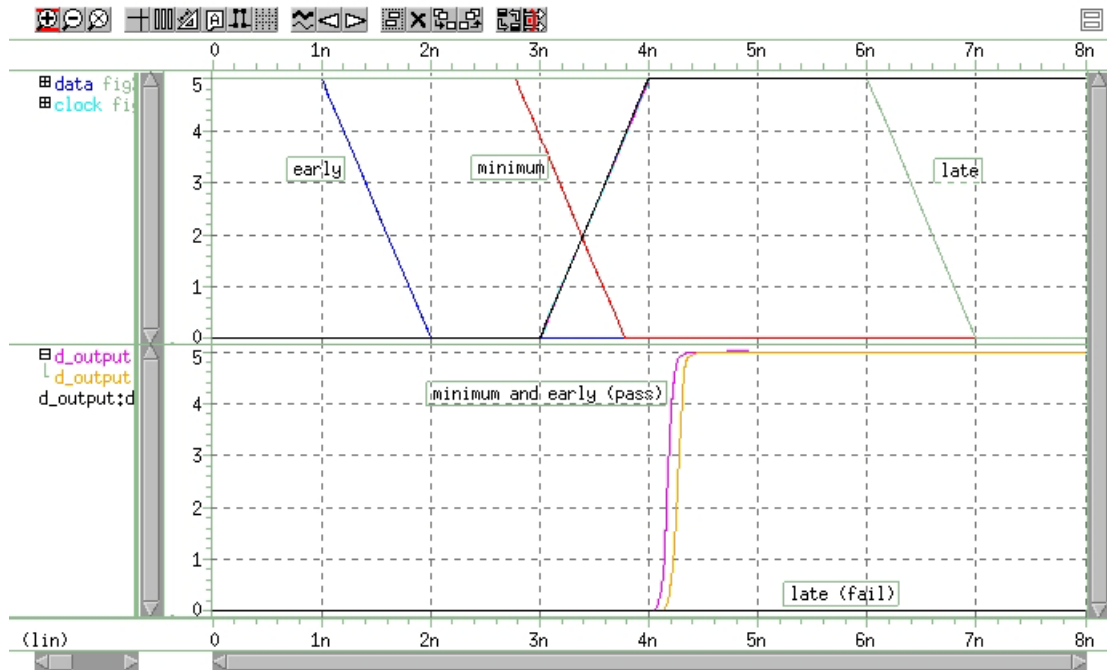


Figure 182 Early, Minimum, and Late Setup and Hold Times

This analysis produces the following results:

```
*** parameter DelayTime = .000E+00 *** $ Early
setuptime= 2.0000E-09 targ= 3.5000E-09   trig= 1.5000E-09
*** parameter DelayTime = 1.7822E-09 *** $ Optimal
setuptime= 2.1780E-10 targ= 3.5000E-09   trig= 3.2822E-09
*** parameter DelayTime = 5.000E-09 *** $ Late
setuptime= -3.0000E-09 targ= 3.5000E-09   trig= 6.5000E-09
```

Minimum Pulse Width Analysis

This example uses a pass/fail bisectional search to find a minimum pulse width required so the input pulse can propagate to the output of an inverter. It is based on demonstration netlist `iva_a.sp`, which is available in directory `$installdir/demo/hspice/bisect`. [Figure 183](#) shows the results of this demo.

Input Listing Directory

This input listing file is located in: `$installdir/demo/hspice/bisect/inv_a.sp`.

Results

Figure 183 shows results of pass/fail search, for two different capacitive loads.

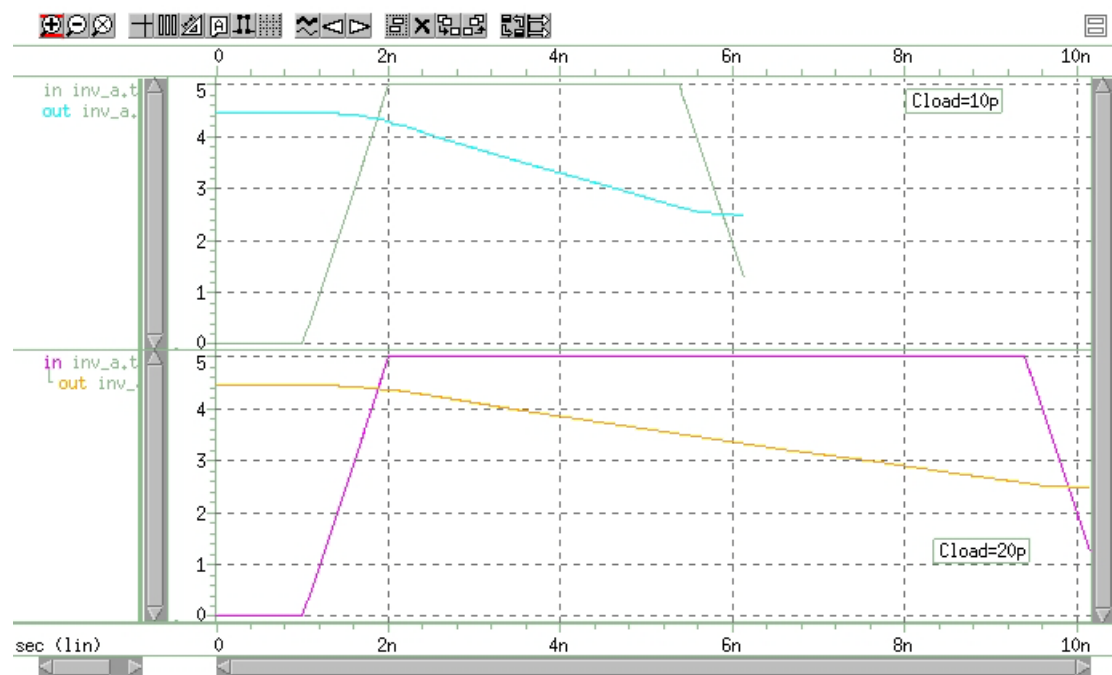


Figure 183 Results of Bisectional Pass/Fail Search

Pushout Bisection Methodology

For setup- or hold-time optimization analysis, a normal bisection method varies the input timing to find the point just before failure. At this point, delaying the input longer results in failure, and the output does not transition. In pushout analysis, instead of finding the last point just before failure, the first successful output transition is used as the golden target. You can then apply a maximum allowed pushout time to decide if the subsequent results are classified as passes or failures. Finding the optimized pushout result is similar to a normal bisection because both use a binary search to approach the desired solution. The main difference is the goal or the optimization criteria.

Chapter 30: Timing Analysis Using Bisection

Pushout Bisection Methodology

You can add measure options to support selective pushout, using these keywords:

- **POSITIVE:** pushout constraints only take effect when the measuring results are larger than the golden measure. For example:

```
.MEASURE TRAN result MeasureClause pushout=time  
+ pushout_perpercentage POSITIVE
```

- **NEGATIVE:** pushout constraints only take effect when the measuring results are smaller than the golden measure. For example:

```
.MEASURE TRAN result MeasureClause pushout=time  
+ pushout_perpercentage NEGATIVE
```

To limit the range you can add both absolute and relative pushout together. (Note the comma-separated parameters.)

For example:

```
.Measure Tran pushout When v(D_Output)='vih/2'  
+ rise=1 pushout=20p,50p pushout_per=0.1
```

The final measure result for the preceding example should be in the range of:

$$| \text{measresult-goldmeas} | < \text{Min} (\text{pushout_max}, \text{pushout_per} * \text{goldmeas})$$

Or the final measure result should satisfy,

$$\text{Max}(\text{pushout_per} * \text{goldmeas}, \text{pushout_min})$$

See [.MEASURE \(Pushout Bisection\)](#) for more information. If neither of the keywords above is set, the pushout flow follows the standard described above and shown in the next example.

The following example ([Figure 184 on page 949](#)) shows a transition of Vin with a varying delay during a Vclk transition. When the “lower” input transitions, it indicates that the device being tested is functioning. The “upper” input does not transition, which indicates that the device is not functioning.

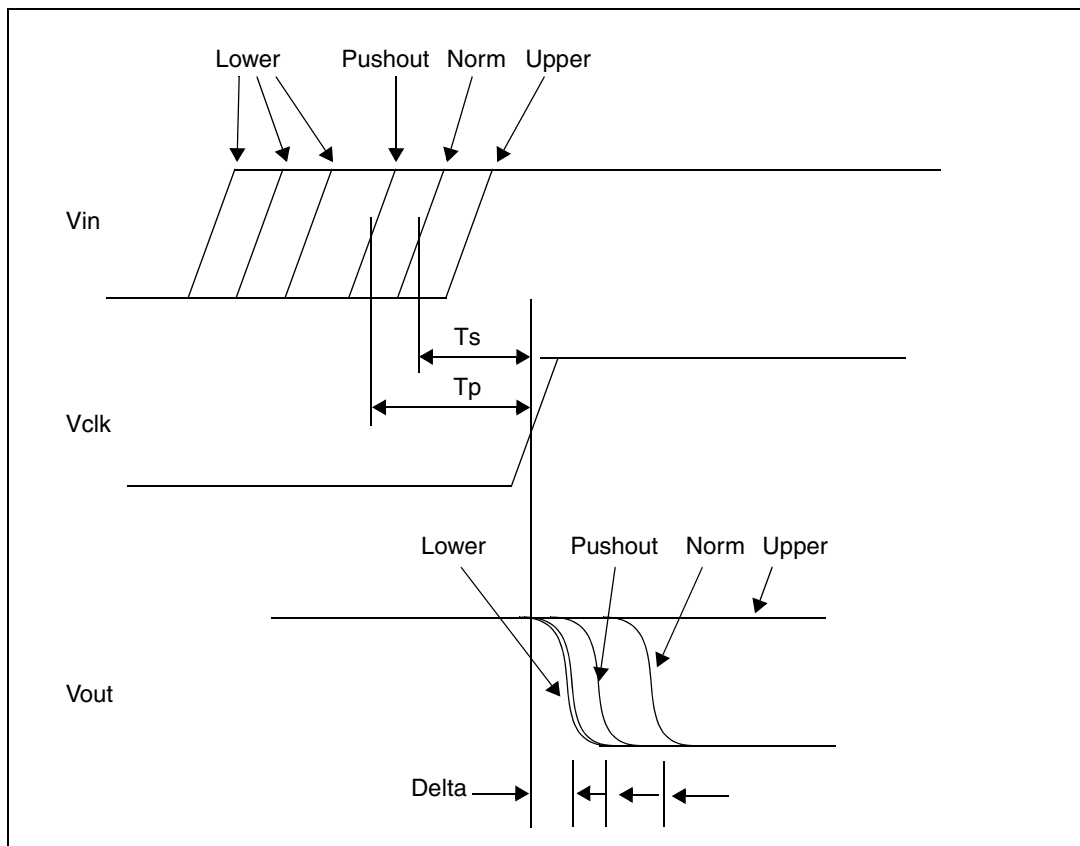


Figure 184 Pushout Bisection Example

Consider the pushout bisection example located in the following directory: `$installdir/demo/hspice/bisect/dff_push.sp`. (See the path to this example, which is a DFF pushout bisection search for setup time, in [Bisection-Timing Analysis Examples](#) in this user guide.)

The transition of V_{in} is delayed by varying amounts with respect to a V_{clk} transition. For the Lower input transition, the output transitions and indicates that the device under test is functioning. For the Upper input transition, the output does not transition and indicates that the device is not functioning.

Normal bisection varies the input timing to find the point just before failure (called Norm here). At this point, a failure occurs when the device is delayed longer and the output does not transition. The circuit works at points between Lower and Norm, but the output transition is delayed from the lower conditions by setting Delta. This is called the Pushout. The pushout can also lie between Norm and Upper, which depends on your use of the lower or upper option.

Chapter 30: Timing Analysis Using Bisection

Using RELOUT and RELIN to Affect HSPICE Bisection Optimization

In you use normal bisection in this example, the resulting gain is $\text{delaytime}=7.374\text{e-}10$ $\text{pushout}=-1.672\text{e-}09$. Instead, when setting $\text{pushout}=0.01$, the result is $\text{delaytime}=3.918\text{e-}11$ $\text{pushout}=3.970\text{e-}09$.

Using RELOUT and RELIN to Affect HSPICE Bisection Optimization

You can use the `.MODEL` command parameters `RELOUT` and `RELIN` to influence the HSPICE Bisection optimization convergence process. The optimization statement has many parameters. The default values of these parameters are suitable for most applications. However, if the optimization does not converge, you might need to adjust the error tolerance of the `RELOUT` and `RELIN` command-line parameters on the `.MODEL` card.

.MODEL Optimization Syntax

```
.MODEL mname OPT [METHOD=BISECTION|PASSFAIL] [close=num]
+ [max] [cut=val] [difsiz=val] [grad=val] [parmin=val]
+ [relin=val] [relout=val] [absout=val]
+ [itrop=val] [absin=val]
+ [DYNACC=0|1] [cendif=num]
```

RELOUT Parameter

Use the `RELOUT` parameter to set the output error tolerance. For example, if you specified the `GOAL` value as `5v` and `RELOUT=0.1`, when the bisection output is smaller than $0.5v(5v * 0.1)$, `RELOUT` is satisfied.

RELIN Parameter

Use the `RELIN` parameter to set the input error tolerance. For example, if the bisection initial input window is `5ns`, and `RELIN` is set to `1E-3` (default), when the difference between the latest bisection input and the previous one is smaller than $5ps(5ns * 1E-3)$, `RELIN` is satisfied.

The following lines are from the `*.lis` file of a successful optimization run that appear only if you use the `OPTLST=3` option in the netlist.

```
*START OF *.lis FILE
.
.
.
parm names init guess, lower, upper bounds

delaytime          0.          0.          5.0000E-09          0.
```

Chapter 30: Timing Analysis Using Bisection
Using RELOUT and RELIN to Affect HSPICE Bisection Optimization

```

bisec-opt.  iter =   1 xlo =           0.      xhi =   5.00000E-09
              x =           0.      xnew =   5.00000E-09
              err =  -1.91761E-02
.
.
.
bisec-opt.  iter =   7 xlo =   1.87500E-09  xhi =   2.03125E-09
              x =   2.03125E-09  xnew =   1.95313E-09
              err =   0.99949
bisec-opt.  iter =   8 xlo =   1.95313E-09  xhi =   2.03125E-09
              x =   1.95313E-09  xnew =   1.99219E-09
              err =  -0.11111
.
.
.
bisec-opt.  iter =  10 xlo =   1.21094E-09  xhi =   1.23047E-09
              x =   1.23047E-09  xnew =   1.22070E-09
              err =   0.46334
bisec-opt.  iter =  11 xlo =   1.22070E-09  xhi =   1.23047E-09
              x =   1.22070E-09  xnew =   1.22559E-09
              err =  -1.89848E-02
      optimization completed, the condition
      relin =  1.0000E-03 is satisfied
      optimization completed, the condition
      relout =  0.4000      is satisfied

* END OF *.lis FILE

```

In each iteration, err is calculated as $(GOAL - RESULT) / GOAL$
 where: $GOAL$ is the target value and $RESULT$ is the value calculated for each iteration.

RELOUT is satisfied when $|err| < RELOUT$

RELIN is satisfied when $|X(new) - X(new-1)| < RELIN * |X(upper) - X(lower)|$

where: $X(new)$ is the $xnew$ value of the n th iteration and $X(new-1)$ is the $xnew$ value of the $(n-1)$ th iteration.

Both RELIN and RELOUT must be satisfied before the optimization can converge.

Note that in $iter = 8$, the condition for RELOUT is satisfied but the condition for RELIN is not. (Substitute $5n$ for $X(upper)$ and $0n$ for $X(lower)$ in the

Chapter 30: Timing Analysis Using Bisection

Using Bisection with Monte Carlo Analysis

formula for RELIN). In `iter = 11`, check that both RELIN and RELOUT are satisfied.

Note: If you use `.OPTION RELIN` its value overrides any value you enter on the `.MODEL` statement for the parameter RELIN. (There is *no* `.OPTION RELOUT` in HSPICE).

For more information about using RELOUT and RELIN in the `.MODEL` command for bisection, see [.MODEL](#) in the *HSPICE Reference Manual: Commands and Control Options*.

Using Bisection with Monte Carlo Analysis

Bisection method integrated with Monte Carlo analysis enable both analyses to run in a single simulation. Bisection with Monte Carlo analysis can be run either with traditional HSPICE Monte Carlo (see [Chapter 22, Monte Carlo—Traditional Flow Statistical Analysis](#)) or Monte Carlo using the Variation Block flow (see [Chapter 24, Monte Carlo Analysis—Variation Block Flow](#)).

Monte Carlo analysis is the generic tool for simulating the effects of process variation on circuit performance. Bisection uses a binary search to find the value of an input variable (target value), which satisfies a goal value of an output variable. It is used extensively in analyzing circuit timing violations, performing timing optimization, and in sequential cell characterization.

More than one sweep loop can run in transient simulations. Bisection is enabled as the inner loop, working with a Monte Carlo sweep as the outer loop, during transient analysis. This 2-sweep loop support is exclusive to a bisection with Monte Carlo sweep.

If you add additional independent random variables to a Monte Carlo run you might see that none of these variables has any impact on the simulation run. You might see differences in statistical results between simulations with and without these additional independent random variables. The difference is due the way random values are assigned to them, not to varying the number independent random variables. In the two Monte Carlo modes discussed in the following section, the only difference is the number of independent random variables.

Setting Up Monte Carlo Analysis with Bisection

Running a bisection/Monte Carlo simulation combines both existing procedures and uses enhanced syntax.

Bisection with a Monte Carlo sweep retains the setup requirements for bisection and Monte Carlo. To set up a Monte Carlo analysis, you use one of the two following HSPICE representation styles:

- `.Variation` statement—Specify distributions on model parameters in a Variation Block see [Monte Carlo-Specific Variation Block Options](#)
- `.PARAM` statements—Set a model or element parameter to a Gaussian, Uniform, or Limit function distribution; this statistical process description is used with analysis commands that typically include `.MEASURE` statements to calculate the response mean, variance, sigma, and standard deviation. `.DC`, `.AC`, or `.TRAN` analysis statements enable MONTE.

To set up a bisection run, you use the following statements.

- `.PARAM ParamName= OptParFun (Initial, Lower, Upper—`
Defines a normal optimization specification of the parameter to be optimized.
- `.MEASURE Tran GOAL=GoalValue—`Sets a GOAL value
- `.MODEL optmodel OPT—`Specifies bisection method and relative input/output tolerance.
- `.TRAN . . . SWEEP OPTIMIZE—`Performs transient analysis bisection.

Performing Bisection with Monte Carlo Sweep

When performing bisection with a Monte Carlo sweep, use one or more `.Variation` blocks or parameter statements and one or more `.MEASURE` statements for Monte Carlo and another set of `.MEASURE`, `.PARAM` and `.MODEL Opt` statements for bisection setup. However, only a single common `.TRAN` analysis statement is used for both analyses, plus an enhanced syntax.

.TRAN Syntax for Bisection with Monte Carlo Sweep

```
.TRAN tstep tstop [START=val]
+ SWEEP
+ OPTIMIZE=opt_par_fun|OPTxxx
+ RESULTS=measnames MODEL=optmod
+ MONTE=MCcommand
```

Chapter 30: Timing Analysis Using Bisection

Using Bisection with Monte Carlo Analysis

All arguments have the same meanings as those of the current bisection and Monte Carlo `.TRANSyntax`. For argument names and their descriptions, refer to `.TRAN` in the *HSPICE Reference Manual: Commands and Control Options*.

Note: Monte Carlo random data generation procedure is independent of the bisection analysis results.

Example

In the following example, HSPICE runs 30 bisection analyses, using the 10th to 39th Monte Carlo iterations.

```
.Tran 1n 8n
+ SWEEP                                $$ keyword to enable both functions
+ Optimize = Opt1                       $$ Bisection setup
+ Result = MaxVout                       $$ Bisection setup
+ Model = OptMod                         $$ Bisection setup
+ MONTE = 30 firstrun=10                 $$ Monte Carlo setup
```

Ordinary Output Files

Ordinary Monte Carlo output depends on how you set the output commands. Transient Monte Carlo results are saved in measurement report files (`*.mct#`, `*.mt#`), the output list `*.lis` file, and/or the waveform output file (`*.tr#`) which superimposes all iterations as a single plot.

Ordinary bisection analysis output is typically reported in the `*.lis` file and the measurement report file, `*mt#` file. A waveform output file displays the results of simulation using the value of the optimized parameter.

Note: The `.mt#` format consists of 72 characters in a line and fields that contain 16 characters each.

Bisection with Monte Carlo Sweep Output

The output list file, `*.lis` reports results and/or iteration information from both the Bisection analysis and the Monte Carlo sweep. Bisection results and any additional information depend on the setting of the option `OPTLST`, which is embedded in every Monte Carlo run.

See the following sample `*.lis` file, where the types of information are defined on the right side as:

- Monte Carlo output/information
- Bisection output/information
- Common output/information

List File Content	Info type
<pre> Combination of output *.lis report for bisection with monte carlo.sweep Opening plot unit= 15 file=./results/inv.pa0 ***** HSPICE C-2009.03 32-BIT 11:21:02 01/16/2009 linux ***** operating point information tnom= 25.000 temp=25.000 ***** operating point status is voltage simulation time is 0. node =voltage node =voltage node =voltage +0:1 = 5.0000 0:2 = 5.0000 0:3 = 3.6493n +0:4 = 3.6493n 0:in = 0. 0:out = 5.0000 1 ***** HSPICE C-2009.03 32-BIT 11:21:02 01/16/2009 linux ***** ***** transient analysis tnom= 25.000 temp= 25.000 *** monte carlo index = 1 *** MONTE CARLO PARAMETER DEFINITIONS par1 = 2.7518E-08 par2 = -3.1778E-07 par3 = -4.1848E-08 </pre>	<p>Common output/ information</p> <p>Monte Carlo output/ information</p>

Chapter 30: Timing Analysis Using Bisection
 Using Bisection with Monte Carlo Analysis

List File Content	Info type
entering lmopt	Bisection output/ information
parm names init guess, lower, upper bounds	
delaytime 0.000 0.000 5.0000E-09 0.000	
bisec-opt. iter = 1 xlo = 0.0000 xhi = 5.00000E-09	
x = 0.0000 xnew = 5.00000E-09	
err = -4.97943E-05	
bisec-opt. iter = 2 xlo = 0.0000 xhi = 5.00000E-09	
x = 5.00000E-09 xnew = 2.50000E-09	
err = 0.99997	
bisec-opt. iter = 11 xlo = 1.80664E-09 xhi = 1.81641E-09	
x = 1.80664E-09 xnew = 1.81152E-09	
err = -1.91099E-05	
optimization completed, the condition	
relin = 1.0000E-03 is satisfied	
optimization completed, the condition	
relout = 1.0000E-03 is satisfied	
**** optimized parameters opt1	
.param delaytime = 1.8066n	

Chapter 30: Timing Analysis Using Bisection
Using Bisection with Monte Carlo Analysis

List File Content	Info type
s_delay= 2.5204E-10 targ= 4.3714E-10 trig= 1.8510E-10	Common output/ information
s_power= 6.6497E-03 from= 0.0000E+00 to= 1.0000E-09	
m_delay= 2.5204E-10 targ= 4.3714E-10 trig= 1.8510E-10	Monte Carlo output/ information
***** job concluded	
*** monte carlo index = 2 ***	
par1 = 2.3518E-08	
par2 = -3.3778E-07	
par3 = -4.3848E-08	

Chapter 30: Timing Analysis Using Bisection
 Using Bisection with Monte Carlo Analysis

List File Content	Info type
entering lmopt parm names init guess, lower, upper bounds	Bisection output/ information
delaytime 0.000 0.000 5.0000E-09 0.000	
bisection-opt. iter = 1 xlo = 0.0000 xhi = 5.0000E-09	
x = 0.0000 xnew = 5.0000E-09	
err = -4.97943E-05	
bisection-opt. iter = 2 xlo = 0.0000 xhi = 5.0000E-09	
x = 5.0000E-09 xnew = 2.5000E-09	
err = 0.99997	
....	
bisection-opt. iter = 11 xlo = 1.80664E-09 xhi = 1.81641E-09	
x = 1.80664E-09 xnew = 1.81152E-09	
err = -1.91099E-05	
optimization completed, the condition	
relin = 1.0000E-03 is satisfied	
**** optimized parameters opt1	
.param delaytime = 1.8066n	
s_delay= 2.5204E-10 targ= 4.3714E-10 trig= 1.8510E-10	Common output/ information
s_power= 6.6497E-03 from= 0.0000E+00 to= 1.0000E-09	
m_delay= 2.5204E-10 targ= 4.3714E-10 trig= 1.8510E-10	

Chapter 30: Timing Analysis Using Bisection
Using Bisection with Monte Carlo Analysis

List File Content	Info type
***** job concluded	Monte Carlo output/ information
*** monte carlo index = 3 ***	
par1 = 2.3518E-08	
par2 = -3.3778E-07	
par3 = -4.3848E-08	
...	
meas_variable = s_delay	
mean = 251.0688p varian = 4.397e-22	
sigma = 20.9695p avgdev = 14.2926p	
max = 271.5352p min = 229.6299p	
1-sigma = 20.9695p median = 271.5352p	
meas_variable = s_power	
mean = 6.6620m varian = 971.5124p	
sigma = 31.1691u avgdev = 23.6264u	
max = 6.6974m min = 6.6388m	
1-sigma = 31.1691u median = 6.6974m	
meas_variable = m_delay	
mean = 251.0688p varian = 4.397e-22	
sigma = 20.9695p avgdev = 14.2926p	
max = 271.5352p min = 229.6299p	
1-sigma = 20.9695p median = 271.5352p	

Chapter 30: Timing Analysis Using Bisection
Using Bisection with Monte Carlo Analysis

List File Content	Info type
**** job concluded	Common output/ information
***** HSPICE C-2009.03 32-BIT (Jan 16 2009) 11:21:02 01/16/2009 linux	

***** job statistics summary tnom= 25.000 temp= 25.000	
***** total memory used 167 kbytes	
# nodes = 23 # elements= 10	
# diodes= 0 # bjts = 0 # jfets = 0 # mosfets = 8	
# va device = 0	
analysis time # points tot. iter conv.iter	
op point 0.02 1 24	
transient 0.00 153 219 76 rev=1	
readin 0.00	
errchk 0.01	
setup 0.00	
output 0.00	
total cpu time 0.02 seconds	
job started at 11:21:02 01/16/2009	
job ended at 11:21:04 01/16/2009	

Output Measurement File

The output measurement file has the same information recorded based on the `.MEAS` statements for either bisection analysis or the Monte Carlo sweep. The measure file is one set of results per Monte Carlo sample; and the results are for the optimized value from the bisection analysis. The bisection measure results include the optimized parameter and the measure results.

In the following example, `delaytime` is the parameter optimized, `setuptime` is the measurement result with this `delaytime`.

```
.TITLE '*****'  
index rb1@rb1x delaytime setuptime  
temper alter#  
1.0000 2.100e+04 5.982e-10 4.018e-10  
25.0000 1.0000  
2.0000 1.900e+04 3.448e-10 6.552e-10  
25.0000 1.0000  
...
```

Output Waveform

The output waveform file superimposes all iterations of the Monte Carlo sweep as a single plot, while each iteration simulation of Monte Carlo uses the optimized parameter from Bisection.

Note: If a particular Monte Carlo run fails, either due to Bisection failure or any other reason, HSPICE outputs a 'failed' flag in the `*.mt#` file, and continues with the next Monte Carlo run. The failed point is not included in the summary statistical computations.

If a Monte Carlo simulation terminates when one sample is failed, you can use `.option MONTECON=1` (the default) to force the simulation to continue under this condition.

Bisection Optimization Application Example

```
.model optMod1 OPT METHOD=BISECTION  
.param VDx = optFunc1(1.0,1.0,1.4)  
.option OPTLST = 1  
.measure optMeasure TRIG v(clk) VAL='0.5*VDx' RISE=2 TARG v(q)  
+ VAL='0.5*VDx' RISE=1 GOAL=60p  
.tran 0.01n 10n SWEEP OPTIMIZE=optFunc1 RESULTS=optMeasure  
+ MODEL=optMod1
```

Chapter 30: Timing Analysis Using Bisection

Bisection Optimization Application Example

Caution:

- A simple `.param VDx = 1.1` statement defined after `.param VDx = optFunc1(...)` will overwrite the optimization function `optFunc1`.
- A voltage source with the name `VDx` must not coexist with the parameter `VDx`. Otherwise, the DC value of the voltage source `VDx` will be varied while the parameter `VDx` is left unchanged.

Part 6: Errors-Warnings/ Demonstration Files

The section contains the following chapters:

- [Chapter 31, Running Demonstration Files](#)
- [Chapter 32, Warning/Error Messages](#)

These groups of Example Demo files are available:

- [HSPICE Integration to ADE Demonstration Examples](#)
- [Applications of General Interest Examples](#)
- [Back-Annotation Demo Cases](#)
- [Behavioral Application Examples](#)
- [Benchmark Examples](#)
- [Bisection-Timing Analysis Examples](#)
- [BJT and Diode Examples](#)
- [Cell Characterization Examples](#)
- [Circuit Optimization Examples](#)
- [Device Optimization Examples](#)
- [Encryption Examples](#)
- [Fourier Analysis Examples](#)
- [Filters Examples](#)
- [IBIS Examples](#)
- [Loop Stability Analysis](#)
- [Magnetics Examples](#)
- [MOSFET Device Examples](#)
- [RF Examples](#)
- [Signal Integrity Examples](#)
- [Sources Examples](#)

- S-parameter Examples
- Transmission Lines Examples
- Transmission (W-element) Line Examples
- Variability Examples
- Verilog-A Examples

Running Demonstration Files

Contains examples of basic file construction techniques, advanced features, and simulation hints. Lists and describes over 300 HSPICE demonstration input files plus paths to these files in the general distribution. For HSPICE RF-specific input files see the Getting Started chapter of the HSPICE User Guide: Advanced Analog Analysis and Simulation.

HSPICE ships hundreds of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

These topics are covered in the following sections:

- [Using the Demo Directory Tree](#)
- [Two-Bit Adder Demo](#)
- [MOS I-V and C-V Plot Example Input File](#)
- [Temperature Coefficients Demo](#)
- [Modeling Wide-Channel MOS Transistors](#)
- [Listing of Demonstration Input Files](#)

Using the Demo Directory Tree

To run demonstration files: go to your HSPICE installed version location e.g.: `path_to_hspice_version/hspice/demo/hspice`. In this directory you will find multiple demo files. After a proper hspice path has been set up, you can execute Linux/Solaris/HP:

```
%> hspice -i your_spice_file -o output_file
```

On Windows, execute Start > All > Programs > `your_hspice_installed_version` > `HSPICE-installed_version` > file > simulate and proceed.

Chapter 31: Running Demonstration Files

Using the Demo Directory Tree

The tables in the section [Listing of Demonstration Input Files on page 982](#) list demonstration files, which are designed as training examples. All HSPICE distributions include these examples in the demo directory tree, where `$installdir` is the installation directory environment variable:

Table 81 Demo Directories

Directory Path	File Directory	Description
<code>\$installdir/demo/hspice</code>	<code>/aa_integ</code>	HSPICE integration tutorial, Cadence® Virtuoso® Analog Design Environment
	<code>/apps</code>	General applications
	<code>/back_annotation</code>	Usage of <i>BA_options</i>
	<code>/behave</code>	Analog behavioral components
	<code>/bench</code>	Standard benchmarks
	<code>/bjt</code>	Bipolar components
	<code>/bisection</code>	Bisection optimization
	<code>/cchar</code>	Characteristics of cell prototypes
	<code>/ciropt</code>	Circuit level optimization
	<code>/ddl</code>	Discrete Device Library
	<code>/devopt</code>	Device level optimization
	<code>/fft</code>	Fourier analysis
	<code>/encrypt</code>	Traditional, 8-bit, and 3DES encryption
	<code>/filters</code>	Filters
	<code>/ibis</code>	IBIS examples
	<code>/mag</code>	Transformers, magnetic core components
	<code>/mos</code>	MOS components
	<code>/si</code>	Signal Integrity applications
	<code>/sources</code>	Dependent and independent sources
	<code>/sparam</code>	S-parameter applications
	<code>/tline</code>	Filters and transmission lines
	<code>/twline</code>	W-element transmission lines and field solvers
	<code>/variability</code>	Variation Block, Monte Carlo, and AC/DC Mismatch examples
	<code>/veriloga</code>	Verilog-A examples

Table 81 Demo Directories (Continued)

Directory Path	File Directory	Description
<code>\$installdir/demo/hspicerf</code>	<code>/examples/</code>	RF examples

Two-Bit Adder Demo

This two-bit adder shows how to improve efficiency, accuracy, and productivity in circuit simulation. The adder is in the `$installdir/demo/hspice/apps/mos2bit.sp` demonstration file. It consists of two-input NAND gates, defined using the NAND subcircuit. CMOS devices include length, width, and output loading parameters. Descriptive names enhance the readability of this circuit.

One-Bit Subcircuit

The ONEBIT subcircuit defines the two half adders, with carry in and carry out. To create the two-bit adder, HSPICE uses two calls to ONEBIT. Independent piecewise linear voltage sources provide the input stimuli. The *R* repeat function creates complex waveforms.

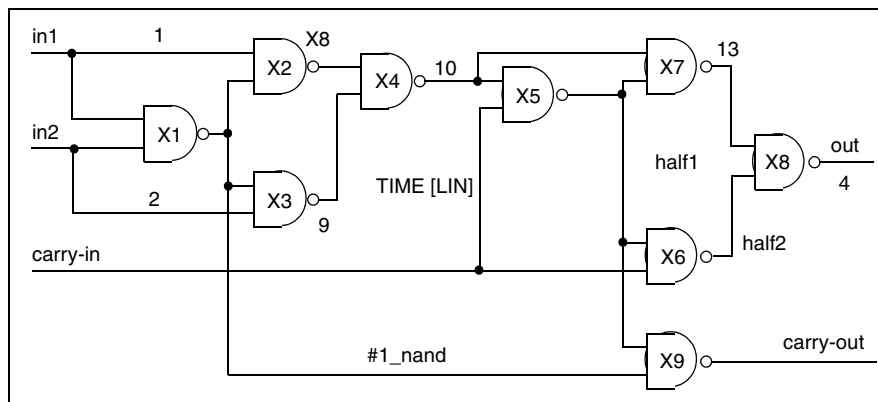


Figure 185 One-bit Adder subcircuit

Chapter 31: Running Demonstration Files
MOS I-V and C-V Plotting Demo

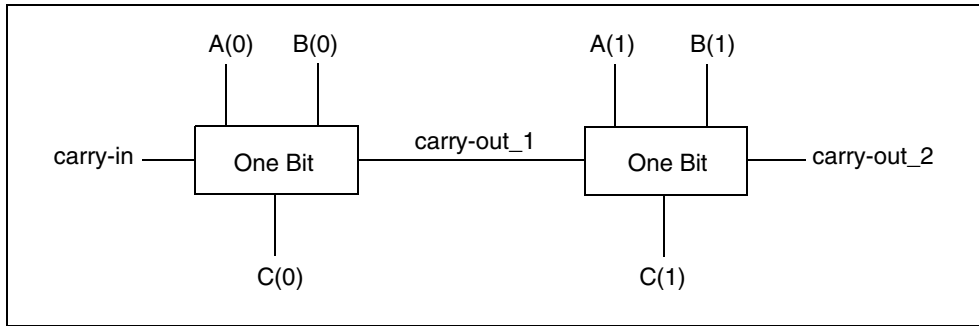


Figure 186 Two-bit Adder Circuit

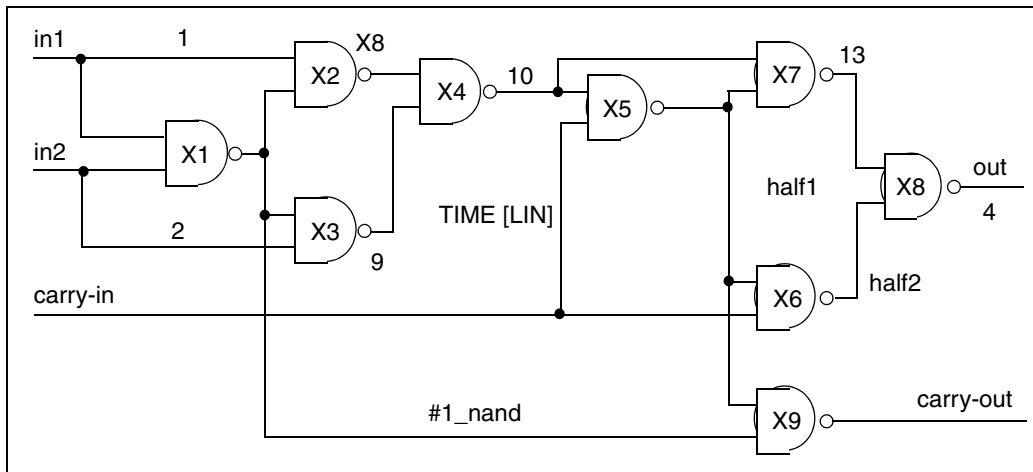


Figure 187 1-bit NAND Gate Binary Adder

MOS Two-Bit Adder Input File

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/apps/mos2bit.sp
```

MOS I-V and C-V Plotting Demo

To diagnose a simulation or modeling problem, you usually need to review the basic characteristics of the transistors. You can use this demonstration

template file, `$installdir/demo/hspice/mos/mosivcv.sp`, with any MOS model. The example shows how to easily create input files, and how to display the complete graphical results. The following features aid model evaluations:

Table 82 MOS I-V and C-V Plotting Demo

Value	Description
SCALE=1u	Sets the element units to microns (not meters). Most circuit designs use microns.
DCCAP	Forces HSPICE to evaluate the voltage variable capacitors, during a DC sweep.
node names	Eases circuit clarity. Symbolic name contains up to 16 characters.
.PRINT	.PRINT statements print internal variables.

Printing Variables

Use this template to print internal variables, such as:

Table 83 Demo Printing Variables

Variable	Description
i(mn1)	i1, i2, i3, or i4 specifies true branch currents for each transistor node.
LV18(mn6)	Total gate capacitance (C-V plot).
LX7(mn1)	GM gate transconductance. (LX8 specifies GDS; LX9 specifies GMB).

Chapter 31: Running Demonstration Files
MOS I-V and C-V Plotting Demo

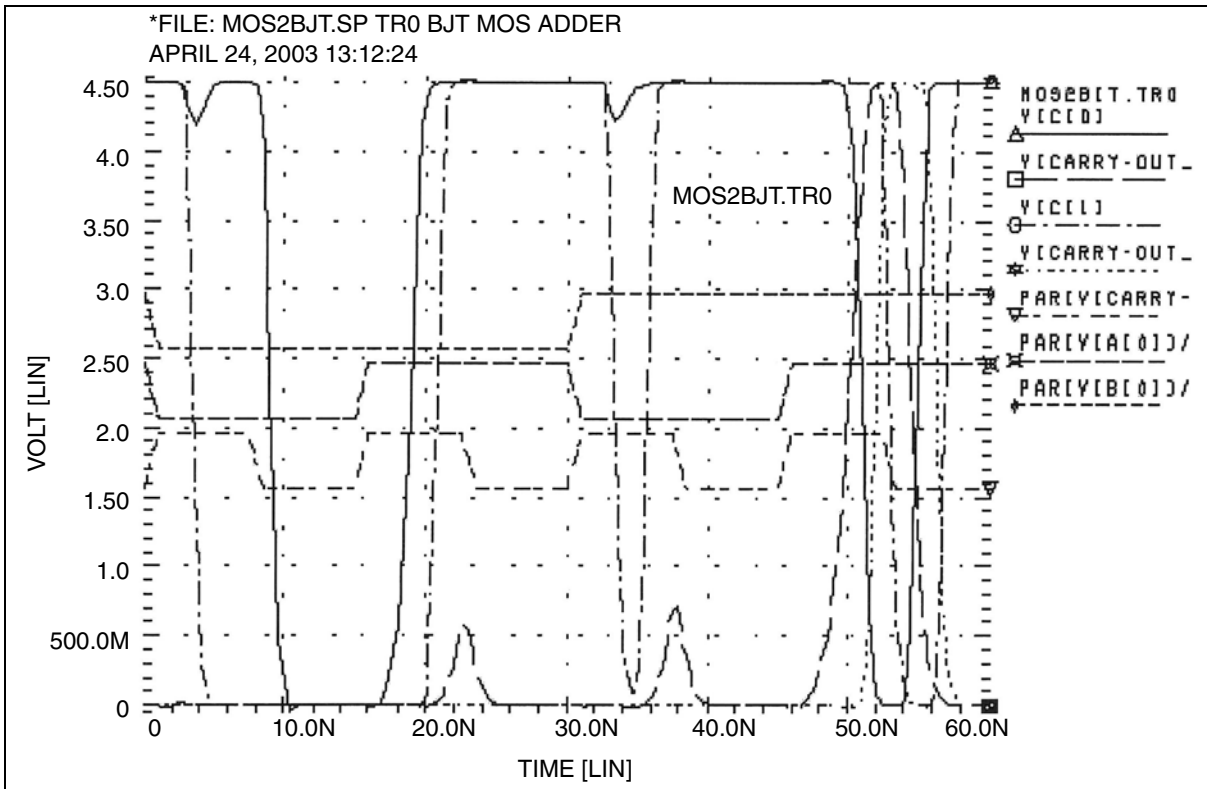


Figure 188 MOS IDS Plot

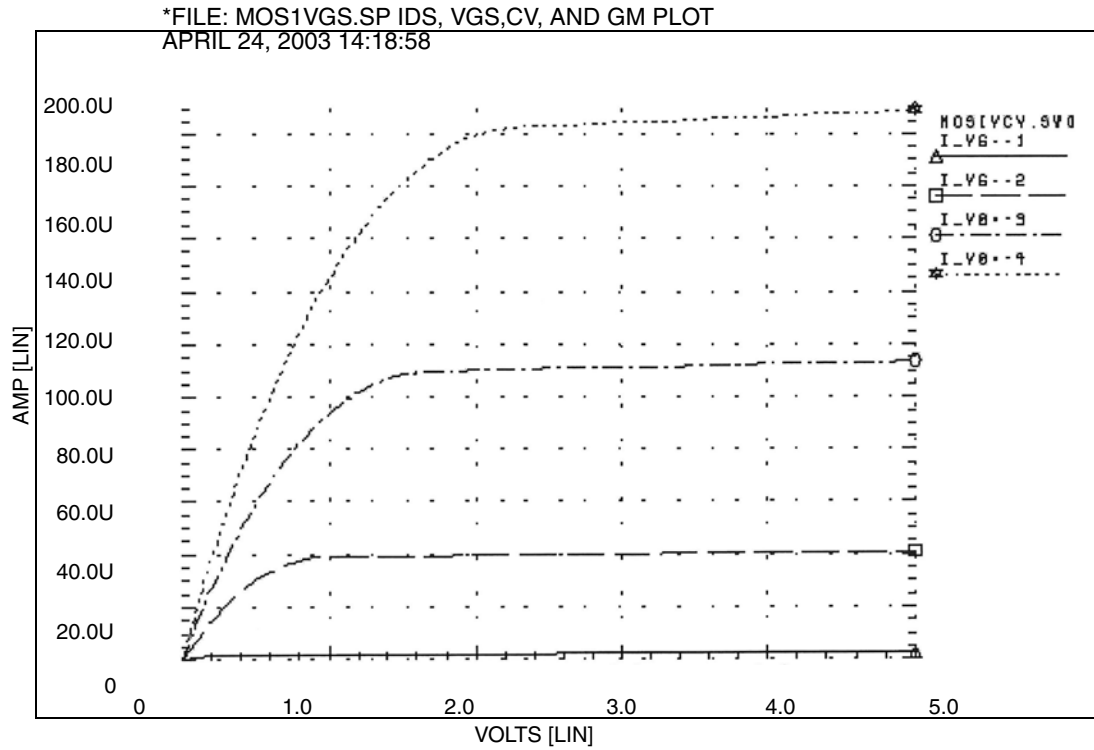


Figure 189 MOS VGS Plot

Chapter 31: Running Demonstration Files
MOS I-V and C-V Plotting Demo

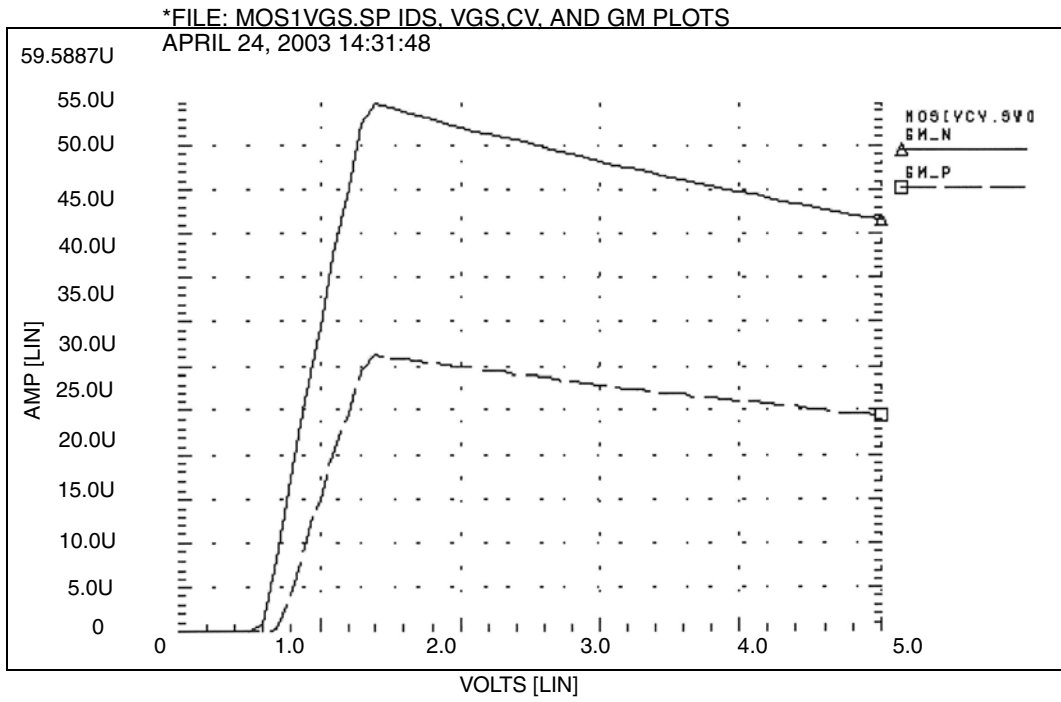


Figure 190 MOS GM Plot

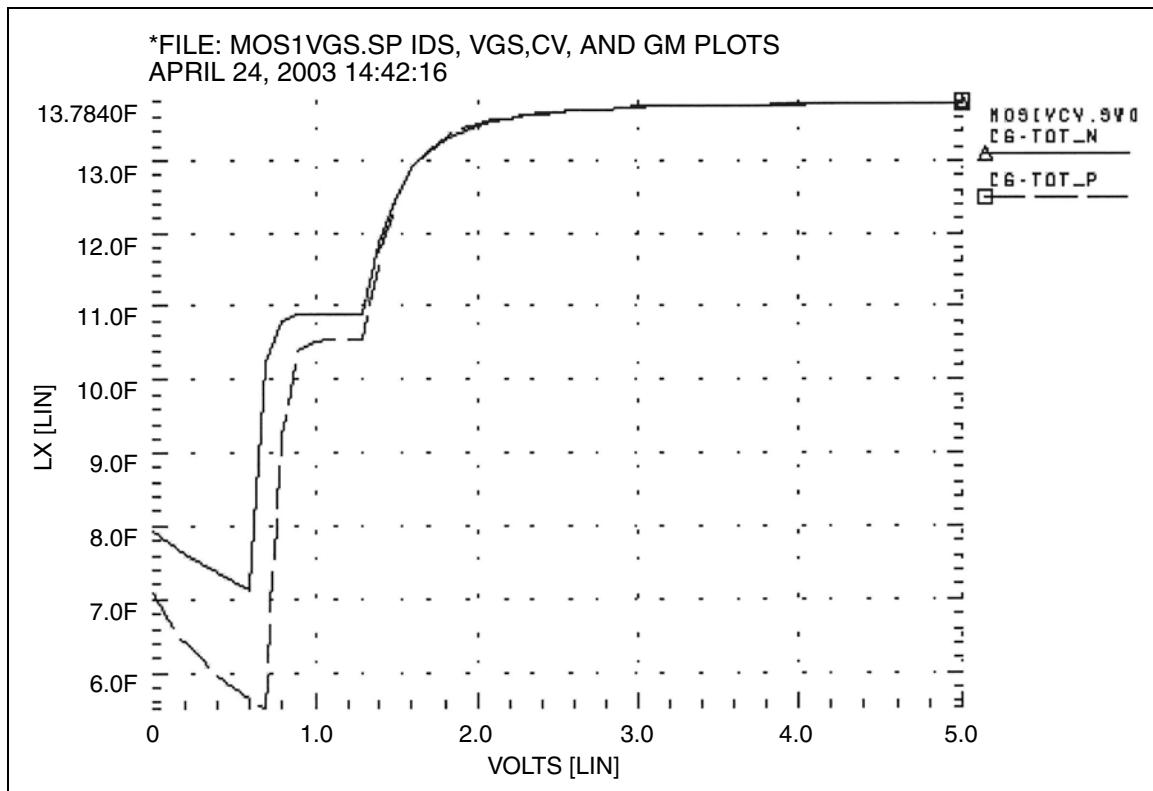


Figure 191 MOS C-V Plot

MOS I-V and C-V Plot Example Input File

You can find the sample netlist for this example in the following directory:

`$installdir/demo/hspice/mos/mosivcv.sp`

CMOS Output Driver Demo

ASIC designers need to integrate high-performance IC parts onto a printed circuit board (PCB). The output driver circuit is critical to system performance. The `$installdir/demo/hspice/apps/asic1.sp` demonstration file shows models for an output driver, the bond wire and leadframe, and a six-inch length of copper transmission line.

This simulation demonstrates how to:

- Define parameters, and measure test outputs.
- Use the LUMP5 macro to input geometric units, and convert them to electrical units.
- Use `.MEASURE` statements to calculate the peak local supply current, voltage drop, and power.
- Measure RMS power, delay, rise times, and fall times.
- Simulate and measure an output driver under load. The load consists of:
 - Bondwire and leadframe inductance.
 - Bondwire and leadframe resistance.
 - Leadframe capacitance.
 - Six inches of 6-mil copper, on an FR-4 printed circuit board.
 - Capacitive load, at the end of the copper wire.

Strategy

The HSPICE strategy is to:

- Create a five-lump transmission line model for the copper wire.
- Create single lumped models for leadframe loads.

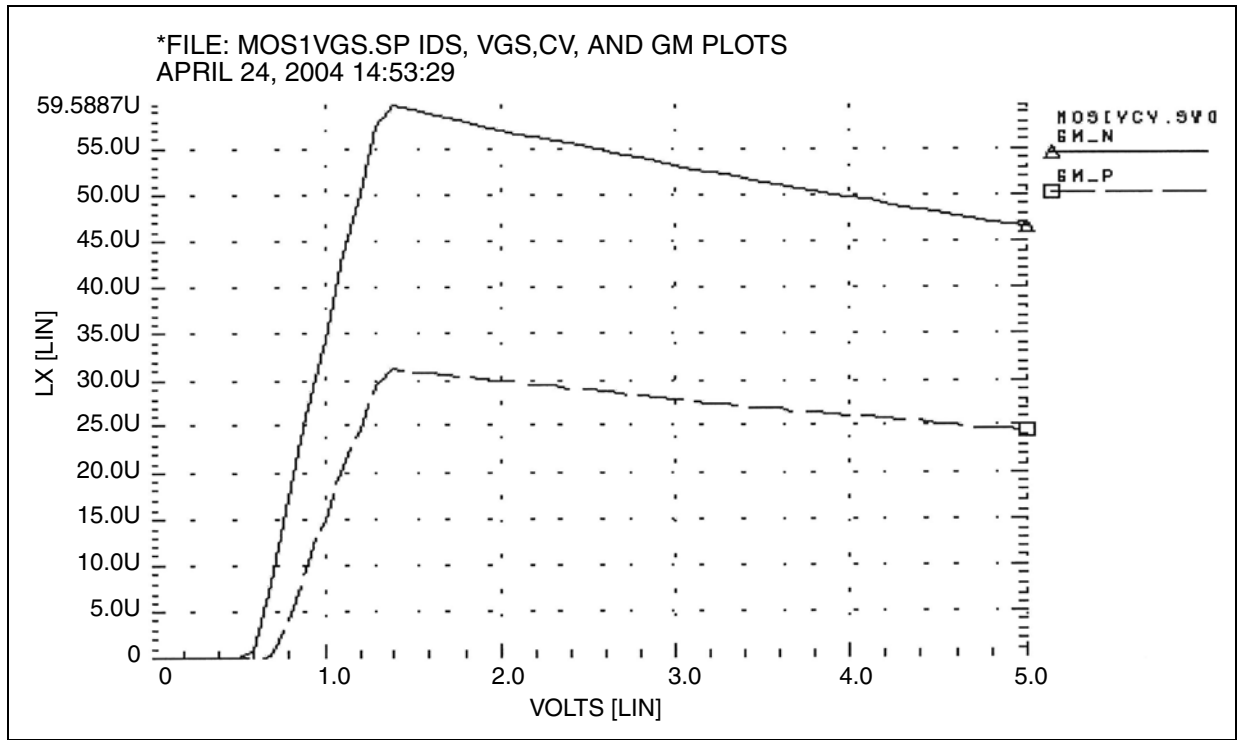


Figure 192 Noise Bounce

Chapter 31: Running Demonstration Files
CMOS Output Driver Demo

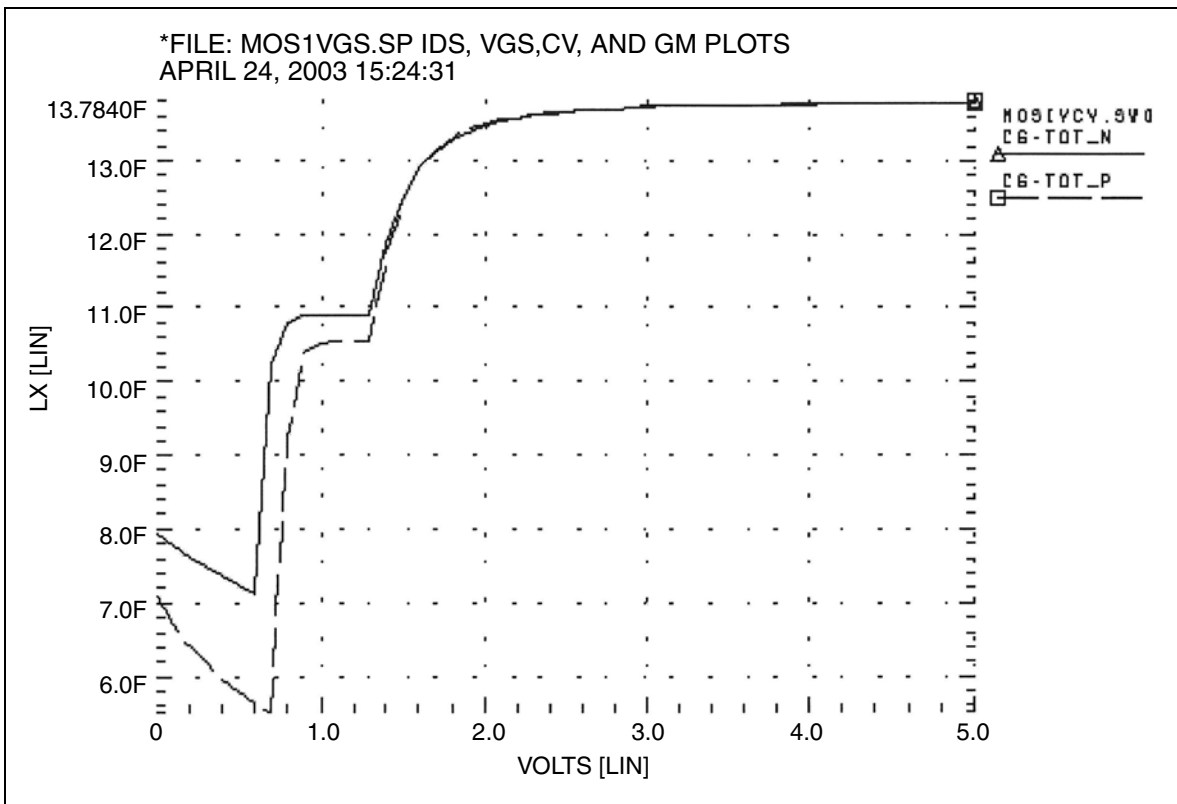


Figure 193 Asic1.sp Demo Local Supply Voltage

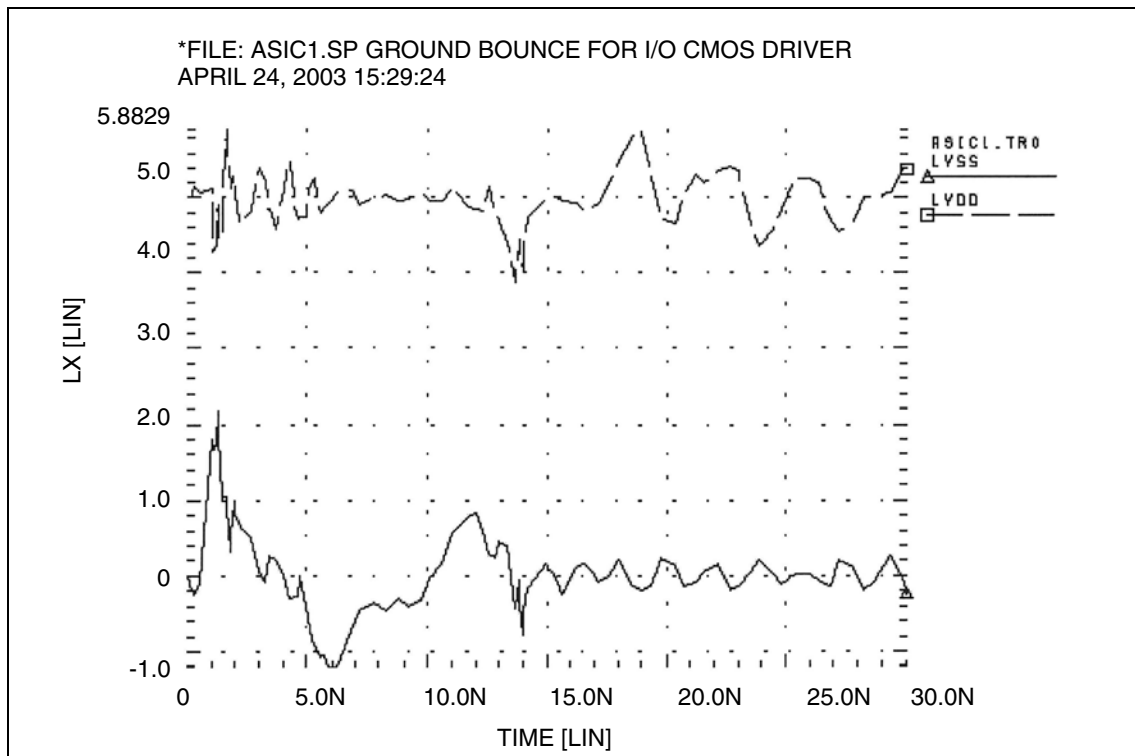


Figure 194 Asic1.sp Demo Local Supply Current

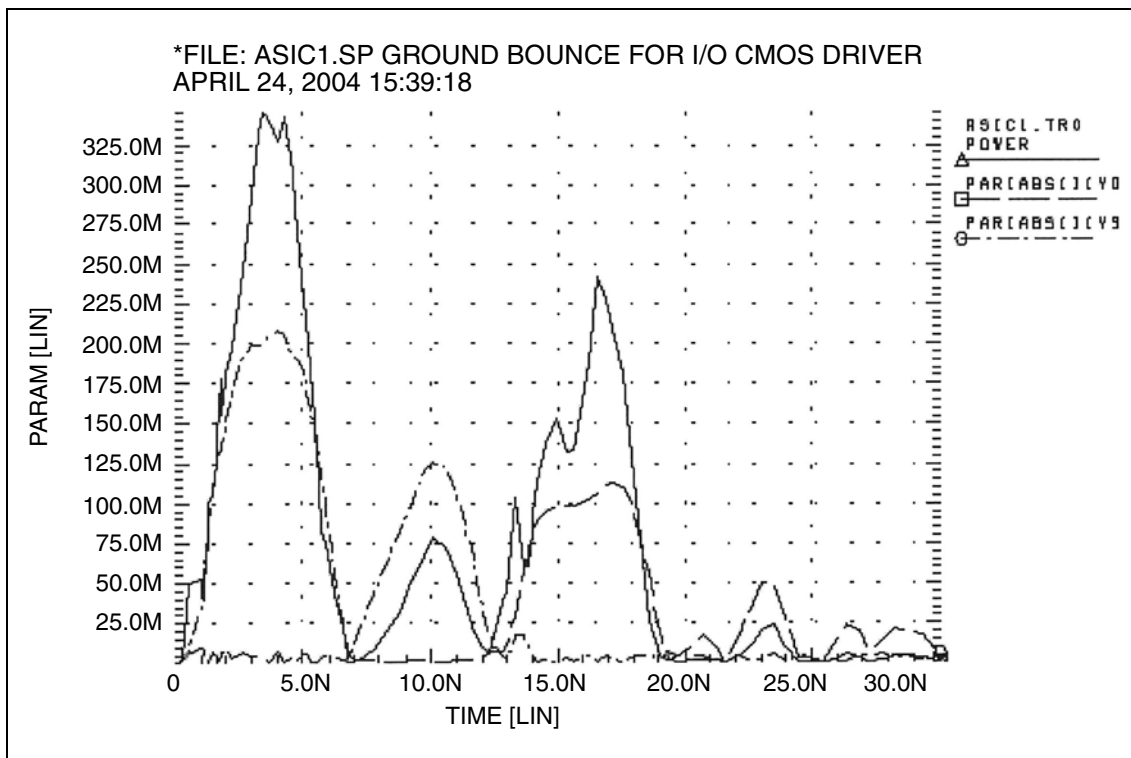


Figure 195 Asic1.sp Demo Input and Output Signals

CMOS Output Driver Example Input File

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/apps/asic1.sp
```

Temperature Coefficients Demo

SPICE-type simulators do not always automatically compensate for variations in temperature. The simulators make many assumptions that are not valid for all technologies. Many of the critical model parameters in HSPICE provide first-order and second-order temperature coefficients, to ensure accurate simulations.

You can optimize these temperature coefficients in either of two ways.

- The first method uses the `TEMP` DC sweep variable.
All analysis sweeps allow two sweep variables. To optimize the temperature coefficients, one of these must be the optimize variable. Sweeping `TEMP` limits the component to a linear element, such as a resistor, inductor, or capacitor.
- The second method uses multiple components at different temperatures.

Example

The following example, the `$installdir/demo/hspice/ciropt/opttemp.sp` demo file, simulates three circuits of a voltage source. It also simulates a resistor at -25, 0, and +25° C from nominal, using the `DTEMP` parameter for element delta temperatures. The resistors share a common model.

You need three temperatures to solve a second-order equation. You can extend this simulation template to a transient simulation of non-linear components (such as bipolar transistors, diodes, and FETs).

This example uses some simulation shortcuts. In the internal output templates for resistors, LV1 (resistor) is the conductance (reciprocal resistance) at the desired temperature.

- You can run optimization in the resistance domain.
- To optimize more complex elements, use the current or voltage domain, with measured sweep data.

The error function expects a sweep on at least two points, so the data statement must include two duplicate points.

Input File for Optimized Temperature Coefficients

You can find the sample netlist for this example in the following directory:

```
$installdir/demo/hspice/ciropt/opttemp.sp
```

Optimization Section

```
.model optmod opt
.dc data=RES_TEMP optimize=opt1
+       results=r@temp1,r@temp2,r@temp3
+       model=optmod
.param tc1r_opt=opt1(.001,-.1,.1)
.param tc2r_opt=opt1(1u,-1m,1m)
.meas r@temp1 err2 par(R_meas_t1) par('1.0 / lv1(r-25)')
.meas r@temp2 err2 par(R_meas_t2) par('1.0 / lv1(r0) ')
.meas r@temp3 err2 par(R_meas_t3) par('1.0 / lv1(r+25) ')
* * Output section *
.dc data=RES_TEMP
.print 'r1_diff'=par('1.0/lv1(r-25)')
+     'r2_diff'=par('1.0/lv1(r0) ')
+     'r3_diff'=par('1.0/lv1(r+25)')
.data RES_TEMP R_meas_t1 R_meas_t2 R_meas_t3
950 1000 1010
950 1000 1010
.enddata
.end
```

Modeling Wide-Channel MOS Transistors

If you select an appropriate model for I/O cell transistors, simulation accuracy improves. For wide-channel devices, model the transistor as a *group* of transistors, connected in parallel, with appropriate RC delay networks. If you model the device as only *one* transistor, the polysilicon gate introduces delay.

When you scale to higher-speed technologies, the area of the polysilicon gate decreases, reducing the gate capacitance. However, if you scale the gate oxide thickness, the capacitance per unit area increases, which also increases the RC product.

Example

The following example illustrates how scaling affects the delay. For example, for a device with:

- Channel width=100 microns.
- Channel length=5 microns.
- Gate oxide thickness=800 Angstroms.

The resulting RC product for the polysilicon gate is:

$$R_{poly} = \frac{W}{L} \cdot 40 \quad poly = \frac{E_{sio} \cdot n_{si}}{tox} \cdot L \cdot W$$

$$R_{poly} = \frac{100}{5} \cdot 40 = 800, \quad C_o = \frac{3.9 \cdot 8.86}{800} \cdot 100 \cdot 5 = 215 fF \quad RC=138 ps$$

For a transistor with:

- Channel width=100 microns.
- Channel length=1.2 microns.
- Gate oxide thickness=250 Angstroms.

The resulting RC product for the polysilicon gate is:

$$R_{poly} = \frac{channel\ width}{channel\ length} \cdot 40$$

$$C_o = \frac{3.9 \cdot 8.86}{Tox} \cdot channel\ width \cdot channel\ length \quad RC=546 ps$$

You can use a nine-stage ladder model to model the RC delay in CMOS devices.

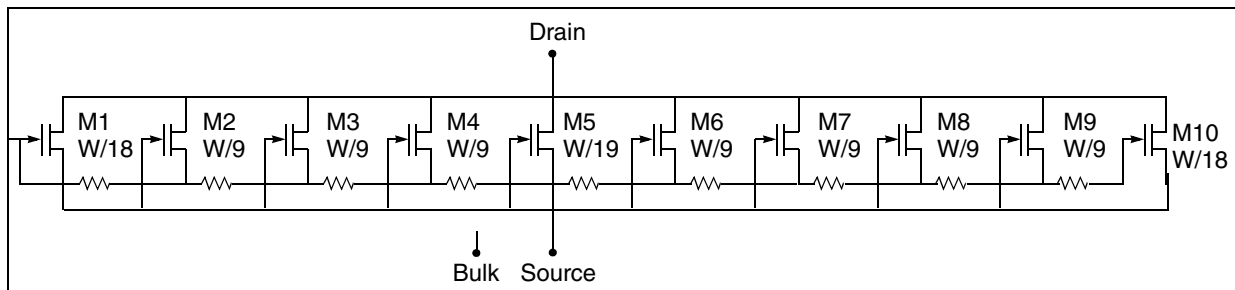


Figure 196 Nine-stage Ladder Model

In this example, the nine-stage ladder model is in data file `$installdir/demo/hspice/apps /asic3.sp`. To optimize this model, HSPICE uses measured data from a wide channel transistor as the target data. Optimization produces a nine-stage ladder model, which matches the timing characteristics of the physical data (HSPICE RF does not support optimization). HSPICE compares the simulation results for the nine-stage ladder model, and the one-stage model by using the nine-stage ladder model as the reference. The one-stage model results are about 10% faster than actual physical data indicates.

Example

Chapter 31: Running Demonstration Files

Listing of Demonstration Input Files

You can find the sample Nine-Stage Ladder model netlist for this example in the following directory:

```
$installdir/demo/hspice/apps/asic3.sp
```

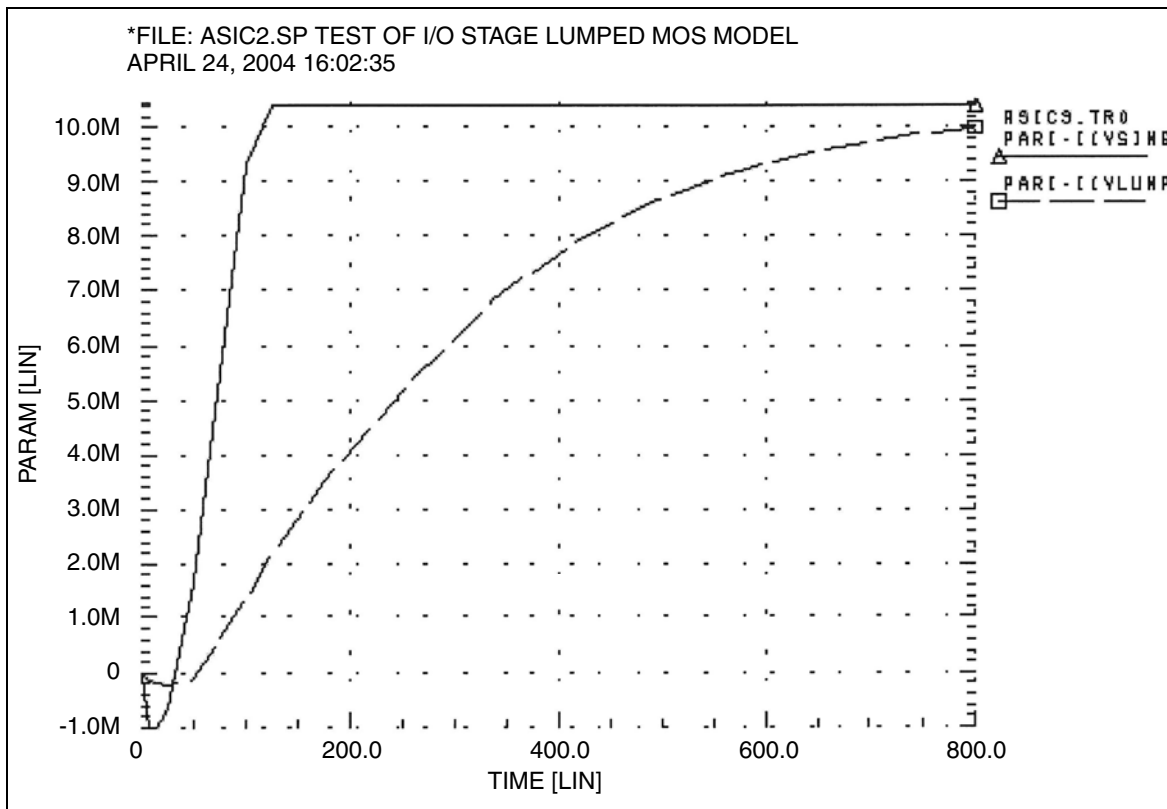


Figure 197 Asic3 Single vs. Lumped Model

Listing of Demonstration Input Files

- [HSPICE Integration to ADE Demonstration Examples](#)
- [Applications of General Interest Examples](#)
- [Back-Annotation Demo Cases](#)
- [Behavioral Application Examples](#)
- [Benchmark Examples](#)

- Bisection-Timing Analysis Examples
- BJT and Diode Examples
- Cell Characterization Examples
- Circuit Optimization Examples
- Device Optimization Examples
- Encryption Examples
- Filters Examples
- Fourier Analysis Examples
- IBIS Examples
- Loop Stability Analysis
- Magnetics Examples
- MOSFET Device Examples
- RF Examples
- Signal Integrity Examples
- Sources Examples
- S-parameter Examples
- Transmission Lines Examples
- Transmission (W-element) Line Examples
- Variability Examples
- Verilog-A Examples

HSPICE Integration to ADE Demonstration Examples

Table 84 HSPICE Integration Tutorial Examples

File Name	Location: \$installdir/demo/hspice/aa_integ/
Corner_Demo_51	Full example of corner analysis
Corner_Demo_61	

Chapter 31: Running Demonstration Files

Listing of Demonstration Input Files

Table 84 HSPICE Integration Tutorial Examples (Continued)

File Name	Location: \$installdir/demo/hspice/aa_integ/
Mosra_Demo_51	Full MOSFET reliability analysis
Mosra_Demo_61	
Optimization_Demo_51	Optimization analysis
Optimization_Demo_61	
Mixer_Demo_61	Netlisting and running of HSPICE RF analyses (hb, hbac, hbnoise, sn, snac, and snoise); PLL Demo_61 complements this demo
Monte_Demo_51	Monte Carlo analysis
Monte_Demo_61	
PLL_Demo_51 PLL_Demo_61	Suites of files for ADE versions 5.1xx and 6.1xx (required to run a guided tutorial found in the first chapter of the HSPICE Integration User Guide, including a Verilog-A example). See Quick-Start Tutorial in the <i>HSPICE Integration to Cadence® Virtuoso® Analog Design Environment User Guide</i>

Applications of General Interest Examples

Table 85 Applications of General Interest Examples

File Name	Location: \$installdir/demo/hspice/apps/
alm124.sp	AC, noise, and transient op-amp analysis
alm124.inc	Macro model
alter2.sp	.ALTER 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

Table 85 Applications of General Interest Examples (Continued)

File Name	Location: \$installdir/demo/hspice/apps/
biaschk.sp	Apply bias check analysis on D flip flop
bjtdiff.sp	BJT diff amp with every analysis type
bjtschmt.sp	Bipolar Schmidt trigger
bjtsense.sp	Bipolar sense amplifier
cellchar.sp	Characteristics of ASIC inverter cell
four.sp	CMOS inverter applied with Fourier analysis
gaasamp.sp	Simple GaAsFET amplifier
gen28.inc	Model library file
grouptim.sp	Group time-delay example
inv.sp	Sweep MOSFET -3 sigma to +3 sigma use .MEASURE output
mcdiff.sp	CMOS differential amplifier
mondc_a.sp	Monte Carlo of MOS diffusion and photolithographic effects
mondc_b.sp	Monte Carlo DC analysis
mont1.sp	Monte Carlo Gaussian, uniform, and limit function
mos2bit.sp	Two-bit MOS adder
noise_app.sp	Uses the .LIN command to do a noise analysis
noise_app_orig.sp	Uses the .NOISE command on the same circuit as above
opampdcm.sp	DCmatch analysis, opamp
quickAC.sp	AC analysis on a RC network
quickINV.sp	Transient analysis on a inverter
quickTRAN.sp	Tran on a resistor divider

Chapter 31: Running Demonstration Files

Listing of Demonstration Input Files

Table 85 Applications of General Interest Examples (Continued)

File Name	Location: \$installdir/demo/hspice/apps/
rc_monte.sp	Transient Monte Carlo on Resistor
sclopass.sp	Switched-capacitor low-pass filter
tlib1	Model library file
tlib2	Model library file
tlib3	Model library file
tlib4	Model library file
trist_buf_opt.sp	Tri-State buffer optimization
wildchar.sp	Wildcard print and probe example
worst.sp	worst-case skew models by using .ALTER
xbjt2bit.sp	BJT NAND gate two-bit binary adder

Back-Annotation Demo Cases

Directories include: top-level netlists, extracted netlists, selected nets file, schematic level netlists, model files, and READMEs.

Table 86 Back-Annotation Cases

Case Name	Location: \$installdir/demo/hspice/back_annotation/
option_ba_activehier/	Demonstrates use of .OPTION BA_ACTIVEHIER to annotate full hierarchical net names that are specified for BA_ACTIVE files.
option_ba_dpfffx/	Demonstrates use of .OPTION BA_DPFFFX to prepend an extra prefix when searching the ideal netlist for instances referenced by the parasitic file.
option_ba_file/dspf/multiba/	Demonstrates use of the .OPTION BA_FILE command to launch multiple DSPF files for parasitic back-annotation.

Table 86 Back-Annotation Cases

Case Name	Location: \$installdir/demo/hspice/back_annotation/
option_ba_file/dspf/	Demonstrates use of the <code>.OPTION BA_FILE</code> command for single DSPF file.
option_ba_file/spef/	Demonstrates use of the <code>.OPTION BA_FILE</code> command for single SPEF file.
option_ba_fingerdelim/	Demonstrates use of <code>.OPTION BA_FINGERDELIM</code> to explicitly specify the delimiter character used for finger devices.
option_ba_hierdelim/	Demonstrates use of <code>.OPTION BA_HIERDELIM</code> to specify the hierarchical separator in the DSPF file.
option_ba_idealpfx/	Demonstrates use of <code>.OPTION BA_IDEALPFX</code> to have the simulator add a second prepended prefix when doing a search of the ideal netlist.
option_ba_mergeport/	Demonstrates use of <code>.OPTION BA_MERGEPORT</code> to control whether to merge net ports into one node.
option_ba_netfmt	Demonstrates use of <code>.OPTION BA_NETFMT</code> to specify the format of the Active Net file (<code>*.rcxt</code> or <code>*.hsimba</code>).
option_ba_print/	Demonstrates use of <code>.OPTION BA_PRINT</code> to control whether to output nodes and resistors/capacitors introduced by back-annotation.
option_ba_terminal/	Demonstrates use of <code>.OPTION BA_TERMINAL</code> to specify the terminal name mapping between the parasitic netlist and the terminal names recognized by the simulator.
selective_ba/ba_acheck/	Demonstrates the use of the <code>.BA_ACHECK</code> command to specify the rule for detecting node activity in back-annotation.
selective_ba/option_ba_active/	Demonstrates use of <code>.OPTION BA_ACTIVE</code> to create an Active Net file for selective back-annotation.
selective_ba/option_ba_coupling/	Demonstrates use of <code>.OPTION BA_COUPLING</code> to control how to treat cutoff coupling capacitors when invoking selective net back-annotation.

Behavioral Application Examples

Table 87 Behavioral Application Examples

File Name	Location: \$installdir/demo/hspice/behave/
acl.sp	Acl gate
amp_mod.sp	Amplitude modulator with pulse waveform carrier
behave.sp	AND/NAND gates by using G-, E-elements AND/NAND gates by using G, E Elements
calg2.sp	Voltage variable capacitance
compar.sp	Behavioral comparator with hysteresis
det_dff.sp	Double edge-triggered flip-flop
diff.sp	Differentiator amplifier and opamp signals
diode.sp	Behavioral diode by using a PWL VCCS
dlatch.sp	CMOS D-latch by using behaviorals
galg1.sp	Sampling a sine wave
idealop.sp	Ninth-order low-pass filter
integ.sp	Integrator circuit
inv_vin_vout.sp	DC sweep of a INV
invb_op.sp	Optimizes the CMOS macromodel inverter
ivx.sp	Characteristics of the PMOS and NMOS as a switch
op_amp.sp	Op-amp from Chua and Lin
pdb.sp	Phase detector by using behavioral NAND gates
pll.sp	PLL build with BJT
pll_bvp.sp	PLL build with behavioral source

Table 87 Behavioral Application Examples

File Name	Location: \$installdir/demo/hspice/behave/
pwl2.sp	PPW-VCCS with a gain of 1 amp/volt
pwl10.sp	Operational amplifier used as a voltage follower
pwl4.sp	Eight-input NAND gate
pwl7.sp	Modeling inverter by using a PWL VCVS
pwl8.sp	Smoothing the triangle waveform by using the PWL CCCS
ring5bm.sp	Five-stage ring oscillator – macromodel CMOS inverter
ringb.sp	Ring oscillator by using behavioral model
rtest.sp	Voltage-controlled resistor, inverter chain
sampling.sp	Sampling a sine wave
swcap5.sp	Fifth-order elliptic switched capacitor filter
switch.sp	Test for PWL switch element
swrc.sp	Switched capacitor RC circuit
vcob.sp	Voltage-controlled oscillator by using PWL functions

Benchmark Examples

Table 88 *Benchmark Examples*

File Name	Location: \$installdir/demo/hspice/bench/
bigmos1.sp	Large MOS simulation
demo.sp	Quick demo file to test installation
example.sp	CMOS amplifier
digstim.vec	Vector stimulus file for m2bit_v.sp
m2bit.sp	72-transistor two-bit adder – typical cell simulation
m2bit_v.sp	Same as m2bit.sp except uses vector stimulus file
senseamp.sp	Bipolar analog test case

Bisection-Timing Analysis Examples

Table 89 *Bisection-Timing Examples*

File Name	Location: \$installdir/demo/hspice/alge/
dff_push.sp	DFF pushout bisection search for setup time
dff_top.sp	DFF bisection search for setup time
fig26_4.sp	Early, Optimal and Late Setup Times of DFF
inv_a.sp	inverter bisection (pass-fail)
tsmc018.m	TSMC model file used by dff_push.sp

BJT and Diode Examples

Table 90 BJT and Diode Device Examples

File Name	Location: \$installdir/demo/hspice/bjt/
bjtbeta.sp	plot BJT beta
bjtgms.sp	plot BJT Gm, Gpi
dpntun.sp	junction tunnel diode
hicum.sp	HICUM BJT MOS terminal characterization
mextram.sp	I-V characteristics of a MEXTRAM BJT
mextram_ac.sp	AC analysis of a MEXTRAM BJT
mextram_dc.sp	DC analysis of a MEXTRAM BJT
mextram_tran.sp	Tran analysis of a MEXTRAM BJT
quasisat.sp	quasisat.sp comparison of bjt Level1 and Level2
self-heat.sp	VBIC BJT with self heating feature
vbic.sp	DC of a VBIC BJT
vbic99_ac.sp	NET analysis of a VBIC99 BJT
vbic99_dc.sp	DC analysis of a VBIC99 BJT
vbic99_tran.sp	TRAN analysis of a VBIC99 BJT

Cell Characterization Examples

Table 91 Cell Characterization Examples

File Name	Location: /\$installdir/demo/hspice/cchar/
diff.sp	.model opt, hemocytoblastic
digin.sp	U-element with digital output

Chapter 31: Running Demonstration Files

Listing of Demonstration Input Files

Table 91 Cell Characterization Examples (Continued)

gen28.inc	level 28 model library used by netlists
inv3.sp	inv3.sp characteristics of an inverter, .model opt, method=passfail
inva.sp	characteristics of an inverter, .model opt, method=passfail
invb.sp	characteristics of an inverter, .model opt, method=bisection
load1.sp	inverter sweep, delay versus fanout
setupbsc.sp	setup characteristics
setupold.sp	setup characteristics, .model opt
setuppas.sp	setup characteristics, .model opt, method=passfail
sigma.sp	sigma.sp sweep MOSFET -3 sigma to +3 sigma by using measure output
tdgtl.a2d	Viewsim A2D HSPICE input file
tdgtl.d2a	Viewsim D2A HSPICE input file
tdgtl.sp	two-bit adder by using D2A Elements

Circuit Optimization Examples

Table 92 *Circuit Optimization Examples*

File Name	Location: \$installdir/demo/hspice/ciropt/
ampgain.sp	Set unity gain frequency of a BJT diff pair
ampopt.sp	Optimize area, power, speed of a MOS amp
asic2.sp	Optimize speed, power of a CMOS output buffer
asic6.sp	Find best width of a CMOS input buffer
delayopt.sp	Optimize group delay of an LCR circuit
lpopt.sp	Match lossy filter to ideal filter
opttemp.sp	Find first and second temperature coefficients of a resistor
rcopt.sp	Optimize speed or power for an RC circuit

Device Optimization Examples

Table 93 *Device Optimization*

File name	Location: \$installdir/demo/hspice/devopt/
beta.sp	LEVEL=2 beta optimization
bjtopt.sp	s-parameter optimization of a 2n6604 BJT
bjtopt1.sp	2n2222 DC optimization
bjtopt2.sp	2n2222 Hfe optimization
d.sp	diode, multiple temperatures
dcopt1.sp	1n3019 diode, I-V and C-V optimization
jopt.sp	300u/1u GaAs FET, DC optimization
ml13opt.sp	MOS LEVEL=2 I-V optimization

Chapter 31: Running Demonstration Files

Listing of Demonstration Input Files

Table 93 Device Optimization (Continued)

File name	Location: \$installdir/demo/hspice/devopt/
ml2opt.sp	MOS LEVEL=3 I-V optimization
opt_bjt.sp	T2N9547 BJT Optimization

Encryption Examples

Table 94 Encryption Examples

File name	Location: \$installdir/demo/hspice/encryption/
8-byte_key.tar	Suite of files demonstrating how to set up an 8-byte_key encryption file.
traditional.tar	Suite of files demonstrating how to set up a traditional (free_lib) encryption file.
triple_DES.tar	Suite of files plus a directory of 2 lib_DES libraries demonstrating how to set up a 3DES encryption file.
README and auxiliary files	

Filters Examples

Table 95 Filters

File Name	Location: \$installdir/demo/hspice/filters/
bandstop1.sp	band reject filter, AC and transient analysis
fbp_1.sp	bandpass LCR filter, measurement
fbp_2.sp	bandpass LCR filter, pole/zero
fbpnet.sp	bandpass LCR filter, using .LIN
fbprlc.sp	LCR AC analysis for resonance
fhp4th.sp	high-pass LCR, fourth-order Butterworth filter, pole-zero analysis
fkerwin.sp	pole/zero analysis of Kerwin's circuit
flp5th.sp	low-pass, fifth-order filter, pole-zero analysis
flp9th.sp	low-pass, ninth-order FNDR, with ideal op-amps, pole-zero analysis

Chapter 31: Running Demonstration Files

Listing of Demonstration Input Files

Table 95 Filters (Continued)

File Name	Location: \$installdir/demo/hspice/filters/
lcline.sp	LC line model using Laplace behavioral elements
low_pass.sp	behavioral model using E and G elements
low_pass9a.sp	active low pass filter using behavioral opamp models
lowloss.sp	RL line model using Laplace behavioral elements
ninth.sp	active low pass filter using Laplace elements
phaseshift.sp	Behavioral model using G table element
rcline.sp	RC line model using Laplace elements

Fourier Analysis Examples

Table 96 *Fast Fourier Transform Examples*

File Name	Location: \$installdir/demo/hspice/fft/
fft5.sp	FFT analysis, data-driven transient analysis
fft6.sp	FFT analysis, sinusoidal source
gauss.sp	FFT analysis, Gaussian window
hamm.sp	FFT analysis, Hamming window
hann.sp	FFT analysis, Hanning window
harris.sp	FFT analysis, Blackman-Harris window
intermod.sp	FFT analysis, intermodulation distortion
kaiser.sp	FFT analysis, Kaiser window
mod.sp	FFT analysis, modulated pulse
pulse.sp	FFT analysis, pulse source
pwl.sp	FFT analysis, piecewise linear source
rect.sp	FFT analysis, rectangular window
rectan.sp	FFT analysis, rectangular window
sffm.sp	FFT analysis, single-frequency FM source
sine.sp	FFT analysis, sinusoidal source
swcap5.sp	FFT analysis, fifth-order elliptic, switched-capacitor filter
tri.sp	FFT analysis, rectangular window
win.sp	FFT analysis, window test
window.sp	FFT analysis, window test
winreal.sp	FFT analysis, window test

IBIS Examples

Table 97 IBIS Modeling Files

File name	Location: \$installdir/demo/hspice/ibis/
at16245.ibs	IBIS model file, used in iob_ex1.sp example file
iob_ex1.sp	Using IBIS buffer example
cmpt1.ibs	IBIS model file
ebd.ebd	IBIS EBD file example
ebd.sp	Using EBD files example
pinmap.ebd	IBIS EBD file example, uses pin mapping
pinmap.sp	Using EBD files example
pinmap.ibs	IBIS model file
readme	readme file for ICM examples
icm/nodepath_rlgc/ bga_1.sp	Using ICM with nodepath description example
bga_example.icm	ICM example file
s_w_test_GHz_db.s4p	TouchStone file for ICM example, called by test1.icm
sect2_s_2.inc	S-parameter model call, used by test1.sp
test1.icm	ICM example file
test1.sp	Using ICM with nodepath description and S-element
sect3_rlgc_4.inc	RLGC file used by test1.sp
sect_w_4.inc	RLGC file used by test1.sp
icm/nodepath_sele/ test1.icm	ICM example file
test1.sp	Using ICM with treepath and rlgc data example
complex.icm	ICM example file
complex.sp	Using ICM with swath matrix expansion

Loop Stability Analysis

Table 98 Loop Stability

File Name	Location: \$installdir/demo/hspice/lstb/
single.sp	.LSTB single-ended mode example for ideal inverting amplifier with single pole

Magnetics Examples

Table 99 Magnetics

File Name	Location: \$installdir/demo/hspice/mag/
aircore.sp	Air-core transformer circuit
bhloop.sp	Magnetic core model, plot B-H loop characteristics
jiles.sp	Effects of core model parameters on B-H loop characteristics
magcore.sp	Magnetic-core transformer circuit
tj2b.sp	Hysteresis effects in magnetic cores
tj_opt.sp	Optimizing magnetic core parameters

MOSFET Device Examples

Table 100 MOSFET Devices

File Name	Location: \$installdir/demo/hspice/mos/
calcap.sp	Calculate AC gate capacitance
calcap.ic0	Results file from calcap.sp
calcap.ic1	Results file from calcap.sp
calcap.lis	Results file from calcap.sp

Chapter 31: Running Demonstration Files

Listing of Demonstration Input Files

Table 100 MOSFET Devices (Continued)

File Name	Location: \$installdir/demo/hspice/mos/
calcap.results	Results file from calcap.sp
calcap.st0	Results file from calcap.sp
capop0.sp	Plot MOS capacitances, LEVEL=2
capop1.sp	Plot MOS capacitances, LEVEL=2
capop2.sp	Plot MOS capacitances, LEVEL=2
cascode.sp	MOS Cascode amplifier example, show effect of level=3 impact ionization parameter
chrgpump.sp	Charge-conservation test, charge pump using LEVEL=3 MOS
gatecap.sp	DC gate capacitance calculation
mcap2_a.sp	MOS charge conservation capacitances
mcap3.sp	MOS charge conservation capacitances
ml13iv.sp	Plot I-V for LEVEL=13
ml13opt.sp	Optimizing MOS LEVEL=13 model parameter
ml27iv.sp	Plot I-V for LEVEL=27 S0SFET
ml5iv.sp	MOS LEVEL=5 example
mosiv.sp	Plot I-V for files that you include
mosivcv.sp	Example of plotting I-V and C-V curves, uses LEVEL=3 model
nch0.inc	MOS model for mosiv.sp and cap_m.sp
selector.sp	Automatic model selector for width and length
ssoi.sp	Floating bulk model
t1.sp	MOS LEVEL=13 TOX calculation test
tempdep.sp	MOS LEVEL=3 temperature dependence

Table 100 MOSFET Devices (Continued)

File Name	Location: \$installdir/demo/hspice/mos/
tgam2.sp	LEVEL=6, gamma model

RF Examples

The following is a listing of shipped demonstration files for illustrating HSPICE RF functionality. (Note the `hspicerf` directory name.)

File Name	Description \$installdir/demo/hspicerf/examples/
acpr.sp	Envelope simulation example
bjt.inc	Transistor model library used by osc.sp
cmos49_model.inc	Transistor model library used by example circuits
cmos90nmWflicker.lib	Transistor model library used by phasefreqdet.sp
gpsvco.sp	Oscillator and Phase Noise analysis example
gsmIna.sp	LNA Linear analysis example
gsmInaIP3_A.sp	3rd order intercept point example
mix_hb.sp	Mixer HB analysis example
mix_hbac.sp	Mixer HBAC analysis example
mix_snac.sp	Mixer Shooting Newton AC example
mix_tran.sp	Mixer transient analysis example
osc.sp	Oscillator tuning curve and phase noise analysis example
pa.sp	Power amplifier HB analysis example
pdfcpGain.sp	Shooting Newton analysis example
phasefreqdet.sp	Shooting Newton and noise analysis example
ringoscSN.sp	Shooting Newton and Phase Noise analysis example

Chapter 31: Running Demonstration Files

Listing of Demonstration Input Files

File Name	Description <i>\$installdir/demo/hspicrf/examples/</i>
tsmc018.m	Transistor model library used by ringoscSN.sp

Signal Integrity Examples

See also [IBIS Examples](#), [S-parameter Examples](#), [Transmission Lines Examples](#), and [Transmission \(W-element\) Line Examples](#).

Table 101 Signal Integrity

File Name	Location: <i>\$installdir/demo/spice/si/</i>
iotran.sp	Signetics I/O buffer with transmission lines example
ipopt.sp	TDR Optimization Example
qa8.sp	Xilinx I/O buffer with transmission lines example
qabounce.sp	Ground bounce example
stateye_ex1.sp	STATEYE example
stateyeAMI	STATEYE example with AMI

Sources Examples

Table 102 Sources

File Name	Location: \$installdir/demo/hspice/sources/
amsrc.sp	Amplitude modulation source example
datadriven_pwl.sp	Data driven PWL source example, somewhat complex as it includes a sweep/tstep/tstop. Both V1 and V2 end up as the separate voltage dc values across the transient, rather than the more common PWL looking waveform.
datamanual_pwl.sp	Simple data pwl example that does not use any sweep or tran tstep/stop syntax. The results are individual voltage waveforms.
eelm.sp	E-element AC source example
exp.sp	Exponential independent source example
prbs.sp	PRBS source example
pulse.sp	Pulse source example
pwl.sp	Repeated piecewise-linear source example
sffm.sp	Single-frequency, FM modulation source example
sin.sp	Sinusoidal source, waveform example
uelm.sp	Digital U-element source example
uelm.d2a	Part of uelem.sp example
vcr1.sp	Switched-capacitor network by using G-switch

S-parameter Examples

Table 103 S-Parameter Examples

File Name	Location: \$installdir/demo/hspice/sparam/
diffamp_s.sp	Mixed mode S-parameter example, differential amplifier; Port element declaration. S-element with mixed mode. The format is FQMODEL.
mixed2p.s4p	Port element declaration. S-element with mixed mode. The parameter format is TOUCHSTONE.
mixedmode_s.sp	Mixed mode S-parameter example, transmission line
sparam.sp	Using S-parameter model in SP model format
spciti.sp	S-element example, calling CITI format S-parameter file
spmod.sp	S-element example, calling Touchstone format S-parameter file
ss_citi.citi	CITI format S-parameter file example
ss_ts.s2p	TouchStone format S-parameter file example

Transmission Lines Examples

Table 104 Transmission Lines (tline) Example Files

File name	Location: \$installdir/demo/hspice/tline/
rcfilt.inc	RC filter macro model
strip1.sp	U-element, two microstrips, in series (8 mil and 16 mil wide)
strip2.sp	U-element, two microstrips, coupled together
stripline.sp	U-element strip line example
uele.sp	U-element, three coupled lines

Transmission (W-element) Line Examples

Table 105 Twline Demo Files

File Name	Location: \$installdir/demo/hspice/twline/
ex1.sp	4 conductor RLGC model W-element example
ex2.sp	4 conductor RLGC file W-element example
ex3.sp	4 conductor W-element using U-element parameters
example.rlc	RLGC file used by ex2.sp
fs_ex1.sp	Field solver, conductor above ground plane
fs_ex2.sp	Field solver, three trace example
fs_ex3.sp	Field solver, coupled line example
fs_ex4.sp	Field solver, Monte Carlo example
petl_ex1.sp	Field solver, 1 conductor coax example
petl_ex2.sp	Field solver, 2 conductor coax example
rlgc.rlc	RLGC file used by rlgc.sp
rlgc.sp	W-element using RLGC file
umodel.sp	4 conductor W-element using U-element parameters

Variability Examples

Table 106 Variation Block, Monte Carlo, and Mismatch Demo Files

File name	Location: \$installdir/demo/hspice/variability/
matrix.sp	Matrix of 9 resistors for testing spatial variation with Monte Carlo

Chapter 31: Running Demonstration Files

Listing of Demonstration Input Files

Table 106 Variation Block, Monte Carlo, and Mismatch Demo Files (Continued)

File name	Location: \$installdir/demo/hspice/variability/
monte_test.tar	Suite of DC test files named <code>test1.sp</code> through <code>test20.sp</code> to test combinations of resistors, subckts, model/instance parameters, etc. See Key to Demonstration Examples for Monte Carlo for discussion of these files.
opampacm.sp	Operational amplifier for ACMAch testing with Variation Block
opampdcm.sp	Operational amplifier for DCMatch testing
opampmc.sp	Operational amplifier for Monte Carlo testing with Variation Block

Verilog-A Examples

Table 107 HSPICE Verilog-A: Netlist and Verilog-A Files

File name	Location: \$installdir/demo/hspice/veriloga/
biterrorate.sp	Bit error rate counter
biterrorate.va	
bjt.sp	BJT model
bjt.va	
colpitts.sp	Colpitts BJT oscillator
colpitts.va	
dac.sp	DAC and ADC
dac.va	
deadband.sp	Deadband amplifier
deadband.va	
ecl.sp	ECL inverter
opamp.sp	Opamp

Table 107 HSPICE Verilog-A: Netlist and Verilog-A Files (Continued)

File name	Location: \$installdir/demo/hspice/veriloga/
opamp.va	
pll.sp	Behavioral model of PLL
pll.va	
resistor.sp	Very simple Verilog-A resistor model
resistor.va	
sample_hold.sp	Sample and hold
sample_hold.va	
sinev.sp	Simple voltage source
sinev.va	

Chapter 31: Running Demonstration Files
Listing of Demonstration Input Files

Warning/Error Messages

Provides an overview of the type of warnings and error messages that HSPICE prints and troubleshooting measures to take when possible.

Users can exercise control over the number of occurrences of warning or error messages and escalate the severity of certain messages by using `.OPTIONS MESSAGE_LIMIT` and `STRICT_CHECK`. See [.OPTION MESSAGE_LIMIT](#) and [.OPTION STRICT_CHECK](#) in the *HSPICE Reference Manual: Commands and Control Options*.

HSPICE ships hundreds of examples for your use; see [Listing of Demonstration Input Files](#) for paths to demo files.

This chapter contains the following topics:

- [Warning Messages](#)
- [Error Messages](#)
- [Analysis Options: DIAGNOSTIC](#)
- [Transient Analysis Errors and Solutions](#)
- [Safe Operating Area \(SOA\) Warnings](#)
- [Verilog-A \(pVA\) Messages](#)
- [Warning Message Index \[10001-10076\]](#)
- [Error Message Index \[20001-20024\]](#)
- [Exit Codes](#)

Warning Messages

The following sections present these topics:

- [Topology Warnings](#)
 - [Model Warnings](#)
 - [Control Option Warnings](#)
 - [Device Warnings](#)
 - [Analysis Warnings](#)
-

Topology Warnings

Note: To suppress the netlist topology checks and cause no topology warnings or errors to be reported set `.OPTION NOTOP`.

Topology Integrity

When HSPICE encounters topology integrity issues, it reports warning messages similar to the four types shown:

```
**warning** only 1 connection at node 1:net0107 defined in subckt
bg: called in element 12:mn0 defined in subckt bg at line 161
within the hspice source, library or include file.
```

```
**warning** both nodes of resistor 1:rinp defined in subckt opa350
are connected. together
```

```
**warning** 2:r11 defined in subckt pwdr resistance limited to
1.000E-05
```

```
**warning** the following singular supplies were terminated to 1
meg resistor
```

supply	node1	node2
vdd18	0:dvdd18 defined in subckt 0	0:0 defined in subckt0
vdd1p8	0:dvdd1p8 defined in subckt 0	0:0 defined in subckt 0

No DC Path to Ground

The warning for no DC path to ground, effective from 2007.09, is:

```
**warning** no dc path to ground from node 13:fl defined in
subckt d****01 now it is connected with gdcpath.
```


Duplicate Initialization

If a node is initialized using a `.ic` or `.nodeset` more than once, the following warning is issued:

```
**warning** a duplicate initialization for node=1620:ram***,  
keeping last value 0.900 only.
```

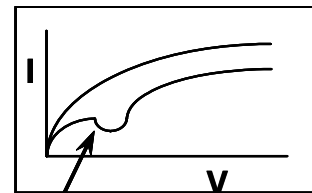
Model Warnings

Zero or Negative Conductance

The following two examples show sample warning messages for negative or zero conductance:

```
**warning** negative-mos conductance = 0:m1 iter= 2  
vds,vgs,vbs =      4.22      2.12      0.925  
gm,gds,gmbs,ids=  1.707E-03  9.366E-05  -1.380E-04  5.040E-04  
  
**warning** conductance of 0. on node 0:net4823 iter= 1
```

The typical causes for these warnings are modeling problems in the subthreshold equations (if in cutoff) or channel length modulation equations (if in saturation). The magnitude reported in the warnings indicates the magnitude of the conductance (leakage) that must be placed across the drain and source to offset the effect of the negative conductance. Typically, `.option GMINDC` and `GMIN` can be used to do this.



Region of Negative Conductance (Negative slope)

Encryption-Related Warnings

```
**warning** Data associated with encrypted blocks were suppressed  
due to encrypted content
```

```
**warning** Some parameters in encrypted block are defined as  
an expression containing output signals. which may cause incorrect  
result. Suggest to use user defined functions to replace.
```

Model Binning Warnings

```
**warning** (L65***.mdl:2) model n_10_rvl device geometries
will not be checked against the limits set by lmin, lmax, wmin
and wmax. To enable this check, add a period(.) to the model name
(i.e. enable model selector).
```

Key Model Parameter Checking

```
*** warning ***: area for diode can not be 0.0, reset to 1e-12
(default value)
```

Parameter Expression Warning

```
**warning** parameter pdt is defined as an expression containing
output signals, which may cause incorrect result. Suggest to use
user defined functions to replace.
```

For example:

```
.Param pdt = "rs*pi*i(node2)+v(out)"
```

Control Option Warnings

RUNLVL

The following is an informational warning about the default RUNLVL option setting.

```
** runlvl is invoked, you can disable it by:
a) Add option runlvl=0 to your current simulation job.
b) Copy $installdir/hspice.ini to your HOME directory and
customize it by adding option runlvl=0, which disables it for all
of your simulation jobs.
c) Re-invoke $installdir/bin/config program and unselect the
option runlvl setting in box 'hspice.ini' which disables it for
whole group simulation jobs.
```

```
** runlvl is invoked, some options are ignored or automatically
set:
```

```
Options below are automatically set (user setting will overwrite
them):
```

```
if runlvl=[1|2|3|4|5|6], .option bypass=2
```

```
Options below are ignored, they are replaced by automated
algorithms:
```

```
lvltim    dvdt      ft      fast    trtol   absvar   relvar
relq      chgtol    dvtr    imin    itl3    rmax
```

** runlvl is invoked, actual option value used by HSPICE are:

runlvl= 3 bypass= 2 mbypass= 2.00 bytol= 100.00u

ACCURATE

***accurate option (accurate=1) sets default value of the options:
lvltim= 3 dvdt= 2 relvar= 200.00m absvar= 200.00m ft= 200.00m
relmos= 10.00m bypass= 2
(used for FFT control) fft_accurate= 1

FAST

warning the fast option set the bypass on and the following options:
dvdt= 3 bytol= 50.00u

GMIN, GMINDC

warning pivtol too large ** reset to half minimum value of (gmindc, gmin)

Device Warnings

Device warnings are specific to each model.

Device Geometry Check

Warning: Pd = 1.36e-06 is less than W.
Model: 0:nch
W = 4.444e-06, L = 5.3e-07

Device Parameter Check

Warning: Moin = 1568.2 is too large.
Warning: Acde = 0.0350921 is too small.

Analysis Warnings

Transient

****warning**** the third value 0.00000D+00 and the fourth value 1.00000D-12 are both smaller than the second value 5.00000D-10, so the transient statement is interpreted as '.tran tstep tstop tstart delmax'.

Example:

```
.tran 1n 1u 0 1p  
.tran 1n 1u 1p 2u
```

Bisection

With option OPTCON=1

```
**warning** endpoints have same sign in bisection  
For x          = 0.0000    , y          = 0.0000    .  
For x          = 1.0000    , y          = 1.0000    .  
Both of these are on the same side of the goal value y =  
0.30000    .
```

Multiple Results:

****warning**** multiple results used in bisection

Example:

```
.tran 1.0e-9 8.0e-9 sweep optimize=opt1 results=y,z  
model=opt_model
```

Pass/Fail

****warning**** passfail does not support more than one result, only first one is validated

Example:

```
.model opt_model opt method=passfail relin=0.01 relout=0.01  
.tran 1.0e-9 8.0e-9 sweep optimize=opt1 results=y,z  
model=opt_model
```

Measure

****warning**** measure results may be incorrect since initial start time is non-zero.

```
vin_pp= 1.4855E-01 from= 1.5000E-05 to= 2.0000E-05
```

****warning**** the Equation Evaluation form of the .MEASURE statement must not be a function of node voltages or branch currents. Unexpected results may incur.

Example:

```
.MEAS VARG PARAM='(V(2) + V(3))/2'
```

.DC and .OP Analysis Warnings

When both DC and TRAN source are defined:

****warning**** dc voltage reset to initial transient source value in source 0:vclk new dc= 0.0000D+00

Example:

```
vlo2 in gnd dc vhaf sin(0 '(pwr(10,((toin)/20)))*(1e-6)*SQ2' fq  
0 0 180)
```

Character line limit warning

The HSPICE line limit is 1024 characters.

****warning**** node full pathname length in .ic file greater than limit, node NOT initialized in the save file nodeset.ic

Autoconverge overflow message

****warning**** Due to a floating point overflow problem, the damped pseudo-tran method was used. Also, gmindc was set to 1.0000E-11

Difficult operating point calculation warning message

****warning**** This was a difficult operating point. You can speed up your simulation by specifying:.OPTION CONVERGE=4

Auto-convergence flow messages

convergence problems in dc sweep curves at 15.894 resimulating with dc convergence controls

Chapter 32: Warning/Error Messages

Error Messages

```
**diagnostic** dc convergence failure, resetting dcon option to  
1 and retrying with dcon=1, it converged for gmindc= 5.500E-14
```

```
**diagnostic** dc convergence failure,  
resetting dcon option to 2 and retrying.
```

```
**diagnostic** although this circuit has failed to converge to  
gmindc= 1.000E-15, it did converge to a gmindc= 7.662E-15 for  
most circuits a value of gmindc 1e-7 or less, is acceptable
```

Operating point diagnostic failure messages

```
**diagnostic** number of iteration exceeds min (7000,  
20*itl1)=7000 in pseudo tran process (converge=1 process). Usually  
this happens when the models are discontinuous, or there are  
uninitialized bi-stable cells (flip-flop) in the circuit. By  
setting options dcon=-1 and converge=-1 you can disable auto  
convergence process. Retry the run, non-convergence diagnostics  
will provide useful information about the nodes and devices which  
can be used to work around the non-convergence problems.
```

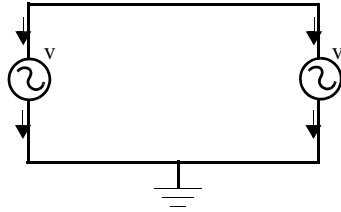
Error Messages

The following sections present these topics:

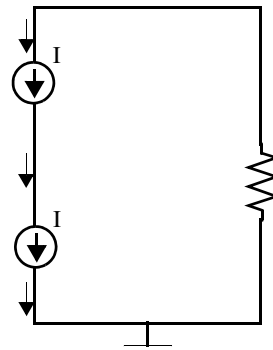
- [Topology Errors](#)
- [Model Errors](#)
- [Analysis Errors](#)

Topology Errors

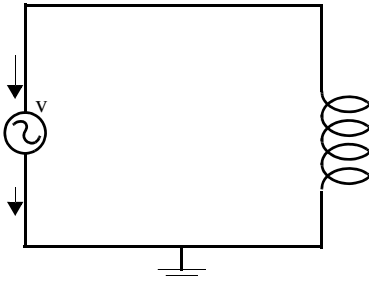
When constructing the circuit description HSPICE does not allow certain topologies. Topology errors will be reported according the following circumstances:



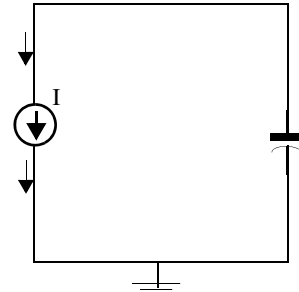
No voltage loops: no voltage sources in parallel with no other elements



No stacked current sources: no current sources in series



No ideal voltage source in closed inductor loop



No ideal current source in closed capacitor loop

Negative or 0 multiplier is not allowed

```
**error** Value of multiplier parameter in 1:x1 is less than or equal to 0
```

Model Errors

Undefined Model

```
**error** model name pch in the element 0:mp is not defined.
```

Redundant Model Definition

```
**error** above line attempts to redefine tnl
**error** (./models/res2:4) difficulty in reading input
```

Undefined parameter

```
**error** no definition for 0:rsit was called by 0:rin
**error** no definition for 0:toxh it was called by 0:n
```

Analysis Errors

.DC and Operating Convergence Errors

No convergence error

```
**error** no convergence in operating point
```

No convergence at a .DC sweep point

```
**error** no convergence in dc sweep curves at 15.851
```

Operating Point Debugging Information

```
*** hspice diagnostic *** nonconvergent voltage failures= 33803  
nonconvergent element current failures= 1
```

...

Convergence Termination Criteria

```
/* NC is the # of non-convergent nodes, currents, or MOSFETs
```

```
Another Iteration: Iteration_Number = Iteration_Number + 1
```

```
NC=0
```

```
Do I=1, # Circuit Nodes
```

```
  if (| V(n) - V(n-1) | > RELV * V(n) + ABSV)
```

```
  then NC = NC + 1
```

```
Do I=1, # Branch Currents
```

```
  if (| I(n) - I(n-1) | > RELI * I(n) + ABSI)
```

```
  then NC = NC + 1
```

```
Do I=1, # MOSFETs
```

```
  if (| Ids(n) - Ids(n-1) | > RELMOS * Ids(n) + ABSMOS
```

```
  then NC = NC + 1
```

```
IF NC = 0
```

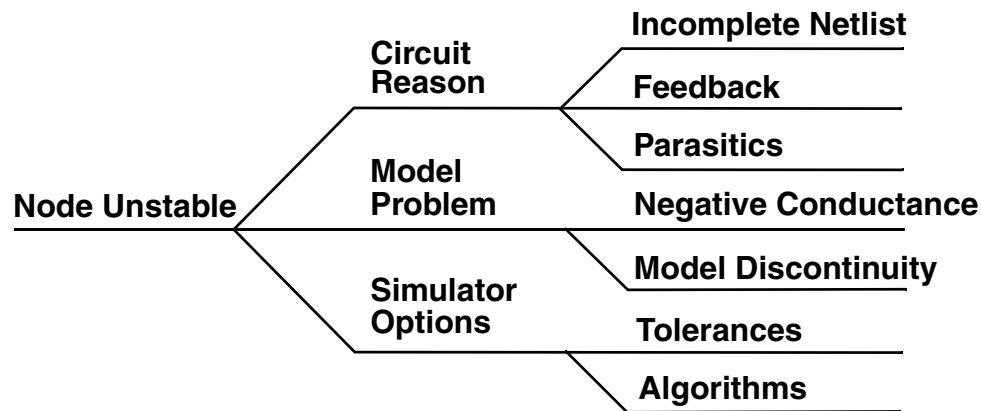
```
  Save Solution
```

```
else
```

```
  If (Iteration_Number < Iteration_Limit) do Another_Iteration
```

```
  else Failed_to_Converge
```


Non-Convergence: Possible Causes



DC/OP Convergence Aids

Aids to Remedy DC Bias Non-Convergence

- Auto-convergence process
- .NODESET/.IC Commands (see the following sections)
- Model-related solutions
- Others, with less impact
 - DCSTEP and GMINDC ramping
 - Source stepping/ramping
 - GSHUNT/CSHDC
 - DV

DC Bias Point Convergence Actions

Take the following actions to resolve issues dealing with DC bias point non-convergence:

- Remove all options *except* `node`, `list`, `post`, and `opts`
- Allow the auto converge process to proceed
- Review the `.lis` file for convergence hints
- Search for “warning” and “error” messages
- Rerun the simulation

DC Bias Point Troubleshooting with .NODESET and .IC

Non-convergence can occur due to poor initial conditions. Set initial conditions and/or nodesets. For example:

```
.IC v(1)=5v v(abc)=0v v(12)=VDD
```

```
.NODESET v(x1.87)=5v
```

Identify the problem nodes by:

- Reviewing the non-convergent diagnostic table in the listing file
- Identifying non-convergent nodes with unusually high voltages, branch currents, or high error tolerances
- Initializing these nodes
- Reviewing the circuit for un-initialized feedback paths (flip-flops, oscillators, etc.)

Because it is inefficient to manually add .NODESET and/or .IC for a large number of nodes, to set a large number of nodes:

1. Comment out all analysis commands except .TRAN.
2. Add UIC to the end of the .TRAN command.
3. Disable auto-convergence process by setting:

```
.option DCON=-1 CONVERGE=-1
```
4. Use .SAVE [TYPE=<nodeset|ic>] [TIME=<x>] to store the calculated operating point as a .ic or .nodeset file.
5. Simulate the circuit.
6. Use .LOAD for loading the file from the .SAVE command.
7. Enable the auto-converge process by removing the DCON and CONVERGE options.
8. Remove UIC from the .TRAN command.
9. Re-simulate the circuit.

Troubleshooting Model-Related DC Bias Point Issues

Inappropriate model parameters are usually the cause having to do with units or negative/zero conductance:

- If the issue is units:
 - .OPTION SCALM (global)

- .MODEL SCALM factor (local value within .MODEL statements)

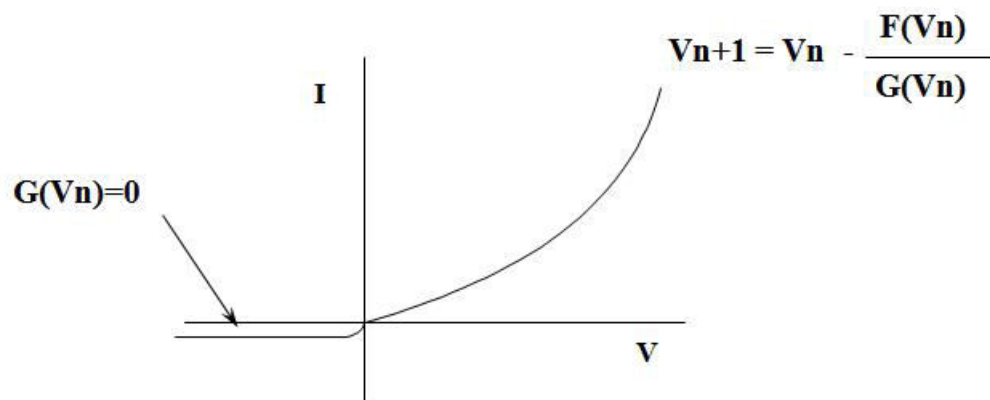
Is a global .OPTION SCALM needed?

- Look at the listing file and review element values.
- Swap in a known good model.

Convergence/Conductance

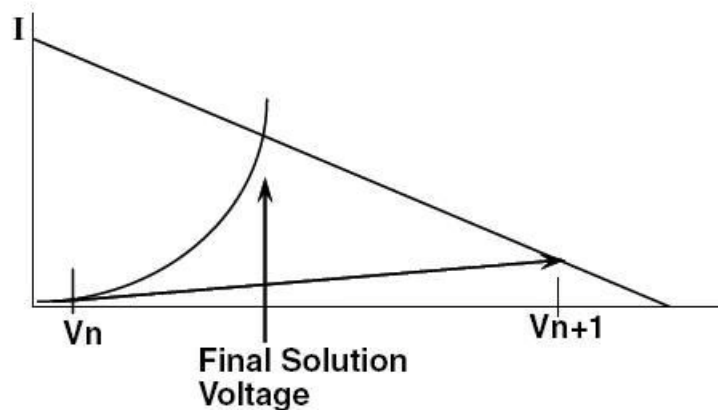
This section describes issues and possible solutions when conductance values impact on convergence. For example: Using a conductance term to predict the next voltage value, can create the problem that if conductance becomes small, the 2nd term becomes large:

- Next voltage value unrealistic
- Causes extra iterations
- Worse: Conductance of zero!



- Since all semiconductor device models contain regions of zero conductance:
 - Shunt R placed in parallel with every PN junction and drain to source
 - Determine smallest parasitic Rp that can be placed across any 2 nodes without influencing circuit behavior
 - $G=1/R_p$
 - Try setting .OPTION GMINDC=1e-9 GMIN=1e-9
 - Default for both GMIN and GMINDC is 1e-12
- You must ask, “How much leakage is acceptable?”
 - Typically, a setting GMIN=1e-10 does not affect CMOS circuit accuracy

- Larger values of GMIN will affect accuracy and indicate that there may be a model problem



Convergence/Diode Resistance

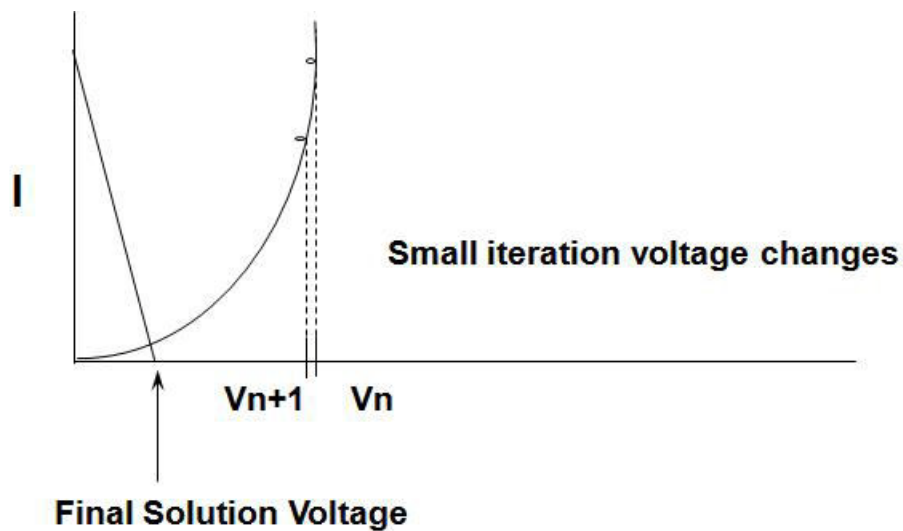
High conductance is troublesome to the algorithm:

Problem: Highly forward-biased diodes (greater than 0.8V)

- Lead to very small iteration-to-iteration voltage changes
- Can cause HSPICE to reach iteration limit before reaching the proper solution voltage

Solution:

- Always specify the series-resistance model parameter for all diodes, bipolar devices, and MOSFETs in the circuit (Default is ZERO ohms).
- At high forward bias, the series resistance dominates the conductance of the device and helps reduce the occurrence of non-convergence.



- Series resistance model parameters for active devices:
 - Diode R_S
 - Bipolar transistor R_E and R_C
 - JFET R_D and R_S
 - MOSFET R_D and R_S

Analysis Options: DIAGNOSTIC

HSPICE automatically prints out the first occurrence of “negative-mos conductance” in the `.lis` file.

`.option DIAGNOSTIC`

- Causes all occurrences of negative model conductances to be printed in the `.lis` file
- If the magnitude of the negative conductance is $> -1e-8$, consult your modeling department or foundry

Transient Analysis Errors and Solutions

The following section discuss these topics:

- [Transient Analysis Error](#)
- [Transient Non-Convergence](#)
- [Transient Convergence Aids](#)

Transient Analysis Error

- Most frequent error message:

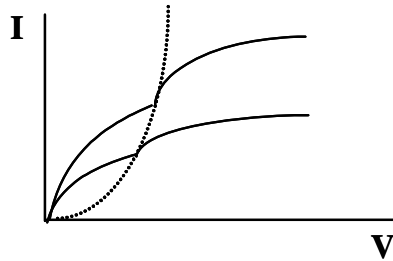
```
**error**  internal timestep too small in transient analysis
```
- Occurs when: $\text{Internal timestep} < \text{RMIN} * \text{TSTEP}$
TSTEP is from .TRAN statement.

Transient Non-Convergence

Rapid Voltage Transitions:

- Dynamic timestep control automatically reduces the timestep size
- As the circuit approaches a voltage transition, two potentially conflicting events occur:
 - Semiconductor devices are switching from one region of operation to another.
 - Timestep is reduced.

Model Discontinuities:



- Separate equations used for the different operating regions of an active device
- Most common discontinuities are at the intersection of the linear and saturation regions
- Failure Mechanism
 - Newton-Raphson can oscillate back and forth across the discontinuity
 - Oscillations use up iterations without progressing toward a solution
 - A sweep increases likelihood of hitting model discontinuities

Transient Convergence Aids

Corrective actions include:

- Device model capacitance
- GEAR integration
- Use RUNLVL option

Device model capacitance—Transient non-convergence is primarily caused by a combination of model discontinuities and a reduced step size brought on by voltage transitions within the circuit:

- All simulation models should have their associated capacitance terms set to a non-zero value.
- Real models have real capacitances.
- Capacitive Model Parameters:
 - Diode CJO
 - Bipolar CJE, CJC, CJS
 - JFET CGD, CGS

- MOSFET CGDO, CGSO, CGDO, CBD, CBS, CJ, CJSW

GEAR Integration:

- Numeric integration of time varying currents and voltages are accomplished through Trapezoidal and Gear linearization.
- GEAR integration acts as a filter, removing oscillations that can occur due to the trapezoidal algorithm.
- Circuits that are non-convergent with TRAP will often converge with GEAR.

Why GEAR sometimes converges where TRAP fails:

- Gear integration uses a “weighted average” of past timesteps to determine the next time step.
- This past history helps to project over model discontinuities that exist.

Use of the RUNLVL option:

- This option has an enhanced convergence algorithm providing less chance of encountering “time step too small” error.
- RUNLVL=3 (default setting) is similar to default HSPICE setting.
- RUNLVL=5 is similar to setting ACCURATE option.

Safe Operating Area (SOA) Warnings

You can set `.option warn` and `.option maxwarns` to have HSPICE issue warnings when terminal voltages of a device (MOSFET, HV, BJT, diode, capacitor, resistor, etc.) exceed the SOA (Safe Operating Area). All warning message parameters (e.g., `Bv_max`, `Vbe_max`, etc.) are positive with default value of infinity.

See the following control options for details:

- [.OPTION WARN](#)
- [.OPTION MAXWARNS](#)

A listing of the checking criteria follows.

MOSFET:

BSIM4, PSP, HiSIM_HV, BSIM3, BSIM3SOI, BSIM4SOI, HVMOS. Terminal voltages checked: Vgs, Vgd, Vgb, Vds, Vbs, and Vbd.

Terminal Voltages Checked	Model Params	Check criteria (i.e., the device under checking is called M1) and Warning Issued
Vgs	Vgs_max	if $ V_{gs} > V_{gs_max}$: "Vgs (=xxx) of M1 has exceeded Vgs_max (=yyy)".
Vgd	Vgd_max	if $ V_{gd} > V_{gd_max}$: Vgd (=xxx) of M1 has exceeded Vgd_max (=yyy).
Vgb	Vgb_max	If Vgb_max is not given if $ V_{gb} > V_{gs_max}$: "Vgb (=xxx) of M1 has exceeded Vgs_max (=yyy)". Else if $ V_{gb} > V_{gb_max}$: "Vgb (=xx x) of M1 has exceeded Vgb_max (=yyy)".
Vds	Vds_max	if $ V_{ds} > V_{ds_max}$: "Vds (=xxx) of M1 has exceeded Vds_max (=yyy)".
Vbs	Vbs_max	If Vbs_max is not given if $ V_{bs} > V_{bd_max}$: Vbs (=xxx) of M1 has exceeded Vbd_max (=yyy). Else if $ V_{bs} > V_{bs_max}$: Vbs (=xxx) of M1 has exceeded Vbs_max (=yyy).
Vbd	Vbd_max	if $ V_{bd} > V_{bd_max}$: "Warning: Vbd (=xxx) of M1 has exceeded Vbd_max (=yyy)".

Note: Special considerations:

1. If Vgb_max is not given, Vgs and Vgb share the same model warning parameter: "Vgs_max."
2. Vgd has a different warning parameter for HV device considerations.
3. If Vbs_max is not given, Vbs and Vbd share the same model warning parameter "Vbd_max."

Chapter 32: Warning/Error Messages
 Safe Operating Area (SOA) Warnings

BJT:

GP. Other BJT models may be included.

Terminal Voltages Checked	Model Params	Check criteria (i.e., the device under checking is called Q1) and Warning Issued
Vbe	Vbe_max	if $ V_{be} > V_{be_max}$: Vbe (=xxx) of Q1 has exceeded Vbe_max (=yyy).
Vbc	Vbc_max	if $ V_{bc} > V_{bc_max}$, issue "Warning: Vbc (=xxx) of Q1 has exceeded Vbc_max (=yyy).
Vce	Vce_max	if $ V_{ce} > V_{ce_max}$: Vce (=xxx) of Q1 has exceeded Vce_max (=yyy).
Vcs	Vcs_max	if $ V_{cs} > V_{cs_max}$: Vcs (=xxx) of Q1 has exceeded Vcs_max (=yyy).

Diodes (Including Zener and Schottky)

Terminal Voltages Checked	Model Params	Check criteria (i.e., the device under checking is called D1) and Warning Issued
Vj (from N to P)	Bv_max	if $ V_j > Bv_max$: Vj (=xxx) of D1 has exceeded Bv_max (=yyy).
Vf (forward)	Fv_max	if $ V_f > Fv_max$: Vf (=xxx) of D1 has exceeded Fv_max (=yyy).

Resistor (needed for both model and instance, instance Bv-max overrides model)

Terminal Voltages Checked	Model Params	Check criteria (i.e., the device under checking is called D1) and Warning Issued
Vr	Bv_max	if $ V_r > Bv_max$: Vr (=xxx) of R1 has exceeded Bv_max (=yyy).

Capacitor (needed for both model and instance, instance Bv-max overrides model)

Terminal Voltages Checked	Model Params	Check criteria (i.e., the device under checking is called C1) and Warning Issued
Vc	Bv_max	if $ V_c > Bv_max$: Vc (=xxx) of C1 has exceeded Bv_max (=yyy).

Verilog-A (pVA) Messages

When compiling a Verilog-A module using the pVA compiler, you may see pvaI, pvaW, pvaE, or pvaNIY messages.

These messages have the following meanings:

- pvaI: Informational message that has no effect on compilation and simulation.
- pvaW: Warning message that possibly could affect compilation and simulation.
- pvaE: Error detected by pVA. The compilation will be aborted.
- pvaNIY: A “Not Implemented Yet” message for Verilog-A functions.

These messages provide useful information and help in debugging the Verilog-A module. For example:

```
*pvaI* ##### Total 131 line-size(s), 29 expr(s), 2 contr(s), 4
init(s), 4 behav(s), 2 port(s)
*pvaW* macro `P_Q redefined at (constants.vams:34)
```

Topology Check Warning

In cases where the competing voltage sources have the same value the following warning message is written to the *.lis file.

```
**warning** inductor/voltage loop found
Instance from to
0:x1      n1  0
0:v2      n1  0
```

Warning Message Index [10001-10076]

The following table lists an index of warning messages that can be encountered in HSPICE usage. The table is organized according to the index number, netlist line, sample warning message, and example of what triggered the warning.

Index	**warning** Message and Example
10001	(10001.sp:4) No independent source value specified. Reset to zero. Example: v1 1 0

Chapter 32: Warning/Error Messages

Warning Message Index [10001-10076]

Index	**warning** Message and Example
10002	(10002.sp:5) No resistance value specified. Reset to the value of resmin. Example: R1 1 2
10003	(10003.sp:11) No capacitance value specified. Reset to zero Example: C11 1 2
10004	(10004.sp:21) No inductance value specified. Reset to 1e-12. Example: L1 3 2
10005	(10005.sp:8) Syntax error while using .print/.probe, missing output variable. Line ignored. Example: .probe \$no output variable is specified.
10006	(10006.sp:9) Missing parameters for .ic. Enter parameters for initial condition with their respective values; Line ignored. Example: .ic
10007	(10007.sp:12) Model nch device geometries will not be checked against the limits set by lmin, lmax, wmin and wmax. To enable this check, add a period(.) to the model name(i.e. enable model selector). Example: .model nch nmos level=1 lmin=1n lmax=2n
10008	(10008.sp:32) Fundamental frequency cannot be zero or negative for fourier analysis. Specify a positive frequency value; Fourier analysis ignored. Example: .four 0 v(6)
10009	(10009.sp:32) Number of repeating operation (R) for PAT voltage source v should be integer greater than or equal to -1.Reset to default(R=0). Example: v 1 0 PAT (5 1 1n 0.5n 0.5n 5n b1011 r=-2 rb=1 b01m1z)
10010	(10010.sp:32) Number to specify the starting bit of repeating operation (RB) for PAT voltage source v cannot be less than 1. Please enter a positive number. Reset to default (RB= 1). Example: v 1 0 PAT (5 1 1n 0.5n 0.5n 5n b1011 r=-1 rb=0 b01m1z)

Index	**warning** Message and Example
10011	<p>(10011.sp:4) First TAP value must be largest of all TAP values for Pseudo Random Bit Generator source v. Assumes a descending order sort.</p> <p>Example:</p> <pre>.param a=4 vlow=1 vhigh=5 tdelay=1n trise=0.5n tfall=0.5n + rate=0.1g seed=10 rout=10k v 1 0 vlow LFSR (vlow vhigh tdelay trise tfall rate seed [2,5] + rout)</pre>
10012	<p>(10012.sp:11) Mutual inductance value has not been specified for mutual inductor. Reset to zero. Enter value of mutual inductance coefficient.</p> <p>Example:</p> <pre>k1 l1 l2 l3 tsat MAG=2 \$coefficient is not specified.</pre>
10013	<p>(10013.sp:14) Magnetization (MAG) of mutual inductor can only be -1 0 1, assumes the value is -1. Enter a valid number for MAG.</p> <p>Example:</p> <pre>k1 l1 l2 l3 tsat MAG=2 .model tsat L(ac=1e4 lc=100 hc=.1 tc=1u br=6.4k bs=6.75k hs=.6 + hcr=0)</pre>
10014	<p>(10014.sp:3) Attempt to reference undefined pin x1.mid1 in isub(); branch output ignored. Specify a valid pin name.</p> <p>Example:</p> <pre>x1 1 0 aa .subckt aa in out r1 in mid aa .probe tran isub(x1.mid1) .ends</pre>

Chapter 32: Warning/Error Messages

Warning Message Index [10001-10076]

Index	**warning** Message and Example
10015	(10015.sp:13) Attempt to reference undefined pin x1.1; biaschk output ignored. Specify a valid pin name. Example: x1 1 0 aa .subckt aa in out r1 in mid aa .ends .biaschk subckt terminall=1 simulation=tran monitor=i max=1 min=0.1 sname=x1
10016	(10016.sp:3) ISUB() unsupported for top-level node 1; branch output ignored. Example: .probe tran isub(1)
10017	(10017.sp:12) Unable to find referenced node 100; Output variable ignored. Specify a valid node. Example: .probe tran v(100)
10018	(10018.sp:8) Inductance for the inductor lout >= 0.1 henry, please verify it. Example: lout out 0 1
10019	(10019.sp:17) Skin Effect Coefficient parameter (Rs) for the element rout cannot be negative. Parameter has been ignored. Enter a valid Rs value. Example: rout out 0 1K rs=-1
10020	(10020.sp:6) Frequency "FMAX" cannot be negative for frequency dependent resistor r11. Parameter has been ignored. Enter positive value of "FMAX". Example: R11 1 2 1 Rs=1 FMAX=-100 FBASE=10 CONVOLUTION=1
10021	(10021.sp:6) Frequency "FBASE" cannot be negative for frequency dependent resistor r11. Parameter has been ignored. Enter positive value for "FBASE". Example: R11 1 2 1 Rs=1 FMAX=100 FBASE=-10 CONVOLUTION=1
10022	(10022.sp:6) "CONVOLUTION" can ONLY have value of 0 1 2 for frequency dependent resistor r11. Reset to default(CONVOLUTION=0). Enter valid value of CONVOLUTION. Example: R11 1 2 1 Rs=1 FMAX=100 FBASE=-10 CONVOLUTION=1

Index	**warning** Message and Example
10023	(10023.sp:3) Propagation Delay (TD) cannot be negative for dependent source e11. Reset it to default (TD=0). Enter positive value of "TD". Example: E11 out 0 VCVS DELAY 1 0 td=-1n
10024	(10024.sp:6) "FBASE" cannot be > "FMAX" for element r11. Parameters have been ignored. Enter FBASE value < FMAX. Example: R11 1 2 1 Rs=1 FMAX=1k FBASE=1MEG CONVOLUTION=11
10025	(10025.sp:8) Scaling parameter (SCALE) for the x1.r1 cannot be < or = zero. Parameter has been ignored. Enter a valid value of "SCALE". Example: r1 in mid aa scale=0
10026	(10026.sp:5) Frequency (freq) of sine voltage source v1 cannot be negative. Reset to freq= "1/TSTOP". Enter a positive value. Example: v1 1 0 5 sin (5 1 -1g 0.5n 0.2 60)
10027	(10027.sp:5) Damping factor (theta) for sine voltage source v1 cannot be negative. Please enter a positive value for "theta". Reset to default (theta=0). Example: v1 1 0 5 sin (5 1 1g 0.5n -0.2 60)
10028	(10028.sp:4) Rise delay time for exponential voltage source v cannot be negative. Reset to default (rise delay time=0). Enter positive rise delay time value. Example: v 1 0 v0 exp(4 1 -2n 30n 40n 80n)
10029	(10029.sp:4) Fall delay time for exponential voltage source v cannot be negative. Reset to default (fall delay time= Rise delay time +TSTEP). Enter positive fall delay time value. Example: v 1 0 v0 exp(4 1 2n 30n -40n 80n)
10030	(10030.sp:4) Rise time constant for exponential voltage source v cannot be negative. Reset to default (rise time constant=TSTEP). Enter positive rise time constant value. Example: v 1 0 v0 exp(4 1 2n -30n 40n 80n)
10031	(10031.sp:4) Fall time constant for exponential voltage source v cannot be negative. Reset to default (fall time constant=TSTEP). Enter positive fall time constant value. Example: v 1 0 v0 exp(4 1 2n 30n 40n -80n)
10032	(10032.sp:4) Frequency cannot be negative for FM voltage source v. Reset to default (freq= "1/TSTOP"). Enter a positive frequency value. Example: v 1 0 4 SFFM (4 1 -1 2 20K)

Chapter 32: Warning/Error Messages

Warning Message Index [10001-10076]

Index	**warning** Message and Example
10033	(10033.sp:4) Carrier frequency cannot be negative for AM voltage source v. Reset to default (freq= 0). Enter a positive frequency value. Example: v 1 0 AM (4 1 1 -2 20K)
10034	(10034.sp:4) Modulation frequency cannot be negative for AM voltage source v. Reset to default (freq= 1/TSTOP). Enter a positive frequency value. Example: v 1 0 AM (4 1 -1 2 20K)
10035	(10035.sp:4) Offset coefficient cannot be negative for AM voltage source v. Reset to default (offset coefficient= 0). Enter a positive offset coefficient value. Example: v 1 0 AM (4 -1 1 2 20K)
10036	(10036.sp:4) Propagation delay cannot be negative for AM voltage source v. Reset to default (Propagation delay= 0). Enter a positive propagation delay value. Example: v 1 0 AM (4 1 1 2 -20K)
10037	(10037.sp:3) Rise or fall time cannot be negative for voltage source v. Please enter positive value of rise or fall time. Reset time to TSTEP. Example: v 1 0 0 PULSE (0 5 1p -1p -1p 49p 50p)
10038	(10038.sp:9) Number of turns (NT) for the element x1.l1 cannot be <= zero. Parameter has been ignored. Enter a valid value of NT. Example: l1 mid 0 1n nt=-1
10039	(10039.sp:14) Value of "TD" cannot be negative in .measure of tdly. Reset to absolute value. Example: .measure tran tdly TRIG v(in) VAL=2.5 td=-1n RISE=1 TARG v(out) VAL=1 CROSS=1
10040	(10040.sp:13) .MEASURE tdly never reached the trigger value, Measurement failed. Example: .measure tran tdly TRIG v(in) VAL=9.5 td=1n RISE=1 TARG v(out) VAL=10 CROSS=1 \$the maximum value of v(out) is 2
10041	(10041.sp:13) .MEASURE tdly never reached the target value, Measurement failed. Example: .measure tran tdly TRIG v(in) VAL=9.5 td=1n RISE=1 TARG v(out) VAL=10 CROSS=1 \$the maximum value of v(out) is 2
10042	(10042.sp:4) Both nodes of element v2 are connected together; Line ignored. Example: v2 1 1 2

Index	**warning** Message and Example
10043	(10043.sp:32) Variable v11 does not exist in the netlist for DC analysis. Please specify the variable which is present in the netlist. v11 assumed to be new variable for DC analysis however results may not be desired. Example: <code>.DC v11 0v 5v 0.1v \$v11 does not exist</code>
10044	(10044.sp:33) FIRSTRUN value -1 cannot be less than 1 for monte-carlo sweep in Transient Analysis. Please specify correct value of FIRSTRUN. Parameter has been ignored. Example: <code>.DC v1 0.5 1 0.5 monte=2 firstrun=-1</code>
10045	(10045.sp:27) The third value 0.00D+00 and the fourth value 1.00D-12 both are smaller than the second value 1.00D-06, so the transient statement is interpreted as: <code>.tran tstep tstop tstart delmax</code> . Example: <code>.tran 1n 1u 0 1p</code>
10046	(10046.sp:12) Duplicate <code>.ic</code> declaration for node 1. Taking the last <code>.ic</code> value 1.00 Example: <code>.ic v(1)=0</code> <code>.ic v(1)=1</code>
10047	(10047.sp:6) Area for diode cannot be 0.0, reset to 1e-12 (default value). Enter valid value for area. Example: <code>d1 1 0 dm1 area=-100p</code>
10048	(10048.sp:6) Geometry parameter "pj" cannot be negative for diode. Please specify positive value to "pj". Reset to zero. Example: <code>d1 1 0 dm1 area=100p pj=-10n</code>
10049	(10049.sp:8) Noise parameter (noise) for the resistance can ONLY be 1 or 0. Please enter a valid value of "noise". Parameter has been ignored. Example: <code>r1 in mid aa noise=2</code>
10050	(10050.sp:5) Argument of <code>asin</code> can range from -1 to 1. Reset value of <code>asin(5.00)</code> to zero. Enter a valid argument for <code>asin</code> . Example: <code>.param aa=asin(5)</code>
10051	(10051.sp:4) Argument of <code>acos</code> can range from -1 to 1. Reset value of <code>acos(5.00)</code> to zero. Enter a valid argument for <code>acos</code> . Example: <code>.param aa=acos(5)</code>

Chapter 32: Warning/Error Messages

Warning Message Index [10001-10076]

Index	**warning** Message and Example
10052	(10052.sp:5) Argument of log cannot be zero. The value of log(0) is replaced by the value of log(epsmin), set by option EPSMIN; Default = 1e-28. Example: <code>.param a='-log(0)'</code>
10053	(10053.sp:82) Node name tn in .IC or .NODESET cannot be found. Nodal initial condition is ignored. Enter a valid node. Example: <code>.IC V(TN)=0 \$node TN does not exist</code>
10054	(10054.sp:87) Invalid node pathname on output variable or initialized node net0103; this statement is ignored. Example: <code>.probe ac vdb(xeq_filter.net0103) \$ pathname xeq_filter does not exist</code>
10055	(10055.sp:137) Multiple ACmatch/DCmatch commands found, only the last one is used. Example: <code>.acmatch v(out) threshold=0 perturb=1</code> <code>.acmatch v(xi82.net18)</code>
10056	(10056.sp:116) Maximum 30 output variables are supported in acmatch/dcmatch analysis. Taking first 30 variables defined and ignoring the rest. Example: <code>.acmatch v(out) v(in_neg) v(gnda) v(vdda) v(in_pos)</code> <code>+ v(out) v(in_neg) v(gnda) v(vdda) v(in_pos) ... more than 30 output variables</code>
10057	(10057.sp:29) DCmatch/ACmatch analysis only supports independent voltage source. i(xmdut0) output ignored. Enter a valid output source. Example: <code>.dcmatch I(xmdut0)</code>
10058	(10058.sp:119) Threshold -10.000 specified in ACmatch/DCmatch analysis is negative. Table has not been generated. Example: <code>.dcmatch v(out) threshold=-10 perturb=1</code>
10059	(10059.sp:119) Perturbation 7.0000 specified in ACmatch/DCmatch analysis exceeds valid range from 0.01 to 6.0. Reset to default (2.0). Enter a valid value. Example: <code>.dcmatch v(out) threshold=0 perturb=7</code>
10060	(10060.sp:17) Interval specified in ACmatch/DCmatch analysis cannot be negative. Parameter ignored. Enter a valid value. Example: <code>.acmatch v(12) perturb=3.0 interval=-3 threshold=0.9</code> <code>matched=0.95</code>

Index	**warning**	Message and Example
10061	(10061.sp:119)	Unrecognized Output variable specified in DCmatch/DCsens. Analysis ignored. Enter a valid output variable. Example: <code>.dcmatch gv(out) threshold=10 perturb=1</code>
10062	(10062.sp:49)	Threshold -6.00000E-02 cannot be negative for DCsens analysis. value Reset to zero. Enter a valid vale for threshold. Example: <code>.DCsens v(2) file='2357' pertur=1.0 threshold=-60m interval = 2 _dsdbg=1</code>
10063	(10063.sp:49)	Perturbation 7.0000 specified in DCsens analysis exceeds valid range from 0.0001 to 1.0. Reset it to default (0.05). Enter a valid value. Example: <code>.DCsens v(2) file='2357' pertur=7.0 threshold=60m interval = 2 _dsdbg=1</code>
10064	(10064.sp:49)	Interval specified in DCsens analysis cannot be negative. Parameter reset to default(1). Enter a valid value. Example: <code>.DCsens v(2) file='2357' pertur=1.0 threshold=60m interval = -2 _dsdbg=1</code>
10065	(10065.sp:49)	Groupbydevice (groupbydev) for DCsens analysis can be 0 or 1. Reset to default (0). Enter a valid value. Example: <code>.DCsens v(2) file='2357' pertur=1.0 threshold=60m interval = 2 _dsdbg=1 groupbydev=2</code>
10066	(10066.sp:6)	LEVEL cannot be used along with keyword LAPLACE for element lowpass. Parameter has been ignored. Example: <code>Glowpass 0 out LAPLACE in 0 1.0 / 1.0 2.0 2.0 1.0 level=0</code>
10067	(10067.sp:116)	Measurement time value specified exceeds analysis limit in measure variable vcp. .measure has been ignored. Enter a valid value in .measure. Example: <code>.tran `1n` `10n` start=`0` .measure TRAN vcp FIND v(vcp) AT 780u</code>
10068	(10068.sp:2)	noise is not level= 49 model parameter. Parameter has been ignored. Example: <code>.model nch nmos version=3.22 level=49 noise=1</code>

Chapter 32: Warning/Error Messages

Warning Message Index [10001-10076]

Index	**warning** Message and Example
10069	(10069.sp:96) Multiple analysis statements are not allowed. The latter statement ignored. Example: <pre>.op all .op</pre>
10070	(10070.sp:16) Capacitance between second node and BULK node cannot be negative for resistor instance r1. Please enter a valid value of "C". Parameter has been ignored. Example: <pre>r1 out 0 5 c=-1p</pre>
10071	(10071.sp:21) Capacitance of 1.00 for c0 is too high. Please verify capacitance value. Example: <pre>c0 in_neg 0 1</pre>
10072	(10072.sp:5) Value of resistance r1 defined is limited to 1.000E-05 (RESMIN). Please modify RESMIN to incorporate resistances. Example: <pre>r1 1 3 1e-10</pre>
10073	(10073.sp:336) Measure results may be incorrect since initial start time is non-zero. Example: <pre>.TRAN 0.001NS 9NS START=2NS .measure tran inv_delay trig v(01) val=1.25 rise=1 targ v(02) val=1.25 + fall=1</pre>
10074	(10074.sp:24) Parameter weff is defined as an expression containing output signals, which may cause incorrect result. Recommend replacing output signals with user-defined functions. Example: <pre>.param Weff='(w- (dW+pdWb*MAX((V(B,D)),(V(B,S)))+pdWd*ABS(V(D,S))))'</pre>
10075	(10075.sp:0) Parameter(s), within encrypted block, contain output signals in expression which may cause incorrect result. Recommended: Replace output signals with user-defined functions. Example: <pre>.prot .param Weff='(w- (dW+pdWb*MAX((V(B,D)),(V(B,S)))+pdWd*ABS(V(D,S))))' .unprot</pre>

Index	**warning**	Message and Example
10076	(10076.sp:32)	Number of monte-carlo sweep cannot be negative number for DC analysis. Please enter positive value of sweeps. Monte= -1.00 has been ignored. Example: <code>.DC v1 0.5 1 0.5 monte=-1</code>

Error Message Index [20001-20024]

The following table lists an index of error messages that can be encountered in HSPICE usage. The table is organized according to the index number, netlist line, sample warning message, and example of what triggered the warning.

Index	**error**	Message and Example
20001	(20001.sp:19)	Unbalanced parentheses. Example: <code>.param aa = '5*(1+bb'</code>
20002	(20002.sp:38)	Successive time point 5.000E-12, 4.900E-12 must increase for piecewise linear voltage source v8. Example: <code>v8 8 0 0 PWL (0 0 1p 1 5p 1 4.9p 0 10n 0)</code>
20003	(20003.sp:30)	Cannot find table of data v11 for DC analysis. Please specify data table which has valid definition. Example: <code>.DC DATA=v11 \$ v11 does not exist</code>
20004	(20004.sp:25)	TSTOP cannot be zero or negative for Transient Analysis. Please enter a positive value for TSTOP. Example: <code>.tran 0.1ns -50ns</code>
20005	(20005.sp:25)	Sweep step cannot be 0 for Transient Analysis. Please enter positive value of sweep. Example: <code>.tran 0.1ns 50ns sweep v1 0 5 0</code>
20006	(20006.sp:5)	Number of Stages (TAP) for Pseudo Random Bit Generator source v1 should lie between 2 and 30. Please enter a valid value of TAP. Example: <code>V1 1 0 LFSR (0 1 1u 1n 1n 10meg 1 [-5, 2] rout=10)</code>

Index	**error** Message and Example
20007	(20007.sp:150) For AC analysis STOP frequency cannot be less than START frequency. STOP frequency should be greater than START frequency. Example: <code>.AC lin 1000 10GHz 1GHz</code>
20008	(20008.sp:195) Number of points for AC analysis cannot be less than 1. Please enter number of points more than 1. Example: <code>.AC lin 0 1Hz 1GHz</code>
20009	(20009.sp:4) Definition not declared for rsh(1+b). Please enter a defined name. Example: <code>.param rsh=100 b=10</code> <code>Rbody 1 0 'rsh(1+b)'</code>
20010	(20010.sp:22) Mutual Inductor declaration contains only one reference inductor. Please enter at least 2 reference inductors. Example: <code>K1 L1</code>
20011	(20011.sp:509) Definition of model/subckt "pch_hvt_mac" is not found for the element "xcut.xmmu7012345678901234567890". Please specify a defined model/subckt name. Example: <code>xMMU7012345678901234567890 net132 MU70_GATE MU70_SRC MI33-</code> <code>M_u3_BULK pch_hvt_mac ad=0.0119p as=0.017044p dfm_flag=1 \$</code> <code>pch_hvt_mac does not exist</code>
20012	(20012.sp:15) Number of data points used in Delay (NPDELAY) cannot be negative for element fd. Enter positive NPDELAY value. Example: <code>Fd in 0 DELAY vcc TD=7ns SCALE=5 NPDELAY=-10</code>
20013	(20013.sp:17) For W element the number of terminals exceeds 6. For the given number of signal conductors, N = 2, the number of terminals should be 2(N + 1) = 6. Enter correct number of terminals. Example: <code>WTL1_0_0 1 2 7 0 3 4 0 TABLEMODEL=MODEL_5 N=2 L=LEN delayopt=3</code>

Index	**error** Message and Example
20014	<p>(20014.sp:17) The W-element has invalid number of terminals 5 for the given number of signal conductors N = 2. The number of terminals should be 2(N + 1). Enter valid number of terminals.</p> <p>Example:</p> <pre>WTL1_0_0 1 0 3 4 0 TABLEMODEL=MODEL_5 N=2 L=LEN delayopt=3</pre>
20015	<p>(20015.sp:12) Physical length defined for U-element u1 cannot be zero or negative. Please define a positive value to L.</p> <p>Example:</p> <pre>U1 3 10 2 0 5 1 4 0 USTRIP L=-0.178</pre>
20016	<p>(20016.sp:17) Number of conductors (N) parameter not found, for W element. Please define value of N.</p> <p>Example:</p> <pre>WTL1_0_0 1 2 0 3 4 0 TABLEMODEL=MODEL_5 L=2 delayopt=3 \$ N=2</pre>
20017	<p>(20017.sp:12) Number of nodes "7" of U-element "u1" does not match with number of nodes " 8" specified for U model "ustrip". Number of nodes of instance should match with the model.</p> <p>Example:</p> <pre>U1 3 10 2 0 5 1 0 USTRIP L=0.178 \$LUMPS=1 .Model USTRIP U LEVEL=3 PLev=1 Elev=1 Dlev=2 Nl=3 Ht=381u + Wd=305u Th=25u Sp=102u Ts=838u Kd=4.7 \$ LEVEL=3 required 8 nodes</pre>
20018	<p>(20018.sp:32) Unknown setting for ComputeGo. ComputeGo can be YES or NO.</p> <p>Example:</p> <pre>.Fsoptions opt1 Printdata=yes Computers=yes Computegd=yes + Computego=2</pre>
20019	<p>(20019.sp:38) Physical length defined for W-element cannot be zero or negative. Please define a positive value.</p> <p>Example:</p> <pre>W4 N=3 1 2 3 0 4 5 6 0 RLGCfile=wel4rs.rlc l=-1.2</pre>
20020	<p>(20020.sp:25) TSTEP cannot be zero or negative for Transient Analysis. Please enter a positive value for TSTEP.</p> <p>Example:</p> <pre>.tran -0.1ns 50ns</pre>

Index	**error** Message and Example
20021	(20021.sp:5) Denominator cannot be zero for E-element ehipass. Please enter a non-zero denominator. Example: <pre>Ehipass out 0 LAPLACE in 0 0.0,0.0,0.0,1.0 / 0.0,0.0,0.0,0.0</pre>
20022	(20022.sp:4) Seed for Pseudo Random Bit Generator source v1 should lie between 0 and 1073741823. Please enter a valid value of seed. Example: <pre>.param a=4 vlow=1 vhigh=5 tdelay=1n trise=0.5n tfall=0.5n rate=0.1g seed=-10 rout=10k v1 1 0 LFSR (vlow vhigh tdelay trise tfall rate seed [5,2] rout)</pre>
20023	(20023.sp:13) Number of nodes miss match between instance "xres" and subcircuit "ress ". Subcircuit definition has 2 node(s) whereas subckt instance was found with 3 node(s). Please specify same number of nodes. Example: <pre>.subckt ress 7 1 R1 7 1 50 .ends Xres 7 1 10 ress</pre>
20024	(20024.sp:3) Name of rlgcmodel cannot start with a number. Names should start with alphabet. Check/correct the parameter name. Example: <pre>W1 N=3 1 3 5 0 2 4 6 0 RLGCMODEL=11 l=0.97</pre>

Exit Codes

HSPICE prints these exit codes. The numerals below may be preceded by the word *SIGTERM* or *SIGABRT*. The corresponding meanings are as follows:

- 0: Simulation succeeded.
- 1: Simulation failed due to errors, e.g., syntax error, non-convergence, etc.
- 2: HSPICE stopped due to lack of an HSPICE license.
- 3: The simulation is terminated by **Ctrl+** (Control key plus backslash) on Linux.

- 6: The simulation is terminated by signal SIGABRT due to abnormal operation of the source code, such as: out of memory, invalid memory access, etc.
- 8: Floating-point exception.
- 11: Segmentation fault (invalid memory address access).
- 15: HSPICE stopped by a UNIX `kill` command.
- 24: CPU limit exceeded.
- 101: HSPICE stopped by **Ctrl+C**.

Chapter 32: Warning/Error Messages
Exit Codes

Full Simulation Example

Contains information and sample input netlist for a full simulation example in HSPICE.

The example in this appendix shows the basic text and post-processor output for two sample input netlists. The example uses WaveView to view the results.

Simulation Example Using WaveView

This example demonstrates the basic steps to perform a simulation and to view the waveform results by using the Synopsys WaveView waveform viewer.

Input Netlist and Circuit

This example is based on demonstration netlist example.sp, which is available in directory `$installdir/demo/hspice/bench`. This example is an input netlist for a linear CMOS amplifier. Comment lines indicate the individual sections of the netlist. See the [HSPICE Reference Manual: Commands and Control Options](#) for information about individual commands.

To see the path for this example (CMOS.sp) and others go to [Benchmark Examples](#) in this user guide.

Appendix A: Full Simulation Example

Simulation Example Using WaveView

Input netlist

```
* Example HSPICE netlist, using a linear CMOS amplifier
* netlist options
.option post probe brief nomod
* defined parameters
.param analog_voltage=1.0
* global definitions
.global vdd
* source statements
Vinput in gnd SIN ( 0.0v analog_voltage 10x )
Vsupply vdd gnd DC=5.0v
* circuit statements
Rinterm in gnd 51
Cincap in infilt 0.001
Rdamp infilt clamp 100
Dlow gnd clamp diode_mod
Dhigh clamp vdd diode_mod
Xinv1 clamp inv1out inverter
Rpull clamp inv1out 1x
Xinv2 inv1out inv2out inverter
Routterm inv2out gnd 100x
* subcircuit definitions
.subckt inverter in out
Mpmos out in vdd vdd pmos_mod l=1u w=6u
Mnmos out in gnd gnd nmos_mod l=1u w=2u
.ends
* model definitions
.model pmos_mod pmos level=3
.model nmos_mod nmos level=3
.model diode_mod d
* analysis specifications
.TRAN 10n 1u sweep analog_voltage lin 5 1.0 5.0
* output specifications
.probe TRAN v(in) v(clamp) v(inv1out) v(inv2out) i(dlow)
.measure TRAN falltime TRIG v(inv2out) VAL=4.5v FALL=1
+ TARG V(inv2out) VAL=0.5v FALL=1
.end
```

[Figure 198 on page 1047](#) is a circuit diagram for the linear CMOS amplifier in the circuit portion of the netlist. The two sources in the diagram are also in the netlist.

Note: The inverter symbols in the circuit diagram are constructed from two complementary MOSFET elements. The diode and MOSFET models in the netlist do not have non-default parameter values, except to specify Level 3 MOSFET models (empirical model).

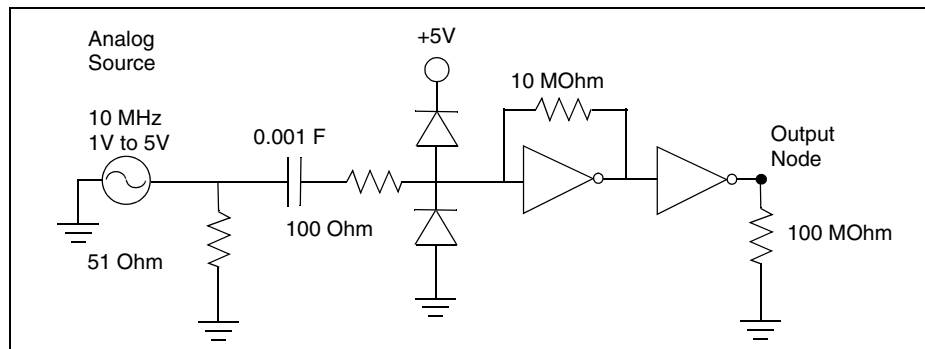


Figure 198 Circuit Diagram for Linear CMOS Inverter

Execution and Output Files

The following section displays the output files from a HSPICE simulation of the amplifier shown in the previous section. To execute the simulation, enter:

```
% hspice example.sp > example.lis
```

In this syntax, the input netlist name is *example.sp*, and the output listing file name is *example.lis*. Simulation creates the following output files:

Table 108 HSPICE Output Files

Filename	Description
example.ic	Initial conditions for the circuit.
example.lis	Text simulation output listing.
example.mt0	Post-processor output for .MEASURE statements.
example.pa0	Subcircuit path table.
example.st0	Run-time statistics.

Appendix A: Full Simulation Example

Simulation Example Using WaveView

Table 108 HSPICE Output Files (Continued)

Filename	Description
example.tr0	Post-processor output for transient analysis.

The following subsections show text files to simulate the amplifier by using HSPICE on a UNIX workstation. The example does not show the two post-processor output files, which are in binary format.

Example.ic0

Initial conditions for the circuit

```
* "simulator" "HSPICE"
* "version" "A-2008.03 32-BIT"
* "format" "HSP"
* "rundate" "15:39:17 04/07/2008"
* "netlist" "example.sp "
* "runtitle" "example hspice netlist using a linear cmos
* amplifier"
* time= 0.
* temperature= 25.0000
*** BEGIN: Saved Operating Point ***
.option
+ gmindc= 1.0000p
.nodeset
+ clamp= 2.6200
+ in= 0.
+ infilt= 2.6200
+ invlout= 2.6200
+ inv2out= 2.6199
+ vdd= 5.0000
*** END: Saved Operating Point ***
```

Example.mt0

Post-processor output for .MEASURE statements

```
$DATA1 SOURCE='HSPICE' VERSION='A-2008.03 32-BIT'
.TITLE '* example hspice netlist, using a linear cmos amplifier'
analog_voltage    falltime          temper          alter#
1.0000            3.788e-08          25.0000         1.0000
2.0000            1.639e-08          25.0000         1.0000
3.0000            1.085e-08          25.0000         1.0000
4.0000            7.843e-09          25.0000         1.0000
5.0000            6.572e-09          25.0000         1.0000
```

Example.lis

Simulation output listing text file

```
Using: /usr/bin/time -p /remote/cktcae/HSPICE/A-2008.03/hspice/  
linux/hspice example.sp
```

```
***** HSPICE -- A-2008.03 32-BIT (Feb 26 2008) linux *****  
Copyright (C) 2008 by Synopsys Inc. All rights reserved.  
Unpublished-rights reserved under US copyright laws.  
This program is protected by law and is subject to the terms and  
conditions of the license agreement found in:
```

```
  /remote/cktcae/HSPICE/A-2008.03/hspice/license.warn
```

```
Use of this program is your acceptance to be bound by this  
license agreement. HSPICE is a trademark of Synopsys, Inc.
```

```
Input File: example.sp
```

```
lic:
```

```
lic:FLEXlm: v8.5b
```

```
lic: USER:   hspiceuser           HOSTNAME: hspiceamd1
```

```
lic: HOSTID: 0015605ff6da        PID:      18077
```

```
lic: Using FLEXlm license file:
```

```
lic: 26585@us01_lic4
```

```
lic: Checkout 1 hspice
```

```
lic: License/Maintenance for hspice will expire on 31-jan-2009/  
2009.12
```

```
lic: FLOATING license(s) on SERVER us01_lic4
```

```
lic:
```

```
Init: hspice initialization file: hspice.ini
```

```
  * netlist options
```

```
  .option post probe brief nomod
```

```
*****
```

```
** runlvl is invoked, you can disable it by:
```

```
  a) Add option runlvl=0 to your current simulation job.
```

```
  b) Copy $installdir/hspice.ini to your HOME directory and  
     customize it by adding option runlvl=0, which disables  
     it for all of your simulation jobs.
```

```
  c) Re-invoke $installdir/bin/config program and unselect the  
     option runlvl setting in box 'hspice.ini' which disables  
     it for whole group simulation jobs.
```

```
** runlvl is invoked, some options are ignored or automatically  
set:
```

```
Options below are automatically set (user setting will  
overwrite them):
```

```
if runlvl=6,           .option bypass=0
```

```
if runlvl=[1|2|3|4|5], .option bypass=2
```

```
Options below are ignored, they are replaced by automated
```

Appendix A: Full Simulation Example

Simulation Example Using WaveView

```
      algorithms:
        lvltim   dvdt       ft    fast   trtol  absvar  relvar
        relq    chgtol     dvtr   imin   itl3    rmax
** runlvl is invoked, actual option value used by HSPICE are:
    runlvl= 3      bypass= 2      mbypass= 2.00  bytol= 100.00u
*****

**warning** dc voltage reset to initial transient source value
           in source           0:vinput    new dc= 0.0000D+00

    Opening plot unit= 15
    file=example.pa0

*** parameter analog_voltage = 1.000E+00 ***

*****
* example hspice netlist, using a linear cmos amplifier
***** transient analysis tnom= 25.000 temp= 25.000 *****
falltime= 3.7885E-08 targ= 7.0408E-08 trig= 3.2524E-08

        ***** job concluded

*** parameter analog_voltage = 2.000E+00 ***

*****
* example hspice netlist, using a linear cmos amplifier

***** transient analysis tnom= 25.000 temp= 25.000 *****
falltime= 1.6394E-08 targ= 5.8128E-08 trig= 4.1734E-08

        ***** job concluded

*** parameter analog_voltage = 3.000E+00 ***
*****

* example hspice netlist, using a linear cmos amplifier

***** transient analysis          tnom= 25.000 temp= 25.000

*****
falltime= 1.0848E-08 targ= 5.5187E-08 trig= 4.4339E-08

        ***** job concluded

*** parameter analog_voltage = 4.000E+00 ***
*****
```


Appendix A: Full Simulation Example
Simulation Example Using WaveView

```
* example hspice netlist, using a linear cmos amplifier

***** transient analysis tnom= 25.000 temp= 25.000 *****
falltime= 7.8434E-09 targ= 5.4334E-08 trig= 4.6490E-08

***** job concluded

*** parameter analog_voltage = 5.000E+00 ***

*****
* example hspice netlist, using a linear cmos amplifier

***** transient analysis tnom= 25.000 temp= 25.000 *****
falltime= 6.5718E-09 targ= 5.3069E-08 trig= 4.6497E-08

meas_variable = falltime
mean = 15.9083n          varian = 165.2538a
sigma = 12.8551n        avgdev = 8.9848n
max = 37.8846n          min = 6.5718n
1-sigma = 12.8551n      median = 16.3940n

***** job concluded

***** HSPICE -- A-2008.03 32-BIT (Feb 26 2008) linux *****
*****
* example hspice netlist, using a linear cmos amplifier
***** job statistics summary          tnom= 25.000 temp= 25.000
*****

          total memory used          159 kbytes
# nodes =          8 # elements=          14
# diodes=          2 # bjts   =          0 # jfets =          0 # mosfets =          4
# va device =          0

analysis time  # points  tot. iter  conv.iter
op point 0.00    1          55
transient 0.06   505        6881      2043 rev=          58
readin          0.00
errchk          0.00
setup           0.00
output          0.00

          total cpu time          0.06 seconds
          total elapsed time          1 seconds
          job started at 15:39:17 04/07/2008
          job ended   at 15:39:18 04/07/2008

Init: hspice initialization file: hspice.ini
lic: Release hspice token(s)
```

Appendix A: Full Simulation Example

Simulation Example Using WaveView

Example.pa0

Subcircuit path output

```
1 xinv1.  
2 xinv2.
```

Example.st0

Run-time statistics

```
***** HSPICE -- A-2008.03 32-BIT (Feb 26 2008) linux *****  
Input File: example.sp  
lic:  
lic: FLEXlm: v8.5b  
lic: USER: hspiceuser          HOSTNAME: hspiceamd1  
lic: HOSTID: 0015605ff6da      PID: 18077  
lic: Using FLEXlm license file:  
lic: 26585@us01_lic4  
lic: Checkout 1 hspice  
lic: License/Maintenance for hspice will expire on 31-jan-2009/  
2009.12  
lic: FLOATING license(s) on SERVER us01_lic4  
lic:  
Init: hspice initialization file: hspice.ini  
init: begin read circuit files,  cpu clock= 0.00E+00  
      option post  
      option probe  
      option brief  
init: end read circuit files,  cpu clock= 0.00E+00 memory= 145 kb  
init: begin check errors,  cpu clock= 0.00E+00  
init: end check errors,  cpu clock= 0.00E+00 memory= 145 kb  
init: begin setup matrix, pivot= 10 cpu clock= 0.00E+00  
establish matrix -- done,  cpu clock= 0.00E+00 memory= 146 kb  
re-order matrix -- done,  cpu clock= 0.00E+00 memory= 147 kb  
init: end setup matrix,  cpu clock= 0.00E+00 memory= 158 kb  
sweep: parameter          parameter1          begin, #sweeps = 5  
parameter: analog_voltage = 1.00E+00  
dcop: begin dcop,  cpu clock= 0.00E+00  
dcop: end dcop,  cpu clock= 0.00E+00 memory= 158 kb tot_iter=  
11  
output: example.mt0  
sweep: tran tran1  begin, stop_t= 1.00E-06 #sweeps= 101 cpu  
clock= 0.00E+00  
tran: time= 1.0000E-07 tot_iter= 97 conv_iter= 31 cpu  
clock= 1.00E-02  
tran: time= 2.0000E-07 tot_iter= 195 conv_iter= 62 cpu  
clock= 1.00E-02  
tran: time= 3.0000E-07 tot_iter= 296 conv_iter= 93 cpu  
clock= 1.00E-02
```

Appendix A: Full Simulation Example
Simulation Example Using WaveView

```

tran: time= 4.0000E-07 tot_iter=    392 conv_iter=    124 cpu
clock= 1.00E-02
tran: time= 5.0000E-07 tot_iter=    484 conv_iter=    154 cpu
clock= 1.00E-02
tran: time= 6.0000E-07 tot_iter=    580 conv_iter=    184 cpu
clock= 1.00E-02
tran: time= 7.0000E-07 tot_iter=    677 conv_iter=    215 cpu
clock= 1.00E-02
tran: time= 8.0000E-07 tot_iter=    778 conv_iter=    246 cpu
clock= 1.00E-02
tran: time= 9.0000E-07 tot_iter=    874 conv_iter=    277 cpu
clock= 1.00E-02
tran: time= 1.0000E-06 tot_iter=    966 conv_iter=    307 cpu
clock= 1.00E-02
sweep: tran tran1  end, cpu clock= 1.00E-02 memory=    159 kb
parameter: analog_voltage =    2.00E+00
dcop: begin dcop,  cpu clock= 1.00E-02
dcop: end dcop,  cpu clock= 1.00E-02 memory=    158 kb tot_iter=
22
output: example.mt0
sweep: tran tran2  begin, stop_t= 1.00E-06 #sweeps= 101 cpu
clock= 1.00E-02
tran: time= 1.0000E-07 tot_iter=    101 conv_iter=     34 cpu
clock= 1.00E-02
tran: time= 2.0000E-07 tot_iter=    200 conv_iter=     68 cpu
clock= 2.00E-02
tran: time= 3.0000E-07 tot_iter=    305 conv_iter=    101 cpu
clock= 2.00E-02
tran: time= 4.0000E-07 tot_iter=    415 conv_iter=    136 cpu
clock= 2.00E-02
tran: time= 5.0000E-07 tot_iter=    535 conv_iter=    174 cpu
clock= 2.00E-02
tran: time= 6.0000E-07 tot_iter=    652 conv_iter=    210 cpu
clock= 2.00E-02
tran: time= 7.0000E-07 tot_iter=    769 conv_iter=    246 cpu
clock= 2.00E-02
tran: time= 8.0000E-07 tot_iter=    886 conv_iter=    282 cpu
clock= 2.00E-02
tran: time= 9.0000E-07 tot_iter=   1003 conv_iter=    318 cpu
clock= 2.00E-02
tran: time= 1.0000E-06 tot_iter=   1120 conv_iter=    354 cpu
clock= 2.00E-02
sweep: tran tran2  end, cpu clock= 2.00E-02 memory=    159 kb
parameter: analog_voltage =    3.00E+00
dcop: begin dcop,  cpu clock= 2.00E-02
dcop: end dcop,  cpu clock= 2.00E-02 memory=    158 kb tot_iter=
33
output: example.mt0
sweep: tran tran3  begin, stop_t= 1.00E-06 #sweeps= 101 cpu
clock= 2.00E-02

```

Appendix A: Full Simulation Example
 Simulation Example Using WaveView

```

tran: time= 1.0000E-07 tot_iter=      130 conv_iter=      40 cpu
clock= 3.00E-02
tran: time= 2.0000E-07 tot_iter=      253 conv_iter=      76 cpu
clock= 3.00E-02
tran: time= 3.0000E-07 tot_iter=      376 conv_iter=     112 cpu
clock= 3.00E-02
tran: time= 4.0000E-07 tot_iter=      499 conv_iter=     148 cpu
clock= 3.00E-02
tran: time= 5.0000E-07 tot_iter=      622 conv_iter=     184 cpu
clock= 3.00E-02
tran: time= 6.0000E-07 tot_iter=      745 conv_iter=     220 cpu
clock= 3.00E-02
tran: time= 7.0000E-07 tot_iter=      868 conv_iter=     256 cpu
clock= 3.00E-02
tran: time= 8.0000E-07 tot_iter=      991 conv_iter=     292 cpu
clock= 3.00E-02
tran: time= 9.0000E-07 tot_iter=     1114 conv_iter=     328 cpu
clock= 3.00E-02
tran: time= 1.0000E-06 tot_iter=     1237 conv_iter=     364 cpu
clock= 3.00E-02
sweep: tran tran3   end,  cpu clock= 3.00E-02 memory=   159 kb
parameter: analog_voltage = 4.00E+00
dcop: begin dcop,  cpu clock= 3.00E-02
dcop: end dcop,  cpu clock= 3.00E-02 memory=   158 kb tot_iter=
44
output: example.mt0
sweep: tran tran4   begin, stop_t= 1.00E-06 #sweeps= 101 cpu
clock= 3.00E-02
tran: time= 1.0000E-07 tot_iter=      159 conv_iter=      45 cpu
clock= 3.00E-02
tran: time= 2.0000E-07 tot_iter=      322 conv_iter=      92 cpu
clock= 4.00E-02
tran: time= 3.0000E-07 tot_iter=      485 conv_iter=     139 cpu
clock= 4.00E-02
tran: time= 4.0000E-07 tot_iter=      648 conv_iter=     186 cpu
clock= 4.00E-02
tran: time= 5.0000E-07 tot_iter=      811 conv_iter=     233 cpu
clock= 4.00E-02
tran: time= 6.0000E-07 tot_iter=      974 conv_iter=     280 cpu
clock= 4.00E-02
tran: time= 7.0000E-07 tot_iter=     1137 conv_iter=     327 cpu
clock= 4.00E-02
tran: time= 8.0000E-07 tot_iter=     1300 conv_iter=     374 cpu
clock= 4.00E-02
tran: time= 9.0000E-07 tot_iter=     1463 conv_iter=     421 cpu
clock= 5.00E-02
tran: time= 1.0000E-06 tot_iter=     1626 conv_iter=     468 cpu
clock= 5.00E-02
sweep: tran tran4   end,  cpu clock= 5.00E-02 memory=   159 kb
parameter: analog_voltage = 5.00E+00

```

```
dcop: begin dcop,  cpu clock= 5.00E-02
dcop: end dcop,  cpu clock= 5.00E-02 memory= 158 kb tot_iter=
55
output: example.mt0
sweep: tran tran5  begin, stop_t= 1.00E-06 #sweeps= 101 cpu
clock= 5.00E-02
tran: time= 1.0000E-07 tot_iter= 173 conv_iter= 48 cpu
clock= 5.00E-02
tran: time= 2.0000E-07 tot_iter= 378 conv_iter= 104 cpu
clock= 5.00E-02
tran: time= 3.0000E-07 tot_iter= 573 conv_iter= 163 cpu
clock= 5.00E-02
tran: time= 4.0000E-07 tot_iter= 786 conv_iter= 222 cpu
clock= 5.00E-02
tran: time= 5.0000E-07 tot_iter= 991 conv_iter= 278 cpu
clock= 5.00E-02
tran: time= 6.0000E-07 tot_iter= 1184 conv_iter= 334 cpu
clock= 6.00E-02
tran: time= 7.0000E-07 tot_iter= 1379 conv_iter= 388 cpu
clock= 6.00E-02
tran: time= 8.0000E-07 tot_iter= 1569 conv_iter= 443 cpu
clock= 6.00E-02
tran: time= 9.0000E-07 tot_iter= 1741 conv_iter= 494 cpu
clock= 6.00E-02
tran: time= 1.0000E-06 tot_iter= 1912 conv_iter= 545 cpu
clock= 6.00E-02
sweep: tran tran5  end,  cpu clock= 6.00E-02 memory= 159 kb
sweep: parameter  parameter 1 end
>info: ***** hspice job concluded
Init: hspice initialization file: hspice.ini
lic: Release hspice token(s)
```

View HSPICE Results in WaveView

These steps show how to use the Synopsys WaveView Waveform Viewer to view the results of the transient analysis from the linear CMOS amplifier simulation.

To view HSPICE transient analysis waveforms, do the following:

1. Invoke SPICE Explorer—From a UNIX command line, type:

```
% sx
```

On a Windows-NT system, choose the menu command:

Programs > (*user_install_location*) > SPICE Explorer

2. In the menu bar, select File > Import Waveform file... (Ctrl-O)

Appendix A: Full Simulation Example

Simulation Example Using WaveView

3. In the Open: Waveform Files dialog box, click on *example.tr0* and click **OK**. This opens the transient analysis waveform file.
4. Expand the hierarchy in the output view browser to show the available output signals.
5. Left-click on the signal $v(in)$, then drag and drop the selected signal into the WaveView panel to display a plot similar to the one shown in [Figure 199](#) on [page 1056](#).

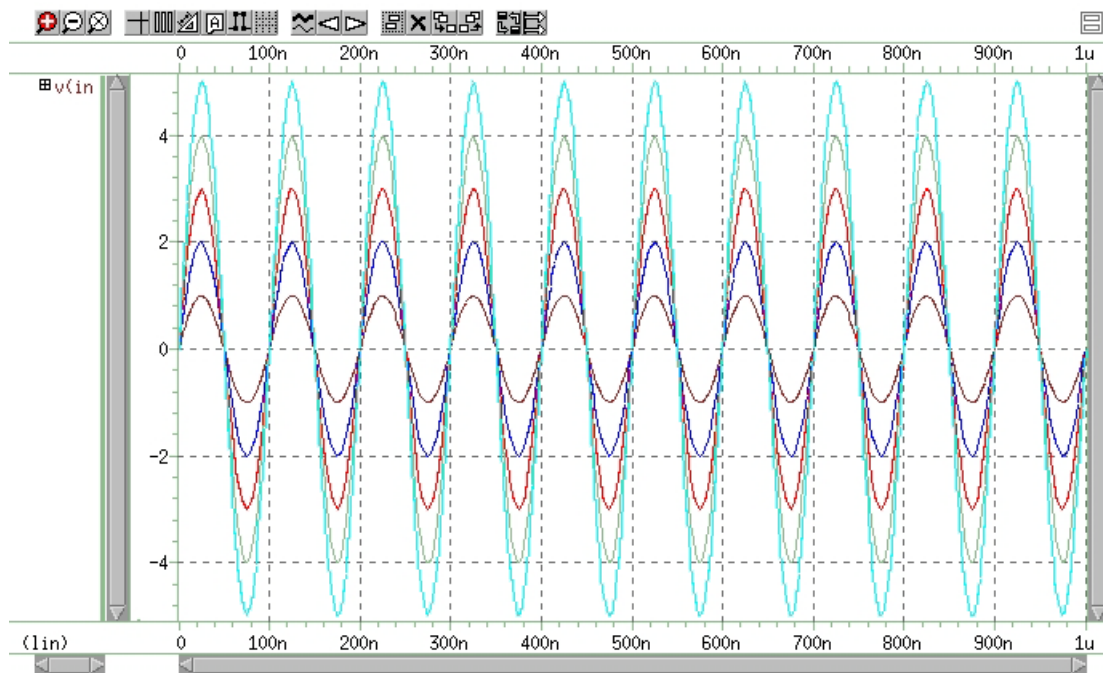


Figure 199 Plot of Voltage on Node in

6. To delete this plot, click on the **X** icon in the WaveView panel tool bar.
7. Repeat [Step 5](#) to plot $v(clamp)$. You should see a plot similar to the one shown in [Figure 240](#).

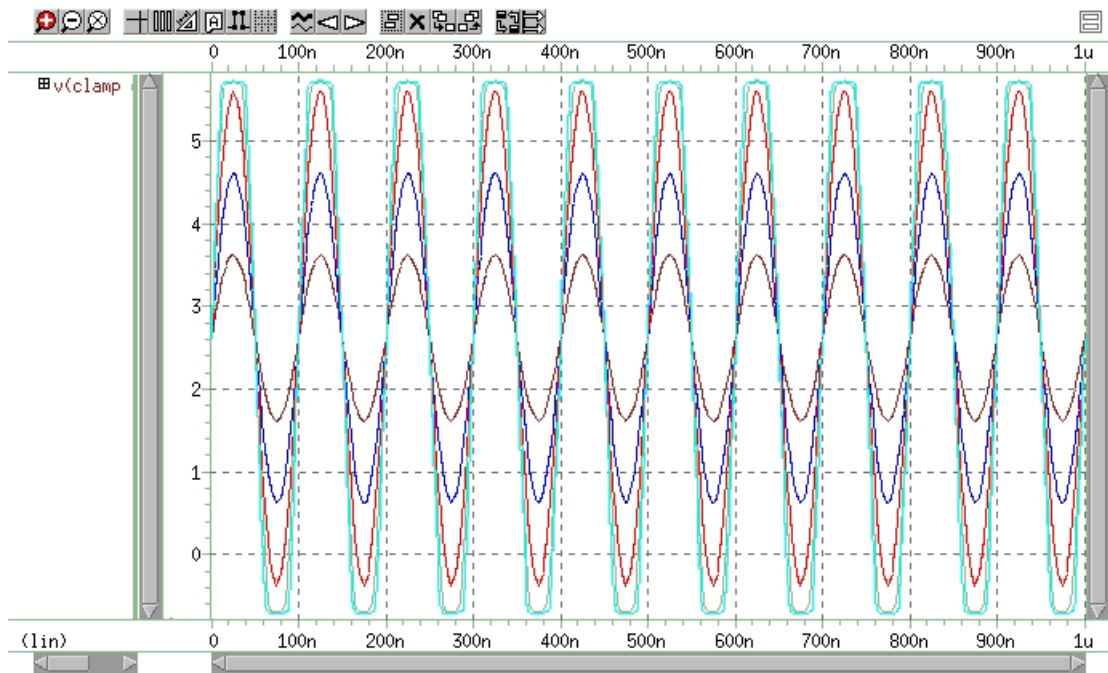


Figure 200 Plot of Voltage on Node clamp vs. Time

- Repeat [Step 6](#) and [Step 7](#) to view the signals $v(inv1out)$ and $v(inv2out)$ ([Figure 201](#) and [Figure 202](#) on page 1059).

Appendix A: Full Simulation Example
Simulation Example Using WaveView

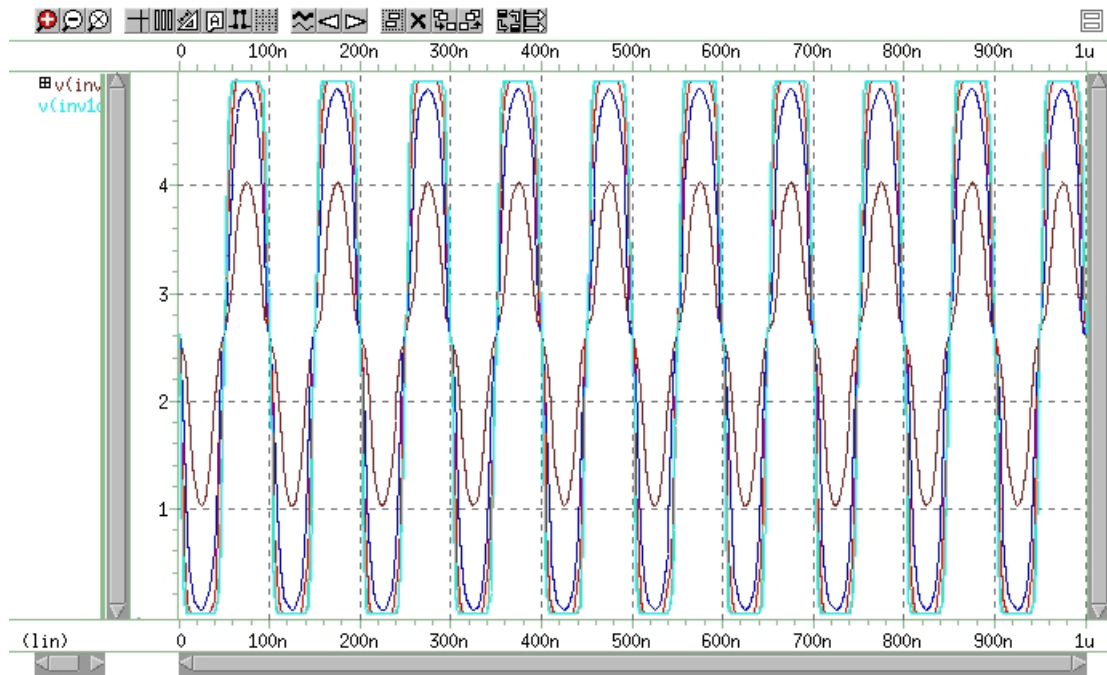


Figure 201 Plot of Voltage on Node inv1out vs. Time

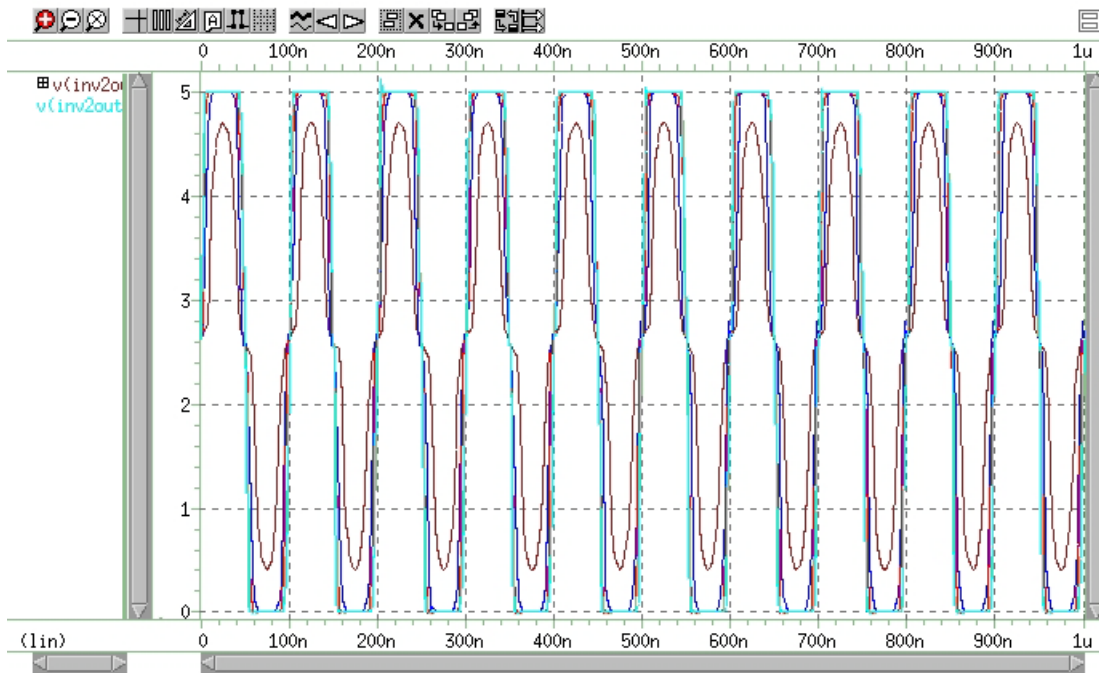



Figure 202 Plot of Voltage on Node *inv2out* vs. Time

9. To delete this plot, click the **X** icon in the WaveView panel tool bar.
10. Left-click on the signal *i(dlow*, then drag and drop the selected signal into the WaveView panel.
11. To show the individual members from the voltage sweep, expand the signal hierarchy by clicking on the '+' sign near the signal name in the WaveView panel.
12. Click the ungroup panels icon () on the WaveView panel tool bar to display a plot similar to the one shown in [Figure 203 on page 1060](#).

Appendix A: Full Simulation Example

Simulation Example Using WaveView

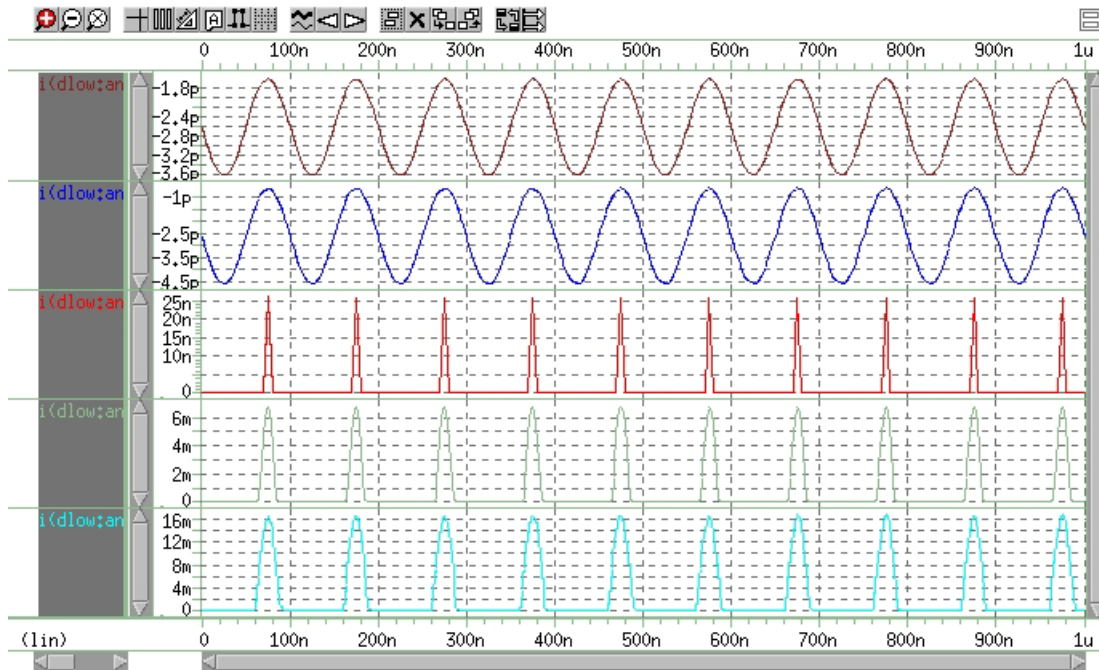


Figure 203 Plot of Current through Diode *dlow* vs. Time

13. To delete this plot, click the **X** icon in the WaveView panel tool bar.
14. In the menu bar, select File > Import Waveform file (Ctrl-O).
15. In the Open: Waveform Files dialog box, click on `example.mt0` and click **OK** to open the measure file.
16. Expand the hierarchy in the output view browser to show the available output signals.
17. Left-click on the signal *falltime*. Drag and drop the selected signal into the WaveView panel. You should see a plot similar to the one shown in [Figure 204 on page 1061](#).

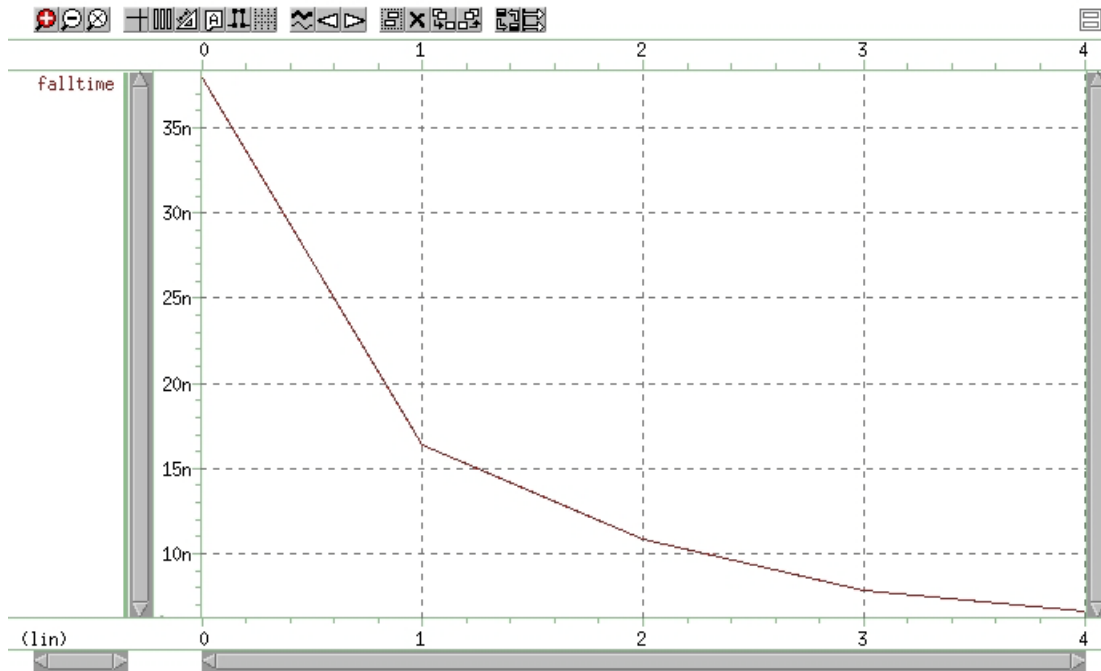


Figure 204 Plot of Measured Variable falltime vs. Amplifier Input Voltage

This concludes the simulation example using WaveView.

The *SPICE Explorer and WaveView User's Manual* includes a full tutorial, information about the various SPICE Explorer tools, and reference information.

You can also find more information on the Synopsys website:

[http:// www.synopsys.com](http://www.synopsys.com).

Appendix A: Full Simulation Example
Simulation Example Using WaveView

Obsolete HSPICE Functionality

Describes out-of-date, rarely used, or de-emphasized functionality.

The following sections are included in the HSPICE documentation set for completeness only. Most of this material has been replaced by more accurate, efficient, and useful methodologies.

U-element Digital and Mixed Mode Stimuli

HSPICE input netlists support two types of digital stimuli: digital vector files (described in the Chapter 9 section, [Specifying a Digital Vector File and Mixed Mode Stimuli](#)) and U-element digital input files, described in the following sections.

U-element Digital Input Elements and Models

This section describes the input file format for the digital input U Element.

In HSPICE, the U-element can reference digital input and digital output models for mixed-mode simulation. If you run HSPICE in standalone mode, the state information originates from a digital file. Digital outputs are handled in a similar fashion. In digital input file mode, the input file is named <design>.d2a, and the output file is named <design>.a2d.

A2D and D2A functions accept the terminal “\” backslash character as a line-continuation character, to allow more than 255 characters in a line. Use line continuation if the first line of a digital file, which contains the signal name list, is longer than the maximum line length that your text editor accepts.

Appendix B: Obsolete HSPICE Functionality

U-element Digital and Mixed Mode Stimuli

Do not put a blank first line in a digital D2A file. If the first line of a digital file is blank, HSPICE issues an error message.

Example

The following example demonstrates how to use the “\” line continuation character, to format an input file for text editing. The example file contains a signal list for a 64-bit bus.

```
...
a00 a01 a02 a03 a04 a05 a06 a07 \
a08 a09 a10 a11 a12 a13 a14 a15 \
... * Continuation of signal names
a56 a57 a58 a59 a60 a61 a62 a63 End of signal names
... Remainder of file
```

General Form

```
Uxxx interface nlo nhi mname SIGNAME=sname IS=val
```

Parameter	Description
Uxxx	Digital input element name. Must begin with U, followed by up to 1023 alphanumeric and special characters.
interface	Interface node in the circuit, to which the digital input attaches.
nlo	Node connected to the low-level reference.
nhi	Node connected to the high-level reference.
mname	Digital input model reference (U model).
SIGNAME	Signal name, as referenced in the digital output file header. Can be a string of up to eight alphanumeric characters.
IS	Initial state of the input element. Must be a state that the model defines.

Model Syntax

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

Digital-to-Analog Input Model Parameters

Table 109 Digital-to-Analog Parameters

Names (Alias)	Units	Default	Description
CLO	farad	0	Capacitance, to low-level node.
CHI	farad	0	Capacitance, to high-level node.
S0NAME			State 0 character abbreviation. A string of up to four alphanumerical characters.
S0TSW	sec		State 0 switching time.
S0RLO	ohm		State 0 resistance, to low-level node.
S0RHI	ohm		State 0 resistance, to high-level node.
S1NAME			State 1 character abbreviation. A string of up to four alphanumerical characters.
S1TSW	sec		State 1 switching time.
S1RLO	ohm		State 1 resistance, to low-level node.
S1RHI	ohm		State 1 resistance, to high-level node.
S19NAME			State 19 character abbreviation. A string of up to four alphanumerical characters.
S19TSW	sec		State 19 switching time.
S19RLO	ohm		State 19 resistance, to low-level node.
S19RHI	ohm		State 19 resistance, to high-level node.
TIMESTEP	sec		Step size for digital input files only.

To define up to 20 different states in the model definition, use the S_n NAME, S_n TSW, S_n RLO and S_n RHI parameters, where n ranges from 0 to 19.

[Figure 205](#) is the circuit representation of the element.

Appendix B: Obsolete HSPICE Functionality

U-element Digital and Mixed Mode Stimuli

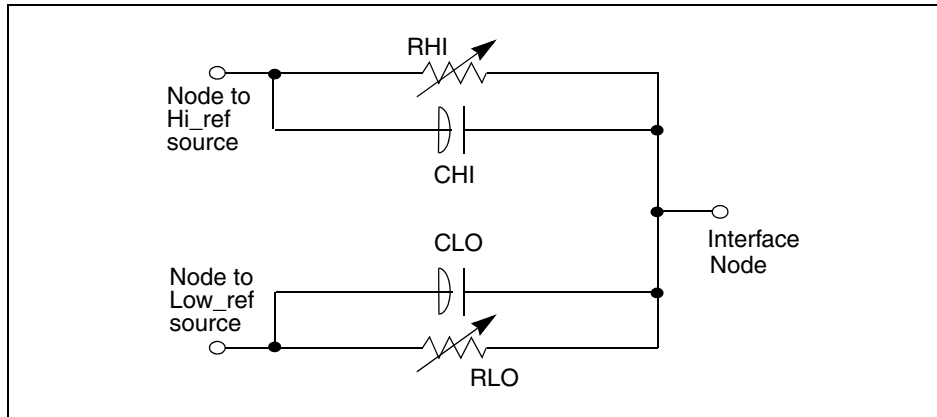


Figure 205 Digital-to-Analog Converter Element

Example

The following example shows how to use the U element and model, as a digital input for a HSPICE netlist.

You can find the sample netlist for this example in the following directory:

`$installdir/demo/hspice/sources/uelm.sp`

The associated digital input file is:

```
1
00 1:1
09 z:1
10 0:1
11 z:1
20 1:1
30 0:1
39 x:1
40 1:1
41 x:1
50 0:1
60 1:1
70 0:1
80 1:1
```

U Element Digital Outputs

Digital output (not supported in HSPICE RF).

Syntax

Uxxx interface reference mname SIGNAME=sname

Parameter	Description
Uxxx	Digital output element name. Must begin with U, followed by up to 1023 alphanumeric and special characters.
interface	Interface node in the circuit, at which HSPICE measures the digital output.
reference	Node to use as a reference for the output.
mname	Digital output model reference (U model).
SIGNAME	Signal name, as referenced in the digital output file header. A string of up to eight alphanumeric characters.

Model Syntax

.MODEL mname U LEVEL=4 *parameters...*

Analog-to-Digital Output Model Parameters

Table 110 Analog-to-Digital Parameters

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

Appendix B: Obsolete HSPICE Functionality
 U-element Digital and Mixed Mode Stimuli

Table 110 Analog-to-Digital Parameters (Continued)

Name (Alias)	Units	Default	Description
S1VLO	volt		State 1 low-level voltage.
S1VHI	volt		State 1 high-level voltage.
S19NAME			State 19 character abbreviation. A string of up to four alphanumerical characters.
S19VLO	volt		State 19 low-level voltage.
S19VHI	volt		State 19 high-level voltage.
TIMESTEP	sec	1E-9	Step size for digital input file.
TIMESCALE			Scale factor for time.

To define up to 20 different states in the model definition, use the S_nNAME , S_nVLO and S_nVHI parameters, where n ranges from 0 to 19. [Figure 206](#) shows the circuit representation of the element.

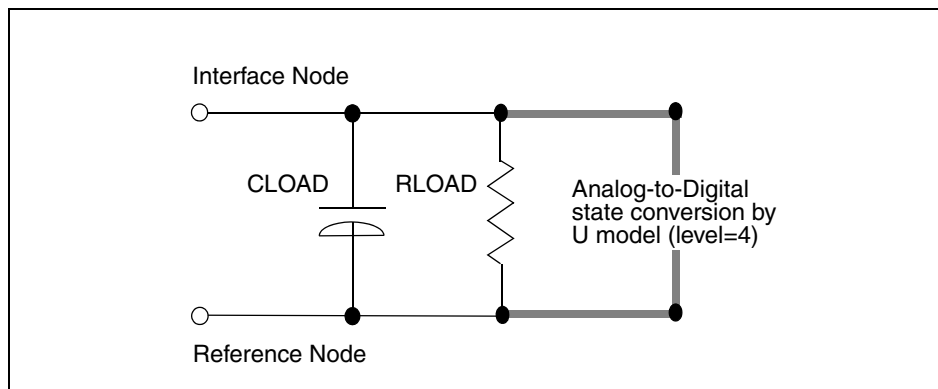


Figure 206 Analog-to-Digital Converter Element

Replacing Sources With Digital Inputs

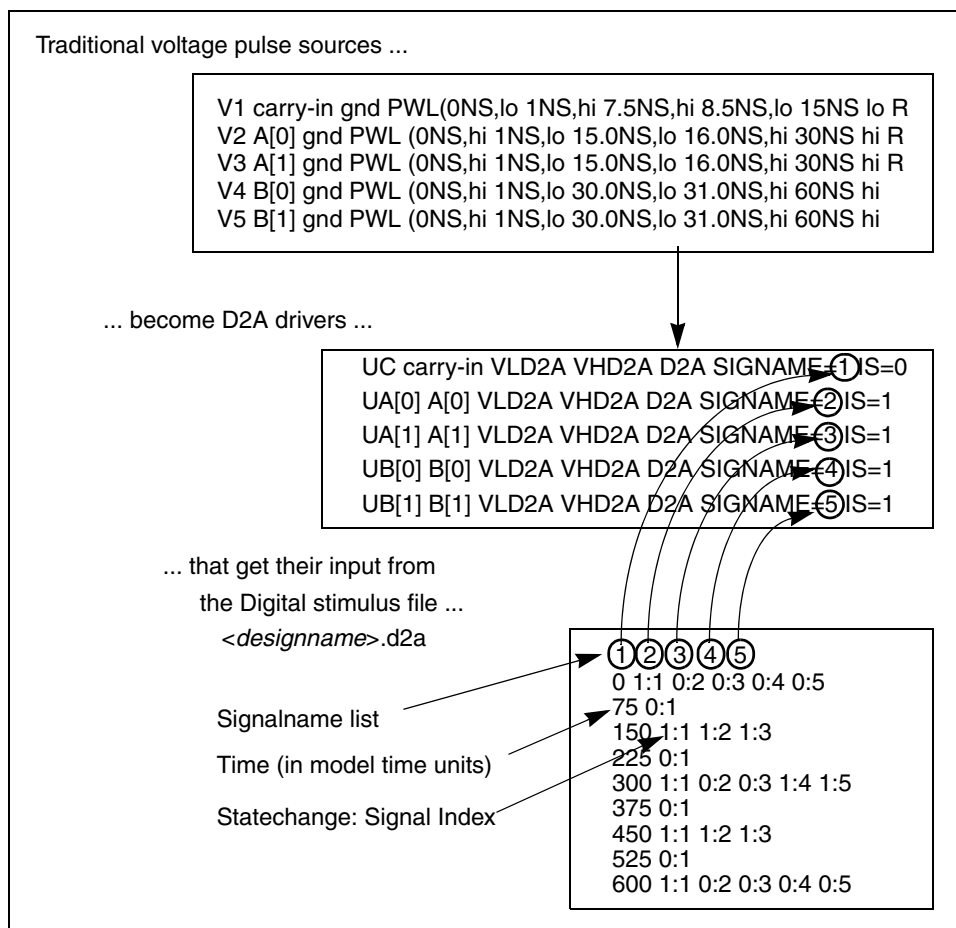


Figure 207 Digital File Signal Correspondence

Example

The following is an example of replacing sources with digital inputs. This example is based on demonstration netlist `digin.sp`, which is available in directory `$installdir/demo/hspice/cchar`:

Appendix B: Obsolete HSPICE Functionality

Replacing Sources With Digital Inputs

```
* EXAMPLE OF U-ELEMENT DIGITAL OUTPUT
.OPTION POST
VOUT carry_out GND PWL 0N 0V 10N 0V 11N 5V 19N 5V 20N 0V
+ 30N 0V 31N 5V 39N 5V 40N 0V
VREF REF GND DC 0.0V
UCO carry_out REF A2D SIGNAME=12
R1 REF 0 1k

* DEFAULT DIGITAL OUTPUT MODEL (no "X" value)

.MODEL A2D U LEVEL=4 TIMESTEP=0.1NS TIMESCALE=1
+ S0NAME=0 S0VLO=-1 S0VHI= 2.7
+ S4NAME=1 S4VLO= 1.4 S4VHI=9.0
+ CLOAD=0.05pf
.TRAN 1N 500N
.END
```

The digital output file should look something like this:

```
12
0      0:1
105    1:1
197    0:1
305    1:1
397    0:1
```

- *12* represents the signal name
- The first column is the time, in units of 0.1 nanoseconds.
- The second column has the signal value: signal index pairs. Signal index is corresponds to the position of signal name in the signal name list.
- This file uses more columns to represent subsequent outputs.

For another example, see the file identified and the plot in Figure 27.

\$installdir/demo/hspice/cchar/mos2bit.sp

See the plot in [Figure 208 on page 1071](#).

In this example, a 2-bit MOS adder uses a digital input file. In the plot, the `a[0]`, `a[1]`, `b[0]`, `b[1]`, and `carry-in` nodes all originate from a digital file input similar to [Figure 207 on page 1069](#). HSPICE outputs a digital file.

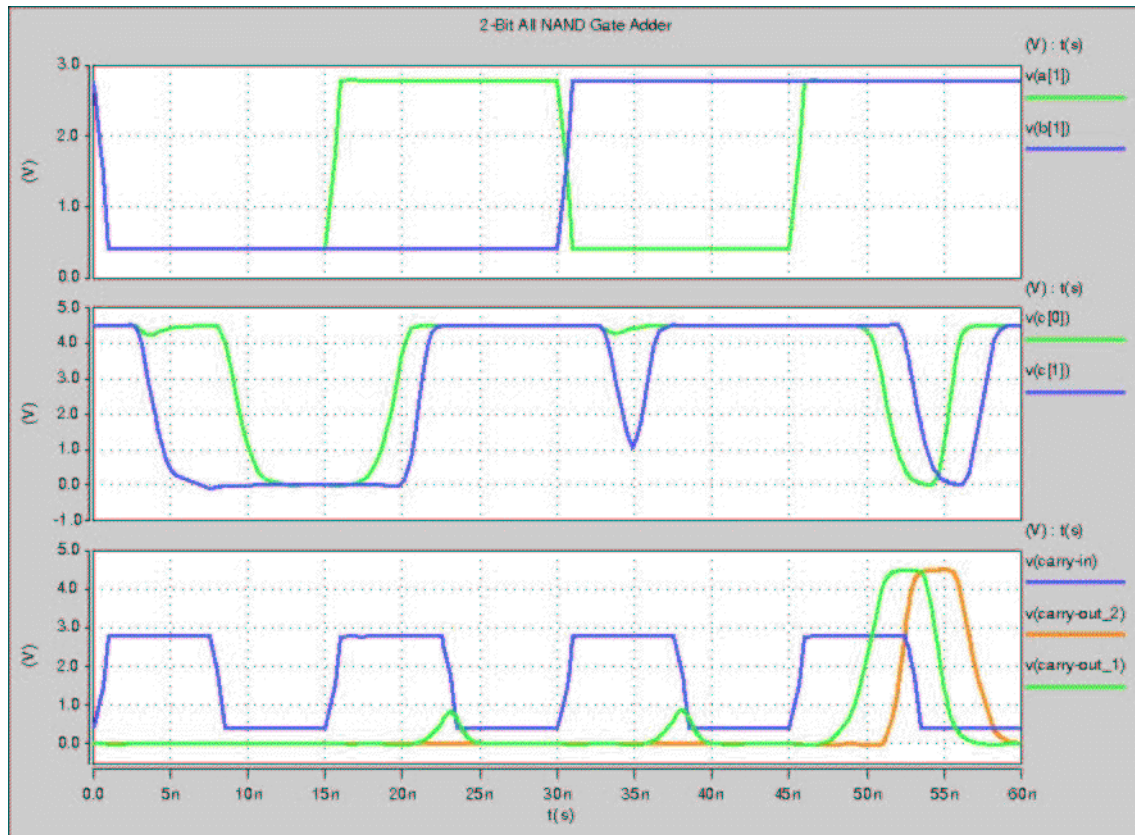


Figure 208 Digital Stimulus File Input

.NET Parameter Analysis

HSPICE uses the AC analysis results to perform network analysis. The .NET statement defines Z, Y, H, and S-parameters to calculate. The following list shows various combinations of the .NET statement for network matrices that HSPICE calculates:

```
.NET Vout Isrc   V = [Z]   [I]
.NET Iout Vsrc   I = [Y]   [V]
.NET Iout Isrc   [V1 I2]T = [H]   [I1 V2]T
.NET Vout Vsrc   [I1 V2]T = [S]   [V1 I2]T
([M]T represents the transpose of the M matrix).
```

Appendix B: Obsolete HSPICE Functionality

.NET Parameter Analysis

Note: The preceding list does not mean that you must use combination (1) to calculate Z parameters. However, if you specify `.NET Vout Isrc`, HSPICE initially evaluates the Z matrix parameters. It then uses standard conversion equations to determine S-parameters or any other requested parameters.

Figure 209 shows the importance of variables in the `.NET` statement. Here, `Isrc` and `Vce` are the DC biases, applied to the BJT.

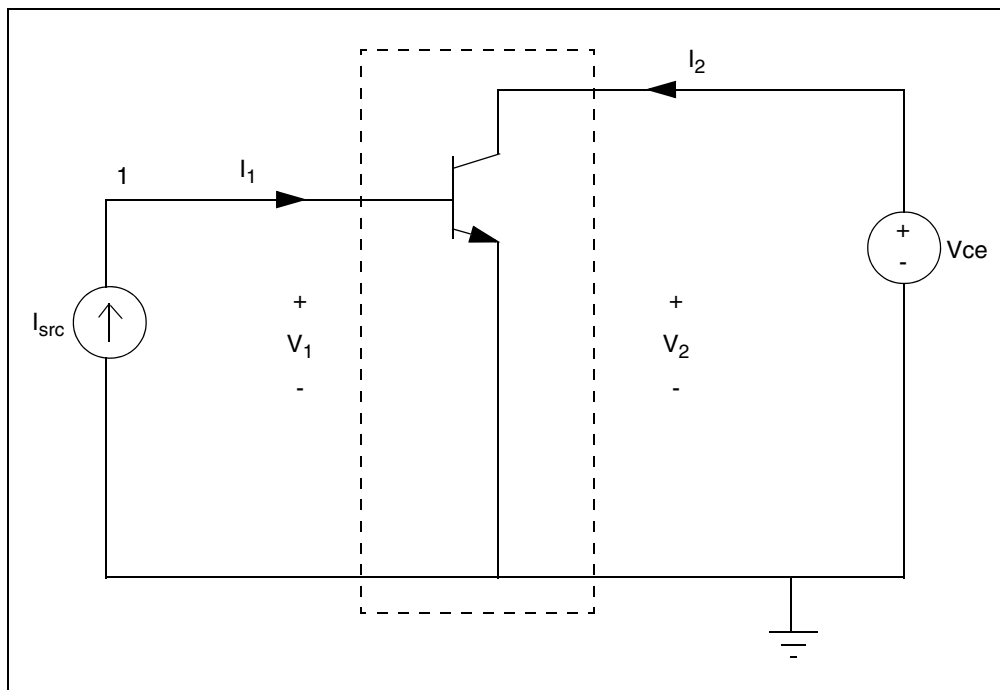


Figure 209 Parameters with `.NET V(2) Isrc`

This `.NET` statement provides an incorrect result for the Z parameter calculation:

```
.NET V(2) Isrc
```

When HSPICE runs AC analysis, it shorts all DC voltage sources; all DC current sources are open-circuited. As a result, `V(2)` shorts to ground and its value is zero in AC analysis. This affects the results of the network analysis.

In this example, HSPICE attempts to calculate the Z parameters (Z_{11} and Z_{21}), defined as $Z_{11}=V_1/I_1$ and $Z_{21}=V_2/I_1$ with $I_2=0$. The above example does

not satisfy the requirement that I_2 must be zero. Instead, V_2 is zero, which results in incorrect values for Z_{11} and Z_{21} .

Figure 210 shows the correct biasing configurations for performing network analysis for the Z, Y, H, and S-parameters.

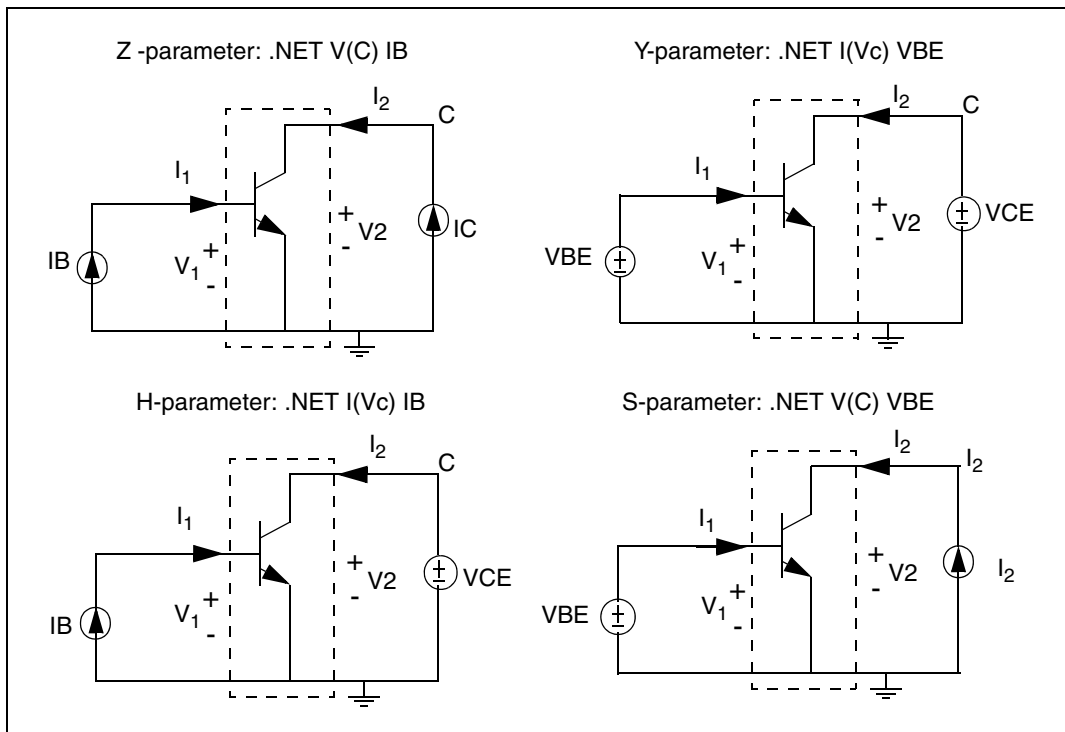


Figure 210 Network Parameter Configurations

Example

To calculate the H parameters, HSPICE uses the .NET statement.

```
.NET I (VC) IB
```

V_C denotes the voltage at the C node, which is the collector of the BJT. With this statement, HSPICE uses the following equations to calculate H parameters immediately after AC analysis:

$$V_1 = H_{11} \cdot I_1 + H_{12} \cdot V_2$$

$$I_2 = H_{21} \cdot I_1 + H_{22} \cdot V_2$$

Appendix B: Obsolete HSPICE Functionality

Behavioral Modeling, Obsolete Functionality

To calculate Hybrid parameters (H11 and H21), the DC voltage source (V_{CE}) sets V2 to zero, and the DC current source (IB) sets I1 to zero. Setting I1 and V2 to zero, precisely meets the conditions of the circuit under examination: the input current source is open-circuited, and the output voltage source shorts to ground.

A data file containing measured results can drive external DC biases applied to a BJT. Not all DC currents and voltages (at input and output ports) might be available. When you run a network analysis, examine the circuit and select suitable input and output variables. This helps you to obtain correctly calculated results. The following example demonstrates HSPICE network analysis of a BJT.

Behavioral Modeling, Obsolete Functionality

Digital Stimulus Files

Complex transition files are difficult to process, if you use piecewise linear sources. You can use the A2D and D2A conversion functions, in the U-element, to simplify processing of transition files.

- The A2D function converts analog output to digital data.
- The D2A function converts digital input data to analog.

You can also export the output to either logic or VHDL simulators.

Op-Amp Subcircuit Generators (Behavioral Modeling)

The subcircuit generator automatically designs operational amplifiers, to meet electrical specifications (such as PSRR, CMRR, and V_{os}). The generator produces component values, for each element in the design. When HSPICE combines these values, the resulting subcircuits simulate faster than conventional circuit-level implementations.

Op-Amp Model Generator

HSPICE uses the model generator to automatically design and simulate both board-level and IC op-amp designs. Here's how:

1. Start from the existing electrical specifications for a standard industrial operational amplifier.
2. Enter the specifications in the op-amp model statement.
HSPICE automatically generates the internal components of the op-amp, to meet the specifications.
3. You can then call the design from a library, for a board-level simulation.

The HSPICE op-amp model is a subcircuit. It is about 20 times faster to simulate, than an actual transistor level op-amp. You can adjust the AC gain and phase to within 20 percent of the actual measured values, and set the transient slew rates accurately. This model does not contain high-order frequency response poles and zeros, so it can significantly differ from actual amplifiers, in predicting high-frequency instabilities.

You can use this model to represent normal amplifier characteristics, including:

- input offsets
- small signal gain
- transient effects

The op-amp subcircuit generator consists of a model, and one or more elements. Each element is in the form of a subcircuit call.

1. The model generates an output file of the op-amp equivalent circuit, which you can collect into libraries.
The file name is the name of the model (mname), with an `.inc` extension.
2. After you generate the output file, other HSPICE input files can reference this subcircuit, using a `.SUBCKT` call to the model name.
3. The `.SUBCKT` call automatically searches for the file in the present directory.
4. It then searches the directories specified in any `.OPTION SEARCH = 'directory_path_name'`.
5. Finally, it searches the directory where the DDL (Discrete Device Library) is located.

The amplifier element references the amplifier model.

If the model generator creates op-amp model that do not converge in DC analysis, use the `.IC` or `.NODESET` statement, to set the input nodes to the voltage that is halfway between the VCC and VEE. This balances the input nodes, and stabilizes the model.

Op-Amp Element Statement Format

COMP=0 (internal compensation)

```
xa1 in- in+ out vcc vee modelname AV=val
```

COMP=1 (external compensation)

```
xa1 in- in+ out comp1 comp2 vcc vee modelname AV=val
```

Parameter	Description
in-	Inverting input
in+	Non-inverting input
out	Output, single ended
vcc	Positive voltage supply
vee	Negative voltage supply
modelname	Subcircuit reference name

Op-Amp .MODEL Statement Format

```
.MODEL mname AMP parameter=value ...
```

Parameter	Description
mname	Model name. Elements use this name to reference the model.
AMP	Identifies an amplifier model (HSPICE only).
parameter	Any model parameter, as described following this table.
value	Value assigned to a parameter.

Example

```
X0 IN- IN+ OUT0 VCC VEE ALM124
.MODEL ALM124 AMP
+ C2= 30.00P SRPOS= .5MEG SRNEG= .5MEG
+ IB= 45N IBOS= 3N VOS= 4M
+ FREQ= 1MEG DELPHS= 25 CMRR= 85
+ ROUT= 50 AV= 100K ISC= 40M
+ VOPOS= 14.5 VONEG= -14.5 PWR= 142M
+ VCC= 16 VEE= -16 TEMP= 25.00
+ PSRR= 100 DIS= 8.00E-16 JIS= 8.00E-16
```

Op-Amp Model Parameters

Table 111 shows the model parameters for op-amps. The defaults for these parameters depend on the DEF parameter setting. Table 112 on page 1083 shows defaults for each of the three DEF settings.

Table 111 Op-Amp Model Parameters

Names (Alias)	Units	Default	Description
AV (AVD)	volt/volt		Amplifier gain, in volts out, per volt in. The DC ratio of the voltage in, to the voltage out. Typical gains are from 25k to 250k. If the frequency is too low, increase the negative and positive slew rates, or decrease DELPHS.
CMRR	volt/volt		Common mode rejection ratio. This is usually between 80 and 110 dB. You can enter this value as 100 dB, or as 100000.
AV1K	volt/volt		Amplifier gain, at 1 kHz. Estimates the unity-gain bandwidth. You can express gain as actual voltage gain, or in decibels (a standard unit conversion). If you set AV1K, HSPICE ignores FREQ. A typical value for AV1K is AV1K=(unity gain freq)/1000.

Appendix B: Obsolete HSPICE Functionality
Behavioral Modeling, Obsolete Functionality

Table 111 Op-Amp Model Parameters (Continued)

Names (Alias)	Units	Default	Description
C2	farad		<p>Internal feedback compensation capacitance. For an internally compensated amplifier, if you do not specify a capacitance value, the default is 30 pF.</p> <p>If the gain is high (above 500k), the internal compensation capacitor is probably different (typically 10 pF).</p> <p>For an externally compensated amplifier (COMP=1), set C2 to 0.5 pF, as the residual internal capacitance.</p>
COMP			<p>Compensation level selector. If set to 1, it modifies the number of equivalent nodes, to include external compensation nodes. See C2 for external compensation settings.</p> <p>COMP=0 internal compensation (default).</p> <p>COMP=1 external compensation.</p>
DEF			<p>Default model selector. Choose one of these:</p> <p>0= generic (0.6 MHz bandwidth) (default)</p> <p>1= ua741 (1.2 MHz bandwidth)</p> <p>2= mc4560 (3 MHz bandwidth).</p>

Table 111 Op-Amp Model Parameters (Continued)

Names (Alias)	Units	Default	Description
DELPHS	deg		<p>Excess phase, at the unity gain frequency. Also called the <i>phase margin</i>. HSPICE measures DELPHS in degrees. Typical excess phases range from 5° to 50°.</p> <p>To determine DELPHS, subtract the phase at unity gain from 90°.</p> <p>The result is the phase margin.</p> <p>Use the same chart as used for the FREQ determination above.</p> <p>DELPHS interacts with FREQ (or AV1K). Values of DELPHS tend to lower the unity gain bandwidth, especially values greater than 20°.</p> <p>Pick the DELPHS closest to measured value, that does not reduce unity gain bandwidth more than 20%.</p> <p>Otherwise, the model might not have enough poles, to always return correct phase and frequency responses.</p>
DIS	amp	1e-16	Saturation current, for diodes and BJTs.

Appendix B: Obsolete HSPICE Functionality
Behavioral Modeling, Obsolete Functionality

Table 111 Op-Amp Model Parameters (Continued)

Names (Alias)	Units	Default	Description
FREQ (GBW, BW)	Hz		<p>Unity gain frequency, measured in hertz. Typical frequencies are from 100 kHz to 3 MHz. If you do not specify this parameter, measure the open-loop frequency response at 0 dB voltage gain, and measure the actual compensation capacitance. Typical compensation is 30 pF, and single-pole compensation configuration.</p> <p>If AV1K > zero, HSPICE calculates unity gain frequency from AV1K; it ignores FREQ.</p>
IB	amp		<p>Amount of current required to bias the input differential transistors. This is usually a fundamental electrical characteristic. Typical values are between 20 and 400 nA.</p>
IBOS	amp		<p>Input bias offset current, or <i>input offset current</i>. Amount of unbalanced current, between input differential transistors. Usually a fundamental electrical characteristic. Typical values are 10% to 20% of the IB.</p>
ISC	amp		<p>Input short circuit current – not always specified. Typical values are 5 to 25 mA. HSPICE can determine ISC from output characteristics (current sinking), as the maximum output sink current. ISC and ROUT interact with each other. If ROUT is too large for the ISC value, HSPICE reduces ROUT.</p>
JIS	amp		<p>JFET saturation current. Default=1e-16. You do not need to change this value.</p>
LEVIN			<p>Input level type selector. You can create only a BJT differential pair. LEVIN=1 BJT differential input stage.</p>

Table 111 Op-Amp Model Parameters (Continued)

Names (Alias)	Units	Default	Description
LEVOUT			Output level type selector. You can create only a single-ended output stage. LEVOUT=1 single-ended output stage.
MANU			Manufacturer's name. Add this to the model parameter list, to identify the source of model parameters. HSPICE prints the name in the final equivalent circuit.
PWR (PD)	watt		Total power dissipation, for the amplifier. Includes a calculated value for the op-amp input differential pair. If you set a high slew rate, and very low power, HSPICE issues a warning, and shows power dissipation only for an input differential pair.
RAC (r0ac, roac)	ohm		High-frequency output resistance. This typically is about 60% of ROUT. RAC usually ranges between 40 to 70 ohms, for op-amps with video drive capabilities.
ROUT	ohm		Low-frequency output resistance. To find this value, use the closed-loop output impedance graph. The impedance at about 1kHz, using the maximum gain, is close to ROUT. Gains of 1,000 and above show effective DC impedance, generally in the frequency region between 1k and 10 kHz. Typical ROUT values are 50 to 100 ohms.
SRNEG (SRN)	volt		Negative output slew rate. HSPICE extracts this value from a graph that shows the response for the voltage follower pulse. This is usually a 4 V or 5 V output change, with 10 to 20 V supplies. Measures the negative change in voltage, and the amount of time for the change.

Appendix B: Obsolete HSPICE Functionality
Behavioral Modeling, Obsolete Functionality

Table 111 Op-Amp Model Parameters (Continued)

Names (Alias)	Units	Default	Description
SRPOS (SRP)	volt		Positive output slew rate. HSPICE extracts this value from a graph that shows the response for a voltage follower pulse. This is usually a 4 V or 5 V output change, with 10 to 20 V supplies. Measures the positive change in voltage, and the amount of time for the change. Typical slew rates are from 70 k to 700 k.
TEMP	°C		Temperature, in degrees Celsius. Usually the temperature at which HSPICE measured model parameters, which is typically 25 °C.
VCC	volt		Positive power-supply reference voltage, for VOPOS. HSPICE measures the VOPOS amplifier, with respect to VCC.
VEE	volt		Negative power-supply voltage. HSPICE measures the VONEG amplifier, with respect to VCC.
VONEG (VON)	volt		Maximum negative output voltage. This is less than VEE (the negative power-supply voltage), by the internal voltage drop.
VOPOS (VOP)	volt		Maximum positive output voltage. This is less than VCC (the positive power supply voltage), by the internal voltage drop.
VOS	volt		Required input offset voltage, between input differential transistors, which zeroes output voltage. Usually a fundamental electrical characteristic. Typical values for bipolar amplifiers range from 0.1 mV to 10 mV. HSPICE measures VOS in volts. In some amplifiers, VOS can cause a failure to converge. If this occurs, set VOS to 0, or use the initial conditions for convergence.

Op-Amp Model Parameter Defaults

Table 112 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	25x	17x	52x
DIS	Diode saturation current	8e-16	8e-16	8e-16
FREQ	Frequency, for unity gain	600 k	-	-
IB	Current, for input bias	30 n	250 n	40 n
IBOS	Current, for input bias offset	1.5 n	0.7 n	5 n
ISC	Current, for output short circuit	25 mA	25 mA	25 mA
LEVIN	Circuit-level selector, for input	1	1	1
LEVOUT	Circuit-level selector, for output	1	1	1
MANU	Manufacturer's name	-	-	-
PWR	Power dissipation	72 mW	60 mW	50 mW
RAC	AC output resistance	0	75	70
ROUT	DC output resistance	200	550	100
SRPOS	Positive output slew rate	450 k	1 meg	1 meg

Table 112 Op-Amp Model Parameter Defaults (Continued)

Parameter	Description	Defaults		
		DEF=0	DEF=1	DEF=2
SRNEG	Negative output slew rate	450 k	800 k	800 k
TEMP	Temperature of model	25 deg	25 deg	25 deg
VCC	Positive supply voltage, for VOPOS	20	15	15
VEE	Negative supply voltage, for VONEG	-20	-15	-15
VONEG	Maximum negative output	-14	-14	-14
VOPOS	Maximum positive output	14	14	14
VOS	Input offset voltage	0	0.3 m	0.5 m

Simulation Results

The simulation results include the DC operating point analysis, for an input voltage of 0 v, and power supply voltages of ± 15 v.

- The DC offset voltage is 3.3021 mV, which is less than that specified for the original VOS specification, in the op-amp .MODEL statement.
- The unity-gain frequency is 907.885 kHz, which is within 10% of the 1 MHz that the FREQ parameter (in the .MODEL statement) specifies.
- The required time rate, for a 1 V change in the output (from the .MEASURE statement), is 2.3 μ s (from the SRPOS simulation result listing). This provides a slew rate of 0.434 mV/s, which is within about 12% of the 0.5 mV/s, specified in the SRPOS parameter of the .MODEL statement.
- The negative slew rate is almost exactly 0.5 mV/s, which is within 1% of the slew rate specified in the .MODEL statement.

Example

```

$$ FILE ALM124.SP
.OPTION NOMOD AUTOSTOP SEARCH=' '
.OP VOL
.AC DEC 10 1HZ 10MEG
.MODEL PLOTDB PLOT XSCAL=2 YSCAL=3
.MODEL PLOTLOGX PLOT XSCAL=2
.GRAPH AC MODEL=PLOTDB VM(OUT0)
.GRAPH AC MODEL=PLOTLOGX VP(OUT0)
.TRAN 1U 40US 5US .15MS
.GRAPH V(IN) V(OUT0)
.MEASURE TRAN 'SRPOS'TRIG V(OUT0) VAL=2V RISE=1
+ TARG V(OUT0) VAL=3V RISE=1
.MEASURE TRAN 'SRNEG'TRIG V(OUT0) VAL=-2V FALL=1
+ TARG V(OUT0) VAL=-3V FALL=1
.MEASURE AC 'UNITFREQ'TRIG AT=1
+ TARG VDB(OUT0) VAL=0 FALL=1
.MEASURE AC 'PHASEMARGIN' FIND VP(OUT0)
+ WHEN VDB(OUT0)=0
.MEASURE AC 'GAIN(DB)'MAX VDB(OUT0)
.MEASURE AC 'GAIN(MAG)'MAX VM(OUT0)
VCC VCC GND +15V
VEE VEE GND -15V
VIN IN GND AC=1 PWL 0US 0V 1US 0V 1.1US +10V 15US +10V
+ 15.2US -10V 100US -10V
.MODEL ALM124 AMP
+ C2= 30.00P SRPOS= .5MEG SRNEG= .5MEG
+ IB= 45N IBOS= 3N VOS= 4M
+ FREQ= 1MEG DELPHS= 25 CMRR= 85
+ ROUT= 50 AV= 100K ISC= 40M
+ VOPOS= 14.5 VONEG= -14.5 PWR= 142M
+ VCC= 16 VEE= -16 TEMP= 25.00
+ PSRR= 100 DIS= 8.00E-16 JIS= 8.00E-16
*
```

Unity Gain Resistor Divider Mode

```
*
Rfeed  OUT0  IN-  10K  AC=10000G
RIN    IN    IN-  10K
RIN+   IN+   GND  10K
X0     IN-   IN+   OUT0  VCC VEE ALM124
ROUT0  OUT0  GND  2K
COUT0  OUT0  GND  100P
.END

***** OPERATING POINT STATUS IS VOLTAGE
***** SIMULATION TIME IS 0.
  NODE      =VOLTAGE  NODE      =VOLTAGE  NODE      =VOLTAGE
+ 0:IN      = 0.      0:IN+    =-433.4007U  0:IN-    = 3.3021M
+ 0:OUT0    = 7.0678M  0:VCC    = 15.0000  0:VEE    = -15.0000
unitfreq    = 907.855K  TARG     = 907.856K  TRIG     = 1.000
PHASEMARGIN = 66.403
gain(db)    = 99.663  AT      = 1.000
FROM        = 1.000  TO      = 10.000X
gain(mag)   = 96.192K  AT      = 1.000
FROM        = 1.000  TO      = 10.000X
srpos       = 2.030U  TARG    = 35.471U  TRIG     = 33.442U
srneg       = 1.990U  TARG    = 7.064U  TRIG     = 5.074U
```

HSPICE Parser Strict Syntax Requirements

Discusses HSPICE requirements for efficient usage with regard to improved HSPICE parser.

In 2009.03 (2009.03-SP1 for the Windows release), HSPICE implemented a new, more efficient front-end parser. As a result, certain relaxed syntax which was previously allowed is no longer allowed or is interpreted differently. This list contains examples that users may commonly encounter.

Listing of Tighter Syntax Restrictions

The following restrictions are in effect for the current release:

1. Suppression of netlist echo in the listing file.

Although not a syntax restriction, many users will notice the suppression of the netlist being echoed to the listing file. This can make the listing file much smaller and easier to find errors and warnings in. (Alternatively, use the WARN_SEP option to list errors and warnings in a file separate from the *.lis file.) If you wish to restore the netlist inclusion in the listing file, you can set the variable:

```
HSP_LIS_200809=1
```

This feature became available with the 2009.03-SP1 release in all platforms. It is not intended to make the old and new netlists identical but to restore the netlist to the listing file as a convenience for debugging.

2. Use of quotes around the entry name in a .lib statement. Previously,

```
.lib 'typical.lib' 'slow'
```

Appendix C: HSPICE Parser Strict Syntax Requirements

Listing of Tighter Syntax Restrictions

or

```
.lib 'typical.lib' slow
```

Would match

```
.lib slow
```

Now, the entry name must exactly match in both the .lib call and the .lib definition. If quotes (or any other allowed characters) are used in the .lib call, they are taken literally and must be the same in the .lib definition. For example:

```
slow=slow  
'slow'='slow'
```

Resulting error:

```
.lib entry "'slow'" cannot be found
```

3. Use of “.” in front of the keyword “lib” in .del lib:

Previously allowed:

```
.del .lib 'typical.lib' slow
```

Now required:

```
.del lib 'typical' slow
```

4. Omission of quotes in expression syntax.

Previously allowed:

```
.param a=5  
.param b=-a
```

Now required:

```
.param b='-a'
```

Resulting error message:

```
**error** (top.sp:11) unknown word before or at "-"
```

5. Unnecessary use of quotes in parameter name during assignment.

Previously allowed:

```
.param a=5  
.param 'b'='-a' $OR  
.param 'b'=a
```

Now required:

```
.param a=5  
.param b=a $OR  
.param b='-a'
```

Resulting error message:

```
syntax error when fetching "'c'" "=" "'a'"
```

6. Data name in .STIM command may no longer be in quotes.

Previously allowed:

```
.stim tran data 'abcde' par(time) i(r1)
```

Now required

```
.stim tran data abcde par(time) i(r1)
```

Resulting error message:

```
syntax error when fetching "'abcde'"
```

7. Use of equal sign following par in measure statements:

Previously allowed:

```
.measure tran i_avg avg par='(-1)*I(vdd)' from=1n to=3n
```

Now required:

```
.measure tran i_avg avg par '(-1)*I(vdd)' from=1n to=3n
```

Resulting error message:

```
syntax error when fetching "par"
```

8. Printzo not interchangeable with printz0. Previously, Z0 (z-zero) was interchangeable with Zo (z-letter o). Now, you must only use Zo.

Previously allowed:

```
W1 in1 in2 gnd out1 out2 gnd FSmodel=U_MB N=2 l=0.5 PRINTZ0=poi  
1 1e9
```

Now requires:

```
W1 in1 in2 gnd out1 out2 gnd FSmodel=U_MB N=2 l=0.5 PRINTZo=poi  
1 1e9
```

Resulting error message:

```
syntax error when fetching "1" "1e9"
```

Appendix C: HSPICE Parser Strict Syntax Requirements

Listing of Tighter Syntax Restrictions

9. Parameter sweeps in the form start-stop-step would formerly allow omission of the actual keywords start, stop and step for the first and subsequent sweeps. Now, only one parameter sweep is allowed without these keywords, although they are recommended in all cases:

```
.tran 1u 5e-3 sweep vddval 1 3 .2 <- OK
```

```
.tran 1u 5e-3 sweep vddval 1 3 .2 4 5 .25 <- Not OK
```

```
.tran 1u 5e-3 sweep vddval start=1 stop=3 step=.2 <- OK
```

```
.tran 1u 5e-3 sweep vddval start=1 stop=3 step=.2 start=4  
stop=5 step=.25 <- OK
```

10. Keywords in .MEASURE statements such as FROM, TO and TD are no longer allowed when also used as parameter names in .MEASURE statements. In this example, using a parameter named “td” is no longer allowed:

Previously allowed:

```
.param td=500p  
.MEAS TRAN iddq AVG I(VDDQ) FROM 0 TO td
```

Now requires:

```
.param td1=500p  
.MEAS TRAN iddq AVG I(VDDQ) FROM 0 TO td1
```

11. Verilog-A search patch is context sensitive. Previously, if a Verilog-A source file was referenced from inside a sub-directory, it would attempt to start searching at the design root as well. Now, it applies the search rules as defined in the HSPICE manual:

1. Current working directory
2. Path defined by -hdlpath
3. Path defined by HSP_HDL_PATH

In the failure scenario, the netlist includes a file that calls Verilog-A. The include file and the Verilog-A file are in separate subdirectories. Previously, HSPICE would start looking from both the design root and the context of the sub-directory.

Previously allowed

```
.hdl 'vfiles/buffer.va'
```


Appendix C: HSPICE Parser Strict Syntax Requirements

Listing of Tighter Syntax Restrictions

Now required:

```
.hdl '../vfiles/buffer.va'
```

Resulting error message:

```
file...could not be found in the HSPICE Verilog-A search path
```

12. For a full listing of keywords that cannot be used as parameter or node names in element lines of command statements and keywords that are illegal in specified commands, see [Reserved Keywords](#) in Chapter 4 of this manual.

Appendix C: HSPICE Parser Strict Syntax Requirements
Listing of Tighter Syntax Restrictions

Symbols

!GND node 96
\$installdir installation directory 116

Numerics

3DES encryption 158
3D-IC 671
8b-10b encoding 263

A

A2D
 function 1063, 1074
 model parameter 1063
 output model parameters 1067
 See also mixed mode
.a2d file 25, 1063
ABS element parameter 326
abs(x) function 380
ABSI option 481
ABSMOS option 481
absolute
 power function 380
 value function 380
 value parameter 326
ABSV option 481
AC analysis 398
 output 418
 RC network 503
 resistance 502
 small signals 501
 sources 239
AC analysis measurement results file 26
AC analysis results file 26
AC choke inductor 185
.AC statement 700, 913
.ac# file 25, 26
access functions
 Get_E() 763, 765
 Get_P() 764
accuracy

 simulation time 481
 tolerance 479, 480
ACmatch analysis 847
acos(x) function 380
active component model name keywords, string
 parameters 390
adder
 circuit 968
 demo 967
 NAND gate binary 968
 subcircuit 967
admittance
 AC input 422
 AC output 422
 Y parameters 418
AGAUSS keyword 713
aids for convergence 1019
algebraic
 equations, example 606
 expressions 379
algorithms
 BDF 535
 Damped Pseudo Transient algorithm 489
 GEAR 533
 gear 534
 integration 533
 Levenberg-Marquardt 920
 linear acceleration 643
 Muller 576, 577
 numerical integration 533
 numerical integration controls 533
 RUNLVL 530
 trapezoidal 534
 trapezoidal integration 533
.ALTER
 blocks 101–102
 statement 102, 105, 403
AM
 modulation frequency 562
 source function 260, 260
AMP model parameter 1076
amplifiers, pole/zero analysis 584, 586

Index

B

analog transition data file 24

analyses

Monte Carlo 785

analysis

AC 398

accuracy 480–481

data driven 697, 698

DC 397

element template 398

FFT

AM modulation 562

modulator/demodulator 564

test circuit 569

windows 554

initialization 470

inverter 526

.MEASURE statement 398

Monte Carlo 698, 709, 709–734

optimization 913

parametric 398

pole/zero 575

active low-pass filter 587

CMOS differential amplifier 584

high-pass Butterworth filter 583

Kerwin's circuit 582

overview 575

simple amplifier 586

using Muller method 576

pulse width 946

RC network 503, 525

setup time 942

spectrum 549

statistical 701–734

Taguchi 697

temperature 697, 699

timing 935

transient 397, 520

worst case 697, 701–734

yield 697

analysis error messages 1018

annotate files 877

application notes 12

arccos(x) function 380

arcsin(x) function 380

arctan(x) function 380

arithmetic operators 380

ASIC libraries 117

asin(x) function 380

atan(x) function 380

ATEM characterization system 116

AUNIF keyword 713

autoconvergence 483

autoconvergence flow, DCoperating point 484

auto-convergence warnings 1015

AUTOSTOP option 603

AV model parameter 1077

AV1K model parameter 1077

AVD model parameter 1077

average deviation 698

average value, measuring 429

B

B# node name in CSOS 97

back-annotation, data mining 877

back-annotation, post-layout 650

Bartlett FFT analysis window 555, 557, 571

batch job list, MS Windows launcher 136

behavioral

current source 315

digital stimulus files 1074

voltage source 295

behavioral capacitors 178

behavioral resistors 172

Biaschk 536

binary search 939–??

Bipolar Junction Transistors. *See* BJTs

bisection 941

command syntax 940

function 935

measurement 938

methodology 938

optimization 938

overview 936

pass-fail method 938

pulse width analysis 946

pushout method 947

requirements 939

results plots 944

setup time analysis 942

transient analysis 942

violation analysis 938

bisection analysis warnings 1014

BJTs

current flow 413

- element template listings 442
 - elements, names 220
 - power dissipation 415
 - S-parameters, optimization 925
 - Blackman FFT analysis window 556, 573
 - Blackman-Harris FFT analysis window 556, 573
 - block elements 179
 - bond wire example 973
 - branch current
 - output 410
 - buffers, IBIS 231
 - BW model parameter 1080
- C**
- C2 model parameter 1078
 - calculating 48
 - calculating new measurements
 - new measurements 48
 - capacitance
 - element parameter 174
 - manufacturing variations 721
 - pins 600
 - capacitor
 - conductance requirement 488
 - current flow 412
 - element 173, 176, 439
 - frequency-dependent 177
 - models 174
 - voltage controlled 318, 321
 - CCCS element parameter 304
 - CCVS element parameter 325, 326
 - CDPL 50
 - C-element (capacitor) 176
 - cell characterization 697
 - cell characterization, advanced 602
 - cell characterization, examples 603
 - cell measurements, basic 596
 - CFL function 4
 - characterization of models 477
 - choke elements 179
 - circuits
 - adder 968
 - description syntax 84
 - inverter, MOS 526
 - nonconvergent 492
 - RC network 504
 - reusable 108
 - subcircuit numbers 96
 - temperature 700
 - test, FFT analysis 569
 - See also* subcircuits
 - client/server mode 40
 - client 42
 - quitting 43
 - server 41
 - simulating 42
 - starting 41
 - CLOAD model parameter 1067
 - clock source, random jitter 355
 - CMI 5
 - CMOS
 - differential amplifier, pole/zero analysis 584
 - output driver demo 973
 - tristate buffer, optimization 922
 - CMRR model parameter 1074, 1077
 - commands
 - hspice 34, 49
 - hspicerf 39
 - limit descriptors 403
 - .MOSRA 610
 - .MOSRAPRINT 618
 - output 396
 - comment line
 - netlist 88
 - VEC files 366
 - Common Distributed Processing Library 50
 - COMP model parameter 1076, 1078
 - Compiled Function Library 4
 - compressed netlists 83
 - compute farm 50
 - condition-controlled netlists
 - netlist
 - condition-controlled 106
 - conductance
 - for capacitors 488
 - pn junction 495
 - conductance, negative 1011
 - confidence interval levels, data mining 862
 - configuring text editor in Windows 142
 - configuration
 - MS Windows launcher 131
 - continuation of line
 - netlist 88
 - control option warnings 1012

Index

D

- control options
 - algorithm selection 479
 - convergence 479
 - DC convergence 479
 - initialization 478
 - method 528
 - printing 403
 - transient analysis
 - method 528–??
- controlled sources 273, 275
- CONVERGE option 482, 489
- convergence
 - error causes 1019
 - problems 489
 - analyzing 490
 - autoconverge process 483
 - causes 492
 - CONVERGE option 489
 - DCON setting 483
 - diagnosing 489–495
 - diagnostic tables 489
 - floating point overflow 489
 - GMINDC ramping 483
 - op-amp models 1076
 - pole/zero analysis 577
 - reducing 486
 - remedies 1019
- convergence error messages 1018
- convergence termination criteria 1018
- convergence/conductance error messages 1021
- convergence-diode resistance errors/solutions 1022
- corner files 873
- cos(x) function 380
- cosh(x) function 380
- current
 - branch 411
 - controlled
 - current sources 274, 304, 440
 - voltage sources 274, 325, 441
 - in HSPICE elements 411
 - output 409
 - sources 308
- custom CMI 5
- Custom WaveView 133
- C-V plots 969

D

- D2A
 - function 1063, 1074
 - input model parameters 1065
 - model parameter 1063
 - See also* mixed mode
 - .d2a file 1063
- Damped Pseudo Transient algorithm 489
- data
 - encryption 146
 - sheet parameters 595
- data mining 873
 - annotate file 877
 - back-annotation 877
 - cannotate file 877
 - confidence intervals 862
 - corner files 873
 - external sampling 873
 - extremes and corners 867
 - other_percentiles 865
 - response correlation matrix 870
 - sample momentse 861
 - sample quartiles 864
 - summary statistics 860
 - variable screening 868
- .DATA statement 98, 603
- data-driven analysis 98
- data-driven analysis 697, 698
 - PWL source function 255
- db(x) function 381
- DC
 - analysis 397, 478–479
 - capacitor conductances 488
 - initialization 478
 - convergence control options 479
 - errors, reducing 486
 - matching 836
 - operating point
 - analysis 473
 - initial conditions file 24
 - See also* operating point
 - operating-point convergence 484
 - sources 239
 - sweep 476
- DC analysis measurement results file 26
- DC analysis results file 26
- DC block elements 179
- DC mismatch 836

- .DC statement 476, 700, 913
- DC/OP analyses warnings 1015
- DCCAP option 969
- DCmatch 836
- .DCMATCH output tables file 29
- DCON option 482
- DCSTEP option 488
- DDL 116, 117
- DDLPATH environment variable 117
- decibel function 381
- DEF model parameter 1078
- DEFAULT_INCLUDE variable 23
- DEFW option 388
- .DEL LIB statement 81
 - in .ALTER blocks 101
 - with .ALTER 105
 - with .LIB 105
 - with multiple .ALTER statements 103
- DELAY element parameter 310, 326
- delays
 - element example 322
 - group 421
 - plotting 599
 - simulation example 596, 602
 - time (TD) 421
- DELMAX option 551
- DELPHS 1079
- DELTA
 - element parameter 310, 326
- DELVTO model parameter 702
- demo examples, RF 1001
- demo files 12
 - application examples 984–??
 - behavioral applications 988
 - benchmarks 990
 - bisection-timing analysis 990
 - BJTs and diode devices 991
 - Cell Characterization Examples 991
 - circuit optimization examples 993
 - device optimization ??–993
 - encryption 995
 - FFT analysis 997
 - filter examples 995
 - IBIS examples 998
 - input 982
 - magnetics 999
 - MOSFETs 999
 - signal integrity 1002
 - sources 1003
 - S-parameters 1004
 - transmission (W-element) lines 1005
 - transmission lines 1004
 - variability 1005
 - Verilog-A 1006
- demonstration files, RF 1001
- derivative, measuring 427
- design
 - name 23
- deviation, average 698
- device characterization 116
- device warnings 1013, 1014
- DFT 549
- diagnostic tables 489–491
- digital
 - files 1063
 - vector file 358
- digital vector file
 - Waveform Characteristics section 363
- DIM2
 - parameter 423
- DIM3
 - parameter 423
- diodes
 - breakdown example 322
 - current flow 412
 - elements 441
 - equations 322
 - junction 218
 - models 218
 - polysilicon capacitor length 218
 - power dissipation 415
- directories
 - installation directory 116
 - TEMP 19, 35
 - TMP 19, 35
 - tmp 19, 35
- directory
 - structure 129
- DIS 1079
- Discrete Fourier Transform 549
- distortion 423
- distributed processing 50
- .dm# file 29
- documentation, supplemental 12
- .DOUT statement 361

Index

E

DP 50

- command-line invocation 52
- licensing 53
- limitations 57
- master 51
- network grids 50
- supported features 53
- tasks 51
- workers 51

.dp# file 25

DTEMP parameter 699, 700, 979

E

E Elements

- applications 274
- element multiplier 288
- parameters 287
- syntax statements 285
- temperature coefficients 288
- time delay keyword 288

editor, *notepad.exe* 132

element

active

MESFETs 222

IC parameter 474

identifiers 75

independent source 234, 240

L (inductor) 183

markers, mutual inductors 188

names 95

OFF parameter 471

parameters *See* element parameters 167

passive

resistors 167

R (resistor) 170

statements 89, 116

current output 410

independent sources 234

Laplace 290

op-amps 1076

pole/zero 292

temperature 700

templates 423–445

analysis 398

BJTs 442

capacitor 439

current-controlled 440

function 382

independent 441

inductor 439

JFETs 444

MOSFETs 438

mutual inductor 439

resistor 438

saturable core 445, 446

voltage-controlled 440

transmission line 197, 201

voltage-controlled 273

-element (inductor) 183

element parameter, global variation 768

element parameters

.ALTER blocks 102

BJTs 220

capacitors 174–175

DTEMP 699

F Elements 303–307

G Elements 308–313

H Elements 326–327

independent sources 234–235

data driven PWL function 255

PULSE function 246, 249, 253

SFFM function 258

inductors 182

JFETs and MESFETs 222–223

linear inductors 181, 192

MOSFETs 224–225

mutual inductors, Kxxx 188

POLY 275

PWL 253, 255

resistors 168–169

transmission lines

W-element 199

element parameters, transmission lines

U-element 202

W-element ??–198

element parameters, transmission lines, W-element
198–??

element, active

BJTs 220

JFETs 222

MOSFETs 224

element, passive

capacitors 173

inductor 181

mutual inductor 188

elements, multi-terminal 193

- .ELSEIF
 - .ELSE 106
 - encoding, 8b-10b 263
 - encryption
 - 3DES 158
 - 8-byte key 156
 - d option 148
 - data 153
 - FREELIB keyword 148
 - guidelines 149
 - i option 148
 - launching 147
 - library structure 150
 - o option 148
 - permit file changes 148
 - r option 148
 - structure 155
 - t option 148
 - traditional 152
 - triple DES public-random keys 158
 - encryption warnings 1011
 - encryption, Verilog-A 150
 - .END statement
 - for multiple HSPICE runs 106
 - in libraries 99
 - location 105
 - missing 66
 - with .ALTER 102
 - .ENDL statement 99
 - environment variables 15, 117
 - HSPWIN_KEY 19
 - META_QUEUE 17
 - TEMP 19, 35
 - TMP 19, 35
 - tmpdir 19, 35
 - equations 429, 431
 - ERR function 431
 - ERR1 function 432, 911
 - ERR2 function 432
 - ERR3 function 432
 - error messages 1016
 - errors
 - analysis 1018
 - cannot open
 - output spool file 403
 - convergence 1018, 1019
 - convergence/diode resistance 1022
 - convergence-conductance 1021
 - DC 486
 - DIAGNOSTIC option 1023
 - digital file has blank first line 1064
 - file open 35
 - functions 431–432
 - internal timestep too small 472, 493, 521
 - messages 1016
 - missing .END statement 66
 - model 1017
 - no DC path to ground 488
 - no input data 35
 - parameter name conflict 426
 - system resource inaccessible 403
 - topology 1016
 - transient analysis 1024
 - example
 - AC analysis 503
 - comment line 88
 - digital vector file 368
 - experiments 8
 - Monte Carlo 718, 726
 - optimization 914
 - transient analysis 525, 526
 - worst case 726
 - EXP source function
 - fall time 249
 - initial value 249
 - pulsed value 249
 - rise time 249
 - exp(x) function 381
 - experiment 8
 - Exploration block 885
 - exponential function 249, 381
 - expressions, algebraic 379
 - extended MOSFET element support 227
 - external data files 82
 - external sampling 873
 - external sampling, datamining 873
 - external sampling, Variation Block 806
 - extremes and corners, data mining 867
- F**
- F Elements
 - applications 274
 - multiply parameter 306
 - syntax statements 303
 - time delay keyword 307

Index

F

- value multiplier 306
- Factorial sampling 802
- fall time
 - example 602
 - EXP source function 249
 - simulation example 596
- fanout, plotting 599
- Fast Fourier Transform
 - See FFT
- FFT
 - analysis
 - AM modulation 562
 - frequency
 - of interest 558
 - range 558, 560
 - harmonic distortion 565
 - modulator/demodulator 564
 - results 558–560
 - spectral leakage 565
 - test circuit 569
 - measuring results 561
 - output 558
 - output results 561
 - windows 554–556
 - Bartlett 555
 - Blackman 556
 - Blackman-Harris 556
 - Gaussian 556
 - Hamming 555
 - Hanning 555
 - Kaiser-Bessel 556
 - rectangular 555
- FFT analysis graph data file 27
- .FFT statement 549
- file
 - analog transition data 24
 - DC operating point initial conditions 24
 - hspui.cfg* 132
 - initialization 23
 - input netlist 24
 - library input 24
 - .lis* 132
 - netlist 130, 131
 - output listing 132
 - .sp* 130, 131
- file descriptors limit 403
- files
 - .a2d* 25, 1063
 - AC analysis measurement results 26
 - AC analysis results 26
 - .ac#* 25
 - .d2a* 1063
 - DC analysis measurement results 26
 - DC analysis results 26
 - design.ac0* 595
 - design.mt0* 595
 - design.sw0* 595
 - external data 82, 98
 - FFT analysis graph data 27, 29
 - .ft#* 25, 558
 - .gr#* 25
 - hspice.ini* 117
 - .ic* 25, 471
 - include files 81
 - including 23
 - limit on number 403
 - .lis* 25
 - .ma#* 25
 - .MEASURE* output 595
 - .ms#* 25
 - .mt#* 25
 - multiple simulation runs 106
 - names 23
 - operating point node voltages 27
 - output
 - listing 27
 - status 29
 - .pa#* 25
 - scratch files 19, 35
 - .st#* 25
 - subcircuit cross-listing 29
 - .sw#* 25
 - .tr#* 25
 - transient analysis measurement results 27, 29
 - transient analysis results 30
 - transition data 1074
- files, output 25
- filters
 - active low-pass 587
 - pole/zero analysis
 - Butterworth 583
 - high-pass 583
 - low-pass 579, 587
- FIND keyword 427
- first character descriptions 74
- Foster pole-residue form

- E element 294, 343
- G element 294, 343
- Fourier
 - coefficients 551
 - equation 551
- Fourier transform
- FREELIB keyword 148
- FREQ
 - function 293
 - model parameter 1080
- frequency
 - analysis 549
 - response
 - table 292, 315
 - sweep 500
 - variable 384
 - weighing functions 554
- frequency-dependent
 - capacitor 177
 - inductor 184
- .ft# file 25, 27, 558
- full-factorial method 802
- functions
 - A2D 1063, 1074
 - bisection 935
 - built-in 380–384
 - D2A 1063, 1074
 - DERIVATIVE 431
 - ERR 431
 - INTEG 431
 - LAPLACE 290, 315, 575
 - NPWL 318
 - POLE 291, 315, 575
 - PPWL 319
 - table 380
- See also* independent sources

G

- G Elements
 - applications 274
 - controlling voltages 311, 313
 - current 311
 - curve smoothing 312
 - element value multiplier 312
 - gate type 310
 - initial conditions 311
 - multiply parameter 311
 - names 310

- polynomial 312
- resistance 311
- syntax statements 308
- time delay keyword 312
- transconductance 312
- voltage to resistance factor 312
- GaAsFET model DC optimization 929
- GAUSS
 - FFT analysis 556, 574
 - functions 719
 - keyword 713
 - parameter distribution 709
- GBW model parameter 1080
- global parameters 386
- global variation element parameters 768
- GMIN option 495
- GMINDC option 482, 495
- GND node 96
- GOAL keyword 912
- .gr# file 25
- GRAMP
 - option 486
- graphical user interface 129–??
- ground, node name 96
- group operator, variation block 772
- GUI
 - using 129–??
- Gxxx element parameters 310

H

- H Elements
 - applications 274
 - controlling voltage 328
 - data points 327
 - element multiplier 327
 - element name 326
 - gate type 326
 - initial conditions 326
 - maximum current 327
 - minimum current 327
 - syntax statements 324
 - time delay keyword 327
 - transresistance 327
- H parameters 1073
- Hamming FFT analysis window 555, 572
- Hanning FFT analysis window 555, 572
- HCI and NBTI analysis 613

Index

I

- HD2 distortion 423
 - HD3 distortion 423
 - hertz variable 384
 - hierarchical designs, flattened 82
 - hold time 938
 - HSPICE
 - input netlist 130, 131
 - installation directory 116
 - starting 34
 - training 12
 - hspice command 34, 49
 - HSPICE website 12
 - hspice.ini 23
 - hspice.ini file 117
 - hspicrf command 39
 - HSPUI
 - text editor 142
 - hspui.cfg* 132
 - HSPWIN_KEY environment variable 19
 - hybrid (H) parameters 418
- I
- IB model parameter 1080
 - IBIS buffers 231
 - IBOS model parameter 1080
 - .ic file 25, 471
 - IC parameter 311, 326, 474
 - .IC statement 470, 471, 577
 - balancing input nodes 1076
 - IC, three-dimensional simulation 671
 - .ic# file 27
 - ideal
 - current sources 486
 - delay elements 274
 - op-amp 274, 296, 298
 - transformer 274, 296, 300
 - ideal transformer 192
 - IDELAY statement 364
 - .IF 106
 - .IFELSE 106
 - impedance
 - AC 422
 - Z parameters 418
 - include files 23
 - .INCLUDE statement 81, 102, 117, 119
 - independent sources
 - AC 235, 239
 - AM function 260
 - current 235, 441
 - data driven PWL function 255
 - DC 239
 - elements 234
 - EXP function 249
 - functions 240
 - mixed types 240
 - PULSE function 241
 - PWL function 252
 - SFFM function 258
 - SIN function 245
 - transient 235, 239
 - types 240
 - voltage 235, 441
 - See also* sources
 - individual element temperature 700
 - inductor
 - frequency-dependent 184
 - inductors
 - AC choke 185
 - current flow 412
 - element 181, 439
 - node names 182, 192
 - initial conditions 470
 - file 24
 - statement 475
 - initialization 470, 471
 - saved operating point 476
 - initialization file 23
 - INOISE parameter 423
 - input
 - admittance 422
 - analog transition data file 24
 - data
 - adding library data 105
 - for data driven analysis 98
 - DC operating point initial conditions file 24
 - files
 - analog transition data 24
 - character case 68
 - DC operating point 24
 - demonstration 982
 - library 24
 - names 23
 - netlist 24, 65
 - structure 81
 - table of components 82

- impedance 422
- library file 24
- netlist 84
- netlist file 24, 84–105, 1045
- input netlist file 24
- input stimuli 437
- input syntax
 - Monte Carlo 790
- input/output
 - cell modeling 980
- installation directory \$installdir 116
- int(x) function 381
- integer function 381
- integration
 - algorithms 533
- interactive mode
 - quitting 40
 - running command files 40
- internal
 - nodes, referencing 96
- inverter
 - analysis, transient 526
 - circuit, MOS 526
- invoking
 - hspice 34, 49
 - hspicerf 39
- ISC model parameter 1080
- isomorphic analyses 36
- iterations
 - number 920
- I-V and C-V plotting demo 968

J

- JFETs
 - current flow 412
 - elements 222, 444
 - length 222
 - power dissipation 416
 - width 222
- JIS model parameter 1080
- jitter
 - random, with clock source 355
- jitter, random, clock source 355

K

- Kaiser-Bessel FFT analysis window 556, 558, 574
- Kerwin's circuit, pole/zero analysis 582

- keywords
 - analysis statement syntax 913
 - DTEMP 699
 - ERR1 911
 - FREELIB 148
 - GOAL 912
 - LAST 428
 - MONTE 709
 - optimization syntax 913
 - PAR 374, 379
 - power output 414
 - PP 430
 - source functions 234
- k-string 263

L

- LA_FREQ option 645
- LA_MAXR option 645
- LA_MINC option 645
- LA_TIME option 646
- LA_TOL option 646
- Laplace
 - function 290, 315, 575
 - transform 290, 315
 - frequency 292, 315
- LAST keyword 428
- Latin Hypercube sampling 803, 811
- launcher
 - MS Windows 129
- LDS sampling 805, 812
- leadframe example 973
- LENGTH model parameter 720
- Levenberg-Marquardt algorithm 920
- LEVIN model parameter 1080
- LEVOUT model parameter 1081
- LHS 803, 811
- .LIB
 - call statement 99
 - statement 81, 119
 - in .ALTER blocks 99, 101
 - with .DEL LIB 105
 - with multiple .ALTER statements 103
- .LIB file encryption 154
- libraries
 - adding with .LIB 105
 - ASIC cells 117
 - building 99

Index

M

- configuring 388
 - creating parameters 386
 - DDL 116
 - duplicated parameter names 386
 - encryption 146
 - .END statement 99
 - integrity 385
 - protecting 146
 - search 117
 - selecting 100
 - subcircuits 118
 - vendor 117
 - library input file 24
 - limit descriptors command 403
 - LIMIT keyword 713
 - line continuation
 - VEC files 366
 - linear
 - acceleration 642
 - capacitor
 - element
 - C (capacitor) 176
 - matrix reduction 642
 - linear elements
 - elements, linear 193
 - linear inductor 183
 - linear resistor 170
 - .lis file 25, 27
 - .lis file 132
 - listing file 132
 - listing file, text editor for Windows 142
 - LMAX model parameter 6
 - LMIN model parameter 6
 - .LOAD statement 476
 - local
 - parameters 386
 - log(x) function 381
 - log10(x) function 381
 - logarithm function 381
 - Low Discrepancy Sequence sampling, LDS 805
 - Low-Discrepancy Sequence sampling, LDS 812
 - Lsim models, calibrating 595
 - LV 424
 - LV18 model parameter 969
 - LX 424
 - LX7 model parameter 969
 - LX8 model parameter 969
 - LX9 model parameter 969
- ### M
- M element parameter 306, 311
 - .ma# file 25, 26
 - machine cluster 50
 - MACMOD option 227
 - MACMOD option limitations 229
 - macros 105
 - magnitude, AC voltage 418
 - MANU model parameter 1081
 - manufacturing tolerances 719
 - Marquardt scaling parameter 920
 - MAX parameter 311, 327, 938
 - max(x,y) function 382
 - maximum value, measuring 429
 - mean, statistical 698
 - measure data, pass/fail 433
 - .MEASURE statement 397, 427
 - expression 429
 - failure message 426
 - parameters 378
 - measure statements, order 425
 - measurement warnings 1015
 - measurements 561
 - measurements, complex, statement order 425
 - measuring parameter types 426
 - menu configuration, MS Windows launcher 131
 - MESFETs 222
 - messages, warnings 1010
 - META_QUEUE environment variable 17
 - Metaencrypt 147
 - metaencrypt
 - launching 147
 - options when invoking 148
 - MIN parameter 311, 327
 - min(x,y) function 381
 - minimum
 - value, measuring 429
 - mismatch 836
 - mixed mode
 - See also D2A, A2D
 - mixed sources 240
 - model binning warnings 1012
 - model error messages 1017

- MODEL keyword 913
 - model parameters
 - A2D 1063
 - .ALTER blocks 102
 - capacitance distribution 721
 - D2A 1063, 1065
 - DELVTO 702
 - DTEMP 700
 - LENGTH 720
 - manufacturing tolerances 719
 - op-amps 1077, 1083
 - PHOTO 720
 - RSH 702
 - sigma deviations, worst case analysis 702
 - skew 701
 - TEMP 98, 700
 - temperature analysis 700
 - TOX 702
 - TREF 698, 700
 - XPHOTO 720
 - model parameters *See* model parameters diodes
 - .MODEL statement 700
 - op-amp 1076
 - model warnings 1011
 - models
 - characterization 477
 - DTEMP parameter 979
 - LV18 969
 - LX7, LX8, LX9 969
 - Monte Carlo analysis 709, 716, 726
 - op-amps 1075
 - reference temperature 700
 - specifying 117
 - typical set 705
 - Monte Carlo
 - analysis 697, 698, 726–734, 785
 - distribution options 713–714
 - application considerations 830
 - factorial sampling 802
 - input syntax 790
 - simulation output 794
 - variation block options 791
 - Monte Carlo analysis
 - operating-point results in transient analysis 728
 - transient sigma sweep results 728
 - MONTE keyword 709
 - MOS
 - inverter circuit 526
 - op-amp optimization 930
 - MOSFET
 - extended element support 227
 - MOSFETs
 - current flow 413
 - drain diffusion area 224
 - elements 224, 438
 - initial conditions 225
 - perimeter 224
 - power dissipation 417
 - source 224, 225
 - temperature differential 225
 - zero-bias voltage threshold shift 225
 - .MOSRA command 610
 - MOSRALIFE option 623
 - .MOSRAPRINT command 618
 - MOSRASORT option 624
 - mpp0 file 861
 - confidence interval levels 862
 - extremes and corners 867
 - other_percentiles 865
 - response correlation matrix 870
 - sample quartiles 864
 - variable screening 868
 - MS Windows launcher 129
 - batch job list 136
 - multi jobs 135
 - .ms# file 25, 26
 - .mt# file 25, 27, 29
 - Muller algorithm 576, 577
 - multi-die interconnect analysis 671
 - multiple .ALTER statements 102
 - multiply parameter 110, 169, 235
 - multipoint experiment 8
 - multi-terminal linear elements 193
 - multi-terminal network, S-parameter 203
 - multithreading 59
 - mutual inductor 188, 439
- ## N
- NAND gate adder 968
 - natural
 - log function 381
 - natural frequency 576
 - NBTI and HCI analysis 613
 - NDIM 275

Index

O

- negative conductance 1011
 - .NET parameter analysis 1071
 - netlist 82
 - file example 85
 - flat 82
 - input files 65
 - schematic 82
 - structure 85
 - netlist encryption 146
 - netlist file
 - example 85
 - netlists
 - compression 83
 - .gz format 83
 - network grid, DP 50
 - network output 422
 - Niederreiter sampling 805, 812
 - nodal voltage output 409, 420
 - node voltages, encrypting 146
 - nodes
 - connection requirements 96
 - floating supply 96
 - internal 96
 - MOSFET's substrate 96
 - names 91, 95, 97, 969
 - automatic generation 97
 - ground node 96
 - period in 92
 - subcircuits 95, 96
 - zeros in 97
 - numbers 91, 95
 - shorted 487
 - terminators 96
 - .NODESET statement 470, 577
 - balancing input nodes 1076
 - noise
 - calculations 506
 - input 423
 - output 423, 506
 - norm of the gradient 919
 - notepad.exe* 132
 - NPDELAY element parameter 327
 - NPWL function 318
- O**
- ODELAY statement 364
 - OFAT 801
 - OFF parameter 471
 - on-chip signaling 671
 - one-dimensional function 275
 - One-Factor-at-a-Time sampling 801
 - ONNOISE parameter 423
 - onscreen messages, time progress 407
 - .OP statement 472, 473
 - op-amp analysis 601
 - op-amps
 - automatic generation 1074
 - characterization 601
 - common mode rejection ratio 1077
 - compensation level selector 1078
 - diode and BJT saturation current 1079
 - element statement 1076
 - excess phase parameter 1079
 - gain parameter 1077
 - input
 - bias current 1080
 - bias offset current 1080
 - level type selector 1080
 - offset current 1080
 - offset voltage 1082
 - short circuit current 1080
 - internal capacitance 1078
 - JFETs saturation current 1080
 - manufacturer's name 1081
 - model
 - generator 1075
 - parameters 1077, 1083
 - selector 1078
 - .MODEL statement 1076
 - open loops 487
 - optimization 930
 - output
 - level type selector 1081
 - resistance ROAC 1081
 - slew rate 1081
 - voltage 1082
 - phase margin 1079
 - power
 - dissipation 1081
 - supply voltage 1082
 - subcircuit generator 1074
 - temperature parameter 1082
 - unity gain frequency 1080
 - operating point
 - estimate 472

- initial conditions 24
- pole/zero analysis 577
- saving 97
- solution 470, 471
- operating point information file 27
- operating point node voltages file 27
- operators 380
- OPT keyword 912
- optimization
 - AC analysis 926
 - analysis statements 913
 - bisection method 938
 - CMOS tristate buffer 922
 - control 910
 - convergence options 910
 - curve-fit 911
 - data-driven vs. s-parameters 926
 - DC analysis 915, 917, 927, 929
 - example 914, 981
 - goal 911
 - incremental 928
 - lengths and widths 931
 - MODEL keyword 913
 - MOS 917, 930
 - network 918, 926
 - parameters 926
 - magnitude and phase 926
 - measured vs. calculated 926
 - results
 - function evaluations 920
 - iterations 920
 - Marquadt scaling parameter 920
 - norm of the gradient 919
 - residual sum of squares 919
 - simulation accuracy 910
 - simultaneous 923, 929, 931
 - S-parameters 925
 - statements 912
 - syntax 912
 - time
 - analysis 912
 - required 910
- OPTIMIZE keyword 913
- .OPTION
 - .ALTER blocks 102
 - DCSTEP 488
 - INGOLD, for exponential output 403
 - LA_FREQ 645
 - LA_MAXR 645
 - LA_MINC 645
 - LA_TIME 646
 - LA_TOL 646
 - POST, to display waveform plots 403
 - SIM_LA 642, 645
 - .OPTION MACMOD 227
 - .OPTION SEARCH
 - implicit include command 118
- options
 - MOSRALIFE 623
 - MOSRASORT 624
- OPTxxx parameter 911, 912
- other_percentiles, data mining 865
- output
 - AC analysis measurement results file 26
 - AC analysis results file 26
 - admittance 422
 - commands 396
 - current 410
 - DC analysis measurement results file 26
 - DC analysis results file 26
 - .DCMATCH output tables file 29
 - driver example 973
 - FFT analysis graph data file 27
 - .FFT results 558
 - files
 - AC analysis measurement results 26
 - AC analysis results 26
 - DC analysis measurement results 26
 - DC analysis results 26
 - .DCMATCH output tables file 29
 - FFT
 - analysis graph data 27
 - names 23
 - operating point information 27
 - operating point node voltages 27
 - output listing 27
 - output status 29
 - subcircuit cross-listing 29
 - transient analysis measurement results 29
 - transient analysis results 29
 - impedance 422
 - network 422
 - nodal voltage, AC 420
 - noise 423, 506
 - operating point information file 27
 - operating point node voltages file 27

Index

P

- output listing file 27
- output status file 29
- parameters 407
- power 413
- printing 404–405
- reusing 437
- saving 399
- statements 396
- subcircuit cross-listing file 29
- transient analysis measurement results file 29
- transient analysis results file 29
- variables 397
 - AC formats 420
 - function 383
 - voltage 409
- output files 25
- output listing file 27, 132
- output status file 29
- overview of simulation process 9

P

- .pa# file 25, 29
- PAR keyword 374, 379, 606
- .PARAM statement 100, 426, 697
 - in .ALTER blocks 101
- parameter analysis, .NET 1071
- parameter expression warnings 1012
- parameters
 - admittance (Y) 418
 - algebraic 379, 380
 - analysis 378
 - assignment 373
 - cell geometry 385
 - constants 374
 - data type 373
 - data-driven analysis 98
 - defaults 391
 - defining 371, 386
 - DIM2 423
 - DIM3 423
 - encrypting 146
 - evaluation order 373
 - HD2 423
 - HD3 423
 - hierarchical 109, 385, 427
 - hybrid (H) 418
 - impedance (Z) 418

- inheritance 388, 391
- INOISE 423
- input netlist file 78
- libraries 386–388
- M 110
- measurement 378
- model 1065, 1067
- modifying 98
- multiply 379
- ONOISE 423
- optimization 385
- OPTxxx 911, 912
- output 407
- overriding 386, 391
- PARHIER option 391
- passing 385–394
 - example 606
 - order 373
 - problems 394
 - Release 95.1 and earlier 394
- repeated 426
- scattering (S) 418
- scope 385–386, 394
- SIM2 423
- simple 374
- string 390
- subcircuit 109
- user-defined 376
- parametric analysis 398
- PARHIER option 391
- pass/fail, measure output 433
- passfail 941
- passive component model name keywords, string
 - parameters 390
- path names 96
- PD model parameter 1081
- peak-to-peak value, measuring 429
- permit.hsp file, encryption capability 148
- PHOTO model parameter 720
- PI (linear acceleration) algorithm 644
- piecewise linear sources *See* PWL
- pin capacitance, plotting
 - plotting
 - pin capacitance 600
- .PLOT statement
 - simulation results 404
- plotting
 - delay vs. fanout 599

- op-amp characterization 601
 - pn junction conductance 495
 - POLE
 - function 291, 315, 575
 - transconductance element statement 292
 - voltage gain element statement 291
 - pole/zero
 - analysis 575
 - active low-pass filter 587
 - CMOS differential amplifier 584
 - high-pass Butterworth filter 583
 - Kerwin's circuit 582
 - Muller algorithm 576
 - operating point 577
 - overview 575
 - simple amplifier 586
 - conjugate pairs 292
 - function, Laplace transform 291, 315
 - POLY parameter 275, 312, 327
 - polynomial function 275
 - one-dimensional 275
 - three-dimensional 277
 - two-dimensional 276
 - post-layout back-annotation 650
 - pow(x,y) function 380
 - power
 - dissipation 417
 - function 381
 - output 413
 - stored 414
 - POWER keyword 414
 - power keyword
 - independent sources 280
 - using 281
 - power source
 - independent 280
 - PP keyword 430
 - PPWL
 - element parameter 312
 - function 319
 - print
 - control options 403
 - .PRINT statement 397, 969
 - simulation results 399
 - .PROBE statement 397, 399, 404
 - processing, distributed 50
 - program structure 7
 - progress messages 407
 - PSRR specification 1074
 - PULSE source function 246, 249, 253
 - delay time 242
 - initial value 242
 - onset ramp duration 242
 - plateau value 242
 - recovery ramp duration 242
 - repetition period 242
 - width 242
 - pulse width 946
 - pushout bisection methodology 947
 - PUTMEAS option 426
 - PWL
 - current controlled gates 274
 - data driven 255
 - element parameter 306, 327
 - functions 274, 279
 - gates 274
 - output values 253
 - parameters 252
 - repeat parameter 253
 - segment time values 253
 - sources, data driven 255
 - voltage-controlled capacitors 274
 - voltage-controlled gates 274
 - pwl sources 1074
 - PWL
 - See also* data driven PWL source
 - PWR model parameter 1081
 - pwr(x,y) function 381
 - .PZ statement 478
- ## Q
- quality assurance 697
- ## R
- R Element (resistor) 170
 - RAC model parameter 1081
 - RC
 - analysis 503, 525
 - circuit 504
 - optimizing 918
 - rcells, reusing 386
 - rectangular FFT window 555
 - reference temperature 98, 700
 - RELI option 481

Index

S

- RELMOS option 481
 - reluctors 186
 - RELV option 481
 - repeat function 967
 - residual sum of squares 919
 - resistance 502
 - resistor
 - current flow 412
 - element 168
 - element template listings 438
 - length parameter 169
 - linear 170
 - model name 168
 - node to bulk capacitance 169
 - voltage controlled 317
 - width parameter 169
 - response correlation matrix, data mining 870
 - results, waveform viewer 1045
 - reusing simulation output 437
 - RF
 - demo files 1001
 - RF demo examples 1001
 - rise time 596, 602
 - RLOAD model parameter 1067
 - rms value, measuring 429
 - roac model parameter 1081
 - ROUT model parameter 1081
 - RSH model parameter 702
 - RUNLVL algorithm 530
- S**
- S19NAME model parameter 1068
 - S19VHI model parameter 1068
 - S19VLO model parameter 1068
 - S1NAME model parameter 1067
 - S1VHI model parameter 1068
 - S1VLO model parameter 1068
 - sample moments, data mining 861
 - sample quartiles, data mining 864
 - sampling
 - External 806
 - external 873
 - factorial, full factorial 802
 - LDS, Niederreiter, Sobol sampling 805, 812
 - LHS, Latin Hypercube sampling 803, 811
 - Niederreiter 805, 812
 - OFAT, One-Factor-at-a-Time sampling 801
 - Sobol 805, 812
 - SRS, simple random sampling 810
 - saturable core
 - elements 189, 445
 - models 189
 - winding names 446
 - .SAVE statement 476
 - scale factors 77
 - SCALE parameter 169, 288, 306, 312, 327, 969
 - scaling, effect on delays 980
 - scattering (S) parameters 203, 418
 - scattering parameter element 203
 - schematic
 - netlists 82
 - scope of parameters 386
 - scratch files 19, 35
 - scripting resources 12
 - SEARCH option 118
 - search path, setting 100
 - S-element 203
 - syntax 204
 - .SENS statement 478
 - setup time 938, 940, 942
 - SFFM source function
 - carrier frequency 258
 - modulation index 258
 - output amplitude 258
 - output offset 258
 - signal frequency 258
 - sgn(x) function 381
 - shorted nodes 487
 - sign function 381
 - SIGNAME element parameter 1067
 - signed power function 381
 - silicon-on-sapphire devices 97
 - SIM_LA option 642, 645
 - SIM2 distortion measure 423
 - simple random sampling 810
 - simulate button 130
 - simulation
 - accuracy 910
 - tolerances 479, 480, 481
 - example 1045
 - multiple runs 106
 - process, overview 9
 - results

- printing 404–405
 - specifying 426–427
- reusing output 437
- structure 7
- title 87
- simulation output
 - Monte Carlo 794
- SIN source function 246
- sin(x) function 380
- single point experiment 8
- single-frequency FM source function 258
- sinh(x) function 380
- sinusoidal source function 245
- skew
 - file 705
 - parameters 701
- SMOOTH element parameter 312
- SNUG papers, HSPICE 12
- Sobol sampling 805, 812
- SolvNet articles 12
- SONAME model parameter 1067
- source
 - data driven 255
 - keywords 234
 - statements 89
 - See also* independent sources
- SOVHI model parameter 1067
- SOVLO model parameter 1067
- .sp file 130, 131
- .sp file encryption 153
- S-parameter 203
 - S-element 203
- spectral leakage 554, 565
- spectrum analysis 549
- SPICE Explorer 1045
- sqrt(x) function 380
- square root function 380
- SRN model parameter 1081
- SRNEG model parameter 1081
- SRP model parameter 1082
- SRPOS model parameter 1082
- SRS 810
- .st# file 25, 29
- Star-RC, Variation block 775
- starting
 - hspice 34, 49
 - hspicerf 39
- statement
 - .DOUT 361
- statements
 - .AC 700
 - .DATA 98, 603
 - .DC 476, 700, 913
 - DOUT 397
 - element 89
 - .ENDL 99
 - .GRAPH 404
 - initial conditions 475
 - .LIB 99
 - .LOAD 476
 - .MEASURE 397, 424
 - .MODEL 700
 - .MOSRA 610
 - .MOSRAPRINT 618
 - .OP 473
 - .OPTION
 - CO 403
 - .PARAM 100
 - .PLOT 404
 - .PRINT 397, 399, 404
 - .PROBE 397, 399, 404
 - .SAVE 476
 - source 89
 - .STIM 397, 437
 - .SUBCKT 426, 1075
 - .TEMP 98, 700
 - .TRAN 700
- statistical analysis 701–734
- statistical sensitivity coefficients 775
- statistics
 - calculations 698
- .STIM statement 397, 437
- stimuli 437
- stimulus input files 1074
- string parameters 390
- structure simulation 7
- subcircuit cross-listing file 29
- subcircuits
 - adder 967
 - calling tree 96
 - changing in .ALTER blocks 101
 - creating reusable circuits 108
 - generator 1074
 - hierarchical parameters 109

Index

T

- library structure 118
 - multiplying 110
 - node names 95, 97
 - output printing 404
 - parameter 606
 - path names 96
 - .PRINT statements 112
 - search order 113
 - zero prefix 97
 - subckt
 - val() function 112
 - .SUBCKT statement 426, 1075
 - sub-xpressions, variation block 772
 - summary statistics, data mining 860
 - supplemental documentation 12
 - .sw# file 25, 26
 - sweep
 - frequency 500
 - variables 979
 - switch example 320
 - switch-level MOSFET's example 320
 - sx 1045
 - Synopsys models, calibrating 595
 - Synopsys User Group tutorials 12
 - syntax, S-element 204
- ### T
- tabular data 359
 - Taguchi analysis 697
 - tan(x) function 380
 - tanh(x) function 380
 - tasks, distributed processing 51
 - TC1, TC2 element parameters 288
 - TD parameter 288, 307, 312, 327, 421, 428
 - TDELAY statement 364
 - TEMP
 - directory 19, 35
 - environment variable 19, 35
 - model parameter 98, 700, 1082
 - sweep variable 979
 - .TEMP statement 700
 - temper variable 384
 - temperature
 - circuit 698, 700
 - coefficients 169, 978
 - derating 98, 700
 - element 700
 - optimizing coefficients 978
 - reference 98, 700
 - sweeping 979
 - variable 384
 - Temperature Variation Analysis 697
 - termination criteria, convergence 1018
 - .TF statement 478
 - three-dimensional function 277
 - three-dimensional IC
 - construct usage 673
 - illegal netlist syntax 674
 - intrinsic model card, top-level IC instance 681
 - module usage 672
 - module, instance-specific properties scope 681
 - netlist definitions 672
 - netlist rules 671
 - parameter direct access, top-level netlist 685
 - scope reference rules 675
 - sweeps, temperature and parameters 686
 - top-level IC module reference 680
 - .TRAN and .ALTER simulations 686
 - Verilog-A reference rules 678
 - Verilog-A usage 677
 - time
 - delay 421
 - variable 384
 - TIMESCALE model parameter 1068
 - TIMESTEP model parameter 1068
 - timing
 - analysis 935
 - constraints 935
 - failures 936
 - hold time 938
 - setup time 938
 - violation analysis 936
 - title for simulation 87
 - .TITLE statement 87
 - TMI flow 5
 - TMP directory 19, 35
 - tmp directory 19, 35
 - TMP environment variable 19, 35
 - tmpdir environment variable 19, 35
 - TNOM option 98, 700
 - topology error messages 1016
 - topology integrity warnings 1010
 - TOX model parameter 702

.tr# file 25, 30
 traditional library encryption 152
 .TRAN statement 700, 913
 transconductance
 FREQ function 293
 LAPLACE function 290
 POLE function 292
 transfer function
 frequency domain 575
 poles 576
 roots 576
 zeros 576
 transfer sign function 381
 transformer, ideal 192
 transient
 analysis 397
 initial conditions 521
 inverter 526
 RC network 525
 sources 239
 output variables 407
 transient analysis
 bisection 942
 transient analysis error messages/solutions 1024
 transient analysis measurement results file 29
 transient analysis results file 29
 transient analysis warnings 1014
 transition files 1074
 transmission lines
 example 973
 U-element 201
 TREF model parameter 700
 triode tube 323
 TSMC models 5
 tutorials 12
 two-dimensional function 276

U

U Elements 1063
 digital input 1063
 UIC
 analysis parameter 472
 UNIF keyword 713
 uniform parameter distribution 709
 update training 12

V

valp() function 112
 variability
 defined in HSPICE 747
 introduction 745
 simulating 745
 variation block 748
 variable screening, data mining 868
 variables
 AC formats 420
 changing in .ALTER blocks 101
 DEFAULT_INCLUDE 23
 Hspice-specific 384
 output 397
 AC 418
 DC 407
 transient 407
 plotting 969
 sweeping 979
 TEMP 19, 35
 TMP 19, 35
 tmpdir 19, 35
 variables, environment 15
 variance, statistical 698
 Variation Block
 element parameters, global variation 768
 variation block
 absolute vs relative variation 758
 access functions 763, 765
 advantages 748
 dependent random variables 756
 element parameter variations 765
 example 770
 general section 751
 options 751
 global subblocks 752
 group operator 772
 independent random variables 753
 local subblocks 752
 model parameter variations 758
 overview 749
 structure 751
 subexpressions 772
 variation block options
 Monte Carlo 791
 VCC model parameter 1082
 VCCAP 318
 VCCS *See* voltage controlled current source

Index

W

VCR *See* voltage controlled resistor
VCVS *See* voltage controlled voltage source
vector patterns 359
vector-modulated RF 344
vector-modulated RF
 E element 351
 F element 351
 G element 351
 H element 351
 I element 346
 implementation 345
 V element 346
VEE model parameter 1082
vendor libraries 117
Verilog models, calibrating 595
Verilog value format 362
Verilog-A
 encryption 150
VHDL models, calibrating 595
VIH statement 364
VIL statement 364
VMRF, *See* vector-modulated RF 344
Vnn node name in CSOS 97
VOH statement 365
VOL keyword 300
voltage
 failure 490
 gain
 FREQ function 292
 LAPLACE function 290
 POLE function 291
 logic high 364, 365
 logic low 364
 nodal output DC 409
 sources 285, 324, 409
 summer 298
voltage-controlled
 capacitor 318, 321
 current source 274, 304, 308, 309, 313, 440
 oscillator 300
 resistor 274, 313, 317
 voltage source 274, 286, 440
VON model parameter 1082
VONEG model parameter 1082
VOP model parameter 1082
VOPOS model parameter 1082
VOS model parameter 1082

Vos specification 1074
VREF statement 364
VTH statement 365
Vxxx source element statement 234

W

warning messages 1010
warnings
 all nodes connected together 487
 auto-convergence 1015
 bisection analysis 1014
 control options 1012
 DC/OP analyses 1015
 device 1013, 1014
 encryption 1011
 floating power supply nodes 96
 measures 1015
 model 1011
 model binning 1012
 parameter expression 1012
 topology integrity 1010
 transient analysis 1014
 zero diagonal value detected 489
waveform
 characteristics 363, 364
Waveform Characteristics section 363
waveform viewer 1045
WaveView 133, 1045
webinar sign-up 12
website, HSPICE.com 12
W-elements 197
WHEN keyword 427
white papers 12
wildcard uses 93
WMAX model parameter 6
WMIN model parameter 6
worst case analysis 602, 701, 726
Worst Case Corners Analysis 697
worst-case analysis 734
worst-case, factorial sampling, Monte Carlo 802

X

XL model parameter 702
XPHOTO model parameter 720
XW model parameter 702

Y

yield analysis 697
YIN keyword 422
YOUT keyword 422

Z

zero delay gate 299, 321
ZIN keyword 422
ZOUT keyword 422

Index
Z