HSPICE® User Guide: Advanced Analog Simulation and Analysis

Version K-2015.06, June 2015

SYNOPSYS®

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

Contents

Conventions xiii

Customer Support xiv

1. HSPICE Advanced Analog Features 1

Introduction to HSPICE Advanced Analog Analyses 1

- HSPICE Advanced Analog Analyses Features 2
- Using HSPICE Advanced Analog Analyses 5
- HSPICE Advanced Analog Output Files 5
- Using Custom WaveView 7
- Creating a Configuration File 8
  - Inserting Comments in a .hspicerf File 10
- Using Wildcards 10
- Limiting Output Data Size 11
- Generating Measurement Output Files 12
- Optimization 12
- Optimizing with HB Measurements 14
- Optimizing with HBNOISE or PHASENOISE Measurements 14
- Using CHECK Statements 14
- POWER DC Analysis 18
  - Power DC Analysis Output Format 19
  - POWER Analysis 20
- Detecting and Reporting Surge Currents 21
- Advanced Analog Demonstration Input Files 22

Use of Example Syntax 23

Part I: Time Domain, Steady-State Analysis

2. Steady-State Harmonic Balance Analysis 27

Harmonic Balance Analysis 27

- Features Supported 28
- Harmonic Balance Equations 29

Harmonic Balance Output Measurements 30
HB .PRINT and .PROBE Output Syntax 31
HB Output Data Files and Examples 33
HB Error and Warning Messages 34
Calculating Power Measurements After HB Analyses 36
Calculations for Time-Domain Output 39
Using .MEASURE with .HB Analyses 40

Steady-State HB Sources 43

Phase Differences Between HB and SIN Sources 44

Tutorial Examples Harmonic Balance Analysis 45
Example 1: Using HB Analysis for a Power Amplifier. 45
Example 2: Using HB Analysis for a Low Noise Amp 48

References. 60

3. Steady-State Shooting Newton Analysis 61

Shooting Newton Steady-State Time Domain Analysis (.SN) 61
SN .PRINT and .PROBE Output Syntax 62

Shooting Newton with Fourier Transform (.SNFT) 64
Other Shooting Newton Analyses 65
SN Error and Warning Messages 66

Shooting Newton Analysis — Tutorial Example. 66
Shooting Newton — Analysis Setup 67
Driven Phase Frequency Detector Example 68
Ring Oscillator Example. 73

Part II: Oscillator and PLL Analysis

4. Oscillator and Phase Noise Analysis. 81

Harmonic Balance Oscillator Analysis (.HBOSC) 82
HB Simulation of Ring Oscillators 83
.HBOSC Output Syntax 84
Using the .MEASURE Command with .HBOSC 84
Troubleshooting Convergence Problems 84
General Convergence Issues 85
Outer Loop Convergence 88

Inner Loop Convergence 89
Tutorial Examples Using HBOSC Analysis 90
Example 1— Colpitts Oscillator 90
Example 2 — Using HBOSC for a CMOS GPS VCO. 93

Shooting Newton Oscillator Analysis(.SNOSC). 100

Phase Noise Analysis (.PHASENOISE) 101
Identifying Phase Noise Spurious Signals 104
Phase Noise Algorithms. 105
PHASENOISE Output Syntax 106
Measuring Phase Noise with .MEASURE PHASENOISE 109
Amplitude Modulation/Phase Modulation Separation 111
Interpreting Phase Noise Analysis Results 113
Important Note for AM/PM Users 114

Accumulated Jitter Measurement for Closed Loop PLL Analysis 115
Jitter Measurements from Phase Noise. 116

Clock Source with Random Jitter. 122

Small-Signal Phase-Domain Noise Analysis (.ACPHASENOISE). 122
ACPHASENOISE Analysis .PRINT and .PROBE Syntax 123

Behavioral Noise Sources 124
Using Noise Analysis Results as Input Noise Sources 126
Power Supply Current and Voltage Noise Sources 127

References. 129

5. Large Signal Periodic AC, Transfer Function, and Noise Analyses 131

Multitone Harmonic Balance AC Analysis (.HBAC) 131
Prerequisites and Limitations. 132
Input Syntax. 133
HBAC Analysis Options 133
Output Syntax 133
HBAC Output Data Files 135
Using the .MEASURE Command with .HBAC 135
Errors and Warnings 136
Error Messages 136
Warning Messages. 136
Tutorial Example - Using Multi-Tone HB and HBAC Analyses for a Mixer 136
Two-tone HB Approach 137
HBAC Approach. 138
Comparing Results. 139

Shooting Newton AC Analysis (.SNAC) 141
Prerequisites and Limitations 141
Input Syntax 142
Output Syntax 142
SNAC Output Data Files 144
Using the .MEASURE Command with .SNAC 144
Errors and Warnings 145
Error Messages 145
Warning Messages 145
SNAC Example 145

Steady-State Voltage and Current Sources 146
I and V Element Syntax 146

Phase Noise and Buffer Chains 150

Multitone Harmonic Balance Noise (.HBNOISE) 151
Supported Features 152
Prerequisites and Limitations 152
Input Syntax 152
Output Syntax 154
.PRINT and .PROBE Statements 154
Output Data Files 155
Measuring HBNOISE Analyses with .MEASURE 156
Errors and Warnings 158
HBNOISE Example 158

Shooting Newton Noise Analysis (.SNNOISE) 159
Supported Features 159
Prerequisites and Limitations 160
Input Syntax 160
Specifying Variant Indices 162
Output Syntax 162
.PRINT and .PROBE Statements 162
Output Data Files 163
Measuring SNNOISE Analyses with .MEASURE 163
SNNOISE Analysis Example 165

Periodic Time-Dependent Noise Analysis (.PTDNOISE) 166
PTDNOISE Input Syntax 168
PTDNOISE Output Syntax and File Format 169
Output File Format 170
.MEASURE Syntax and File Format 170
Error Handling and Warnings 171

Usage Example 172

Troubleshooting .PTDNOISE Simulation 173

Multitone Harmonic Balance Transfer Function Analysis (.HBXF). 173
Supported Features 174
Prerequisites and Limitations 174
Input Syntax 175
Output Syntax 175
.PRINT and .PROBE Statements. 175
Output Data Files 176
Using the .MEASURE Command with .HBXF 176
Example 176
HBXF Test Listing. 177

Shooting Newton Transfer Function Analysis (.SNXF). 177
Input Syntax 178
Output Syntax 178
Output Data Files 179
Using the .MEASURE Command with .SNXF 179
Example 179
SNXF Test Listing. 180

References. 181

6. S-parameter Analyses 183

Frequency Translation S-Parameter (HBLIN) Extraction 184
HB Analysis 186
Port Element 187
HBLIN Analysis 187
Input Syntax 187
Output Syntax 189
.PRINT and .PROBE Statements. 189
Output Data Files 190

Large-Signal S-parameter (HBLSP) Analysis 191
.HBLSP Syntax 192
Limitations 193
Input Syntax 193
Output Syntax 195
.PRINT and .PROBE Statements. 195
Output Data Files 196

7. Envelope Analysis 197

Envelope Simulation 197

Envelope Analysis Commands. 198

Nonautonomous Form 198

Oscillator Analysis Form. 199

Fast Fourier Transform Form 200

Output Syntax 201

.PRINT or .PROBE 201

.MEASURE. 202

Envelope Output Data File Format. 202

8. Post-Layout Analysis. 205

Post-Layout Back-Annotation. 205

Standard Post-Layout Flow 207

Standard Post-Layout Flow Control Options 208

Selective Post-Layout Flow 211

Selective Post-Layout Flow Control Options 212

Additional Post-Layout Options 213

Unsupported SPEF Options. 214

Selective Extraction Flow 214

Overview of DSPF Files 215

DSPF File Structure 216

DSPF File Example 219

Overview of SPEF Files 220

SPEF File Structure 220

SPEF File Example 226

Linear Acceleration 230

PACT Algorithm 231

PI Algorithm 232

Linear Acceleration Control Options Summary 232

Part III: Noise Analyses

9. Transient Noise Analysis 237

Overview of HSPICE Transient Noise Analysis 238

Modeling Frequency-Dependent Noise Sources 239

Monte Carlo Noise Analysis. 241

Monte Carlo Noise Techniques Available in HSPICE. 242
Setting up a Monte Carlo .TRANNOISE Analysis 242
Monte Carlo Input Syntax. 243
Monte Carlo Output Data 245
Controlling Noise in Subcircuits 245
Monte Carlo Examples. 246
Correlating Noise Results: .TRANNOISE (Monte Carlo) and .NOISE . . . 246

Stochastic Differential Equation (SDE) Analysis 247
SDE Approach Input Syntax 249
SDE Output Data 250
SDE Examples. 251

Jitter Measurements from .TRANNOISE Results 252
Output Data Files 252
Measure TIE Jitter and Jitter Spectrum Example. 253

Error Handling, Error Recovery, Status Reporting 255

References. 256

10. Simulation of Random Noise 257

Introduction to Noise Sources 257

Characteristics of Random Signals 258
Probability Distribution Function versus Power Spectral Density 258
Multiple Noise Sources in a Circuit 262
Summary 263

Noise Types 263
Thermal Noise 264
Flicker Noise 265
Shot Noise 268
Summary 269

Component Noise Models and HSPICE Noise Simulation 269
Element Noise Models 269
HSPICE Noise Simulation 270
Summary 275

Part IV: Behavioral Modeling Applications

11. Behavioral Modeling 279

Behavioral Design Process 280

Using Behavioral Elements 280

Controlled Sources 282

Libraries 282

Voltage and Current Controlled Elements 283

Modeling with Digital Behavioral Components. 283

Behavioral AND and NAND Gates 283

Behavioral D-Latch. 284

Behavioral Double-Edge Triggered Flip-Flop 286

Calibrating Digital Behavioral Components 288

Building Behavioral Lookup Tables 288

Subcircuit Definition 289

Behavioral N-Channel MOSFET 290

Creating a Behavioral Inverter Lookup Table 290

Optimizing Behavioral CMOS Inverters 292

Optimizing Behavioral Ring Oscillators 294

Analog Behavioral Elements 295

Behavioral Differentiator. 297

Ideal Transformer 298

Behavioral Amplitude Modulator 299

Behavioral Data Sampler 300

Op-Amps, Comparators, and Oscillators 301

741 Op-Amp from Controlled Sources. 301

Inverting Comparator with Hysteresis 304

Voltage-Controlled Oscillator (VCO) 305

LC Oscillator. 307

Phase-Locked Loops (PLL) 310

Phase Detector, with Multi-Input NAND Gates. 310

Phase Locked Loop Modeling 311

References. 315

12. **Modeling Filters and Networks** 317

Transient Modeling 317

Using G- and E-elements. 319

Laplace Transform Function Call 319

Element Statement Parameters 321

Z Transform Function Call 322

G- and E-element Notes 324

Laplace Band-Reject Filter 324

Laplace Low-Pass Filter 325

Circular Convolution Example 328

Notes 328

30-Degree Phase Shift Circuit File 328

Laplace and Pole-Zero Modeling 330

Laplace Transform (LAPLACE) Function 330

General Form of the Transfer Function 330

Finding the Transfer Function 331

Determining the Laplace Coefficients 334

Laplace Transform POLE (Pole/Zero) Function 338

POLE Function Call 338

General Form of the Transfer Function 338

Reduced Form of the Transfer Function 339

RC Line Modeling 342

AWE Transfer Function Modeling 344

Y-parameter Line Modeling 346

Comparison of Circuit and Pole/Zero Models 349

Simulation Time Summary 349

Modeling Switched Capacitor Filters 351

Switched Capacitor Network 351

Switched Capacitor Filter Example 352

Input File for Switched Capacitor Filter 354

References 356

Index 357

About This Guide

This user guide focuses on advanced analog analysis and modeling and concentrates on advanced analog circuit characterization, noise modeling and analysis, and behavioral analysis including Verilog-A.

Conventions

This manual follows these typographical conventions in Synopsys HSPICE documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Italic`	Indicates a user-defined value, such as *`object_name`*.
`Bold`	Indicates user input—text you type verbatim—in syntax and examples. For a graphical user interface, **Bold** indicates a GUI element such as a button, menu, field, or other control.
[ ]	Denotes optional parameters, such as: `write_file [-f` *`filename`*`]`
( )	When shown, the parentheses ( ) are part of the syntax. For example: + LISTFREQ=(1k 100k 10meg)
`...`	Indicates that parameters can be repeated as many times as necessary: *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low	medium	high`
+	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Convention	Description
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.

Customer Support

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

Accessing SolvNet

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

To access SolvNet:

1. Go to the SolvNet Web page at 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:

- Open a case with your local support center from the Web by going to https://solvnet.synopsys.com/EnterACall (Synopsys user name and password required). Choose the **Open A Support Case** tab to begin.
- Send an e-mail message to your local support center.
    - E-mail support_center@synopsys.com from within North America.

    - Find other local support center e-mail addresses at http://www.synopsys.com/support/support_ctr.

- Telephone your local support center.
    - Call (800) 245-8005 from within the continental United States.
    - Call (650) 584-4200 from Canada.
    - Find other local support center telephone numbers at http://www.synopsys.com/support/support_ctr.

1

HSPICE Advanced Analog Features

Describes how to use HSPICE advanced analog analyses, take advantage of its specialized features, and use the Custom WaveView tool; introduces the HSPICE solutions for noise analysis.

HSPICE advanced analog analyses is a set of analysis and design capabilities that supports the design of advanced analog and high-speed circuits. HSPICE advanced analog analyses includes several modeling, simulation, and measurement additions that augment the ultimate-accuracy analog circuit simulation capabilities.

HSPICE advanced analog analyses accepts a netlist file from standard input and delivers the ASCII text simulation results to HTML or to standard output. Standard error output reports error and warning messages.

These following sections cover these topics:

- Introduction to HSPICE Advanced Analog Analyses
- Use of Example Syntax

Introduction to HSPICE Advanced Analog Analyses

The following sections discuss the advanced analog features:

- HSPICE Advanced Analog Analyses Features
- Using HSPICE Advanced Analog Analyses
- HSPICE Advanced Analog Output Files
- Using Custom WaveView
- Creating a Configuration File

- Using Wildcards
- Limiting Output Data Size
- Generating Measurement Output Files
- Optimization
- Optimizing with HB Measurements
- Optimizing with HBNOISE or PHASENOISE Measurements
- Using CHECK Statements
- POWER DC Analysis
- Detecting and Reporting Surge Currents
- Advanced Analog Demonstration Input Files

HSPICE Advanced Analog Analyses Features

This section briefly introduces the features of both the simulation engine and the waveform display tool.

- Steady-state frequency-domain analyses for linear and nonlinear circuits.
- The `.HBLSP` command invokes periodically driven nonlinear circuit analyses for power-dependent S parameters.
- Harmonic Balance (`.HB`) analysis by using Direct and Krylov solvers. The `.HB` command invokes the single and multi tone Harmonic Balance algorithm for periodic steady state analysis.
- `TRANFORHB` element parameter to recognize V/I sources that include SIN and PULSE transient descriptions as well as PWL and VMRF sources.
- Harmonic balance-based periodic AC analysis. The `.HBAC` command invokes periodic AC analysis for analyzing small-signal perturbations on circuits that operate in a large-signal periodic steady state.
- Harmonic Balance-based Periodic Noise analysis (`.HBNOISE`) for noise analysis of periodically modulated circuits, includes stationary, cyclo-stationary, and frequency-dependent noise effects.
- Autonomous Harmonic Balance analysis. The `.HBOSC` command invokes the multi tone, oscillator-capable Harmonic Balance algorithm for periodic steady state analysis.

- Perturbation analysis for Oscillator Phase Noise. The `.HBAC` command invokes phase periodic AC noise for oscillators circuits that operate in a large-signal steady-state.
- Oscillator phase noise analysis, including both a nonlinear perturbation method and a PAC method, and includes stationary, cyclo-stationary, frequency-dependent, and correlated noise effects.
- Frequency translation S-parameter and noise figure extraction with the `.HBLIN` command.
- Envelope analysis. The `.ENV` command: invokes standard envelope simulation. The `.ENVOSC` command invokes envelope startup simulation. The `.ENVFFT` command invokes envelope Fast Fourier Transform simulation.
- `.OPTION HBTRANINIT`, `HBTRANPTS`, and `HBTRANSTEP` for transient analysis of ring oscillators.
- Calculation of the transfer function from an arbitrary source and harmonic in the circuit to a designated output with the `.HBXF` command.
- `.OPTION SIM_ACCURACY` provides simplified accuracy control for all simulations while `.OPTION SIM_ORDER` and `SIM_TRAP` improve transient analysis simulation controls.
- DSPF Flow for fast analysis by using parasitic data from layout.
- Shooting Newton steady-state time domain analysis; the Shooting Newton algorithm provides functionality to support the following commands: `.SN`, `.SNAC` `.SNFT`, `.SNNOISE`, `.SNOSC`, and `.SNXF`.
- Periodic Time-Dependent Noise Analysis (`.PTDNOISE`) calculates the noise spectrum and the total noise at a point in time. This analysis determines jitter in a digital threshold circuit from the total noise and the digital signal slew rate.
- HSPICE advanced analog analyses supports ISUB syntax with the exception of wildcard support with the “?” sign. For example, HSPICE advanced analog analyses does not support `isub(x1.a?)`.
- HSPICE advanced analog analyses supports HSPICE Precision Parallel (-HPP) for multi-threading simulations.
- HSPICE advanced analog analyses supports case sensitivity.

HSPICE advanced analog analyses also includes the following measurement capabilities:

- 1dB compression point.
- Intercept points (for example, IP2, IP3).
- Mixer conversion gain and noise figure.
- VCO output spectrum.
- Oscillator phase noise.
- Options simplify specifying levels of accuracy. As a result, HSPICE provides effective simulation solutions for advanced analog, high-speed, and PCB signal integrity circuit challenges.
- Verilog-A is supported for all advanced analog analyses.

  Standard restrictions for Verilog-A in periodic steady-state analysis are the same as other advanced analog simulators that use Verilog-A. For example:

  - Verilog-A modules that are time-dependent are illegal for HB or SN unless the time dependence is periodic with a period that matches the HB or SN setup.
  - Verilog-A modules with “internal states” may not work correctly in HB or SN because the engine cannot track the internal state, so HB or SN may accept convergence to a periodic steady-state even though the internal state may not be in periodic steady state.
  - Some event-driven constructs in Verilog-A may not be compatible with HB.
- For netlist input guidelines, see Input Netlist and Data Entry in the *HSPICE User Guide: Basic Simulation and Analysis*.
- For information on use of Parameters and Functions, see Parameters and Functions in the *HSPICE User Guide: Basic Simulation and Analysis*.
- For information on use of Monte Carlo sweeps, see Monte Carlo - Traditional Flow Statistical Analysis in the *HSPICE User Guide: Basic Simulation and Analysis*.

Using HSPICE Advanced Analog Analyses

HSPICE advanced analog analyses provides several analyses that support the simulation and analysis of radio-frequency integrated circuits (RFICs). The advanced analog analyses include:

- Steady-State Harmonic Balance Analysis
- Steady-State Shooting Newton Analysis and Shooting Newton with Fourier Transform (.SNFT)
- Harmonic Balance Oscillator Analysis (.HBOSC)
- Shooting Newton Oscillator Analysis(.SNOSC)
- Large Signal Periodic AC, Transfer Function, and Noise Analyses and Multitone Harmonic Balance AC Analysis (.HBAC)
- Shooting Newton AC Analysis (.SNAC)
- Large Signal Periodic AC, Transfer Function, and Noise Analyses
- Shooting Newton Noise Analysis (.SNNOISE)
- Multitone Harmonic Balance Transfer Function Analysis (.HBXF)
- Shooting Newton Transfer Function Analysis (.SNXF)
- Frequency Translation S-Parameter (HBLIN) Extraction
- Envelope Analysis

Important: You can enable analytical derivative computation of expression-based element evaluations in all advanced analog analyses for extensive accuracy by using the .OPTION EQN_ANALYTICAL_DERIV described in the *HSPICE Reference Manual: Commands and Control Options*.

HSPICE Advanced Analog Output Files

The following table shows the output file extensions that HSPICE advanced analog analyses produce. The base file name of each output file is the same as the input netlist file's base name. The `#` at the end of each file extension represents the `.ALTER` run from which the file came.

In general, text output from `.PRINT` commands is for users, while binary output from `.PROBE` or `.OPTION POST` is input for the Custom WaveView tool.

Command	Description	Text Output	Output for Custom WaveView
.ENV	Envelope analysis	`.printev#`	`.ev#`
.ENVFFT	Envelope FFT	(none)	`.fe#`
.ENVOSC	Oscillator Startup	`.printev#`	`.ev#`
.HB	Harmonic Balance	`.printhb#`	`.hb#`
.HBAC	Harmonic Balance AC	`.printhb#`	`.hb#`
.HBLIN	Harmonic Balance Linear Analysis	.PRINT output: `.printhl#` S-param output: `.SnP`	.PROBE output: `.hl#` S-param output: `.SnP`
.HBLSP	.HBLSP large-signal	.PRINT output: `.printls#` S-param output: `.p2d`	.PROBE output: `.ls#` S-param output: `.p2d`
	.HBLSP small-signal	.PRINT output: `.printss#` S/noise output: `.S2P#`	.PROBE output: `.ss#` S/noise output: `.S2P#`
.HBNOISE	HBAC noise	`.printsnpn#`	`.pn#`
.HBOSC	Harmonic Balance OSC	`.printhb#`	.hb#
.HBXF	Transfer Functions	`.printxf#`	`.xf#`
.PHASENOISE	Phase Noise (SNOSC)	`.printsnpn#`	`.pn#`
	Phase Noise (HBOSC)	`.printpn#`	`.pn#`
	Jitter	.printjt#	`.jt#`
.PTDNOISE	Periodic Time Dependent Noise (HB)	`.printptn#`	`.ptn#`
	Periodic Time Dependent Noise (SN)	`.printsnptn#`	`.snptn#`
.SN	Shooting Newton Analysis	`.printsn#`	`.sn#`
.SNAC	Shooting Newton AC	`.printsnac#`	`.snac#`
.SNNOISE	Shooting Newton Noise	`.printsnpn#`	`.snpn#`

Command	Description	Text Output	Output for Custom WaveView
.SNOSC	Shooting Newton OSC	`.printsn#`	`.sn#`
.SNXF	Shooting Newton Transfer Function	`.printsnxf#`	`.snxf#`
.TRANNOISE	Transient Noise	`.printtrpn#`	`.trpn#`
	Jitter	`.printtrjt#`	`.trjt#`
	Autocorrelation Function	none	`.trzn#`

Using Custom WaveView

Synopsys Custom WaveView supports viewing and processing of HSPICE output files. This section presents a basic overview of how to use the tool.

- To start the Custom WaveView tool, type **wv** on the UNIX/Linux command line.
- **Choose File** > **Import Waveform File** (or press CTRL-O) to open the Open Waveform Files dialog box. Use the **File Filters** to limit the file names to Waveform Files. The preceding table lists the HSPICE advanced analog file types. When you open a file, its contents appear in the file browser. The file browser lists all open plot files. Click on the '+' near a waveform file name, to display the hierarchy of the waveform. Clicking on the top level or any hierarchy level to display the contents of the waveform file to appear in the signal browser. To plot one of the signals listed here in the waveform, you can either double-click the signal label or select, drag, and drop the signal label to the waveview.
- To create a panel, use the **Panel** > **New** menu and select the panel type, **X-Y**, **Smith Chart**, or **Polar Plot**. You can also use the panel icon in the tool bar to create new panels.
- To create a new chart, use the **File** > **New** menu. Select either **XY Graph**, **Smith Chart**, or **Polar Chart**. You can also use the first three icons in the tool bar to create new chart windows.
- Use the Custom WaveView tool bar to change how signals look, delete signals, group or ungroup signals.

- Right-click on a frequency-domain signal name and use the **To Time Domain** command to convert the histogram waveform (for example, from an `.hb0` file) to a time domain waveform.
- Configure the axis scale and grid by right-clicking a horizontal or vertical axis and selecting the desired scale or grid from the context sensitive menu.
- Zoom in and out, using the zoom icons on the tool bar, or use the mouse cursor to select an area directly on the waveform.
- You can use dynamic meters to see the signal's precise value at different points. From the menu, select **Tools** > **Dynamic Meter** or use the Dynamic Meter icon in the tool bar. Select and configure the desired Dynamic Meter. You can then move the meter to the desired location on the selected signal.
- To use the measurement tools, choose **Tools** > **Measurement**. You can use the following advanced analog options available under the **All** tab of the Measurement Tool window:
  - 1db compression point (P1dB).
  - 2nd order intercept point (IP2).
  - 3rd order intercept point and spurious free dynamic range (IP3/SFDR).

Creating a Configuration File

You can create a configuration file, called *.hspicerf*, to customize your HSPICE advanced analog simulation. HSPICE first searches for *.hspicerf* in your current working directory, then in your home directory as defined by `$HOME`. Following are the configuration options available in HSPICE:

Keyword	Description	Example
flush_waveform	Flushes a waveform. If you do not specify a percentage, then the default value is 20%.	`flush_waveform percent%`
ground_floating_node	Uses .IC statements to set floating nodes in a circuit to ground. You can select three options for grounding floating nodes: ■ If set to `1,` grounds only floating nodes (gates, bulk, control nodes, non-rail bulk) that the .IC set includes. ■ If set to 2, adds unconnected terminals to this set. ■ If set to 3, uses .IC statements to ground all floating nodes, including dangling terminals.	`ground_floating_ node 1`
hier_delimiter	Changes the delimiter for subcircuit hierarchies from "." to the specified symbol.	`hier_delimiter /`

Keyword	Description	Example
html	Stores all HSPICE output in HTML format.	`htmlhspicerf test` This example creates a file named test.html in the current directory.
integer_node	Removes leading zeros from node names. For example, HSPICE considers 0002 and 2 to be the same node. Without this keyword, 0002 and 2 are two separate nodes.	`integer_node`
max_waveform_size	Automatically limits the waveform file size. ■ If the number is less than 5000, HSPICE resets it to 2G. ■ If you do not set the number, HSPICE uses the default, 2G. ■ If you do not set the line, the file size has no limit.	`max_waveform_ size 2000000000`
negative_td	Allows negative time delay input in `pwl` (piecewise linear with repeat), `pl` (piecewise linear), `exp` (exponential, rising time delay only), `sin` (damped sinusoidal), `pulse` (trapezoidal pulse), and `am` (amplitude modulation) formats.	If you do not set `negative_td`, a negative time delay defaults to zero.
port_element_ voltage_ matchload	Allows the alternate Port element definition. A Port element consists of a voltage source in series with a resistor. For the explanation that follows, let the user-specified DC, AC, or transient value of the Port element be V, and let the voltage across the overall port element be Vp. By default, HSPICE advanced analog analyses sets the internal voltage source value to V. The value of Vp is lower than V, depending on the internal impedance and the network's input impedance. With the alternate definition, the internal voltage source value is adjusts to 2*V, so that Vp=V when the Port element's impedance matches the network input impedance. The actual value of Vp still depends on the port and network impedances.	`port_element_ voltage_ matchload`
rcxt_divider	Defines the hierarchy delimiter in the active nodes file in RCXT format.	`rcxt_divider /`
skip_nrd_nrs	Directs HSPICE advanced analog analyses to consider transistors with matching geometries (except for NRD and NRS) as identical for pre-characterization purposes.	`skip_nrd_nrs`
unit_atto	Activates detection of the “atto.” unit. Otherwise, HSPICE advanced analog analyses assumes that “a” represents “amperes.”	`unit_atto`
v_supply	Changes the default voltage supply range for characterization.	`v_supply 3`

Keyword	Description	Example
wildcard_left_range	Begins range expression.	`wildcard_left_ range [`
wildcard_match_all	Matches any group of characters.	`wildcard_match_ all *`
wildcard_match_one	Matches any single character.	`wildcard_match_ one ?`
wildcard_right_range	Ends range expression.	`wildcard_right_ range ]`

Note: For more information about wildcards, see Using Wildcards.

This section covers the following topics:

- Inserting Comments in a .hspicerf File

Inserting Comments in a .hspicerf File

To insert comments into your .hspicerf file, include a number sign character (#) as the first character in a line. For example, this configuration file shows how to use comments in a *.hspicerf* file:

```
# sample configuration file
# the next line of code changes the delimiter
# for subcircuit hierarchies from "," to "^"
hier_delimiter ^
# the next line of code matches any groups of "*" characters
wildcard_match_all *
# the next line of code matches one "?" character
wildcard_match_one ?
# the next line of code begins the range expression with
# the "[" character
wildcard_left_range [
# the next line of code ends the range expression with
# the "]" character
wildcard_right_range ]
```

Using Wildcards

You can use wildcards to match node names. HSPICE advanced analog uses wildcards somewhat differently than standard HSPICE.

Before you use wildcards, you must define the wildcard configuration in a .hspicerf file. For example, you can define the following wildcards in a .hspicerf file:

```
file .hspicerf
wildcard_match_one     ?
wildcard_match_all     *
wildcard_left_range    [
wildcard_right_range   ]
```

The `.PRINT`, `.PROBE`, `.LPRINT`, and `.CHECK` statements supports wildcards in HSPICE.

For more information about using wildcards in an HSPICE configuration file, see Using Wildcards in PRINT and PROBE Statements in the *HSPICE User Guide: Basic Simulation and Analysis*.

Limiting Output Data Size

For multi-million transistor simulations, an unrestricted waveform file can grow to several gigabytes in size. The file becomes unreadable in some waveform viewers, and requires excessive space on the hard drive.

This section describes options that limit the number of nodes output to the waveform file to reduce the file size. HSPICE supports the following options to control the output:

Control Option	Description
.OPTION SIM_POSTTOP	Use this option to limit the data written to your waveform file to data from only the top n level nodes. This option outputs instances up to n levels deep. **Note:** To enable the waveform display interface, you also need the `POST` option.
.OPTION SIM_POSTSKIP	Use this option to have the `SIM_POSTTOP` option skip any instances and their children that the subckt_definition defines.
.OPTION SIM_POSTAT	Use this option to limit the waveform output to only the nodes in the specified subcircuit instance. This option can operate in conjunction with the `SIM_POSTTOP` option and when present, has precedence over the `SIM_POSTSKIP` option.

Control Option	Description
.OPTION SIM_POSTDOWN	Use this option to include an instance and all children of that instance in the output. This option can operate in conjunction with the `SIM_POSTTOP` option and when present, has precedence over the `SIM_POSTSKIP` option.
.OPTION SIM_POSTSCOPE	Use this option to specify the signal types to probe from within a scope.

Generating Measurement Output Files

You can make measurements with the `.MEASURE` statement when using HSPICE advanced analog analyses.

The results of the `.MEASURE` statements appear in a file with one of the following filename extensions:

- `.mb#` for measurements in HB analysis
- `.mp#` for measurements in HBNOISE and SNNOISE analysis
- `.mpn#` for measurements from PHASENOISE analysis when using HB to obtain the steady state solution
- `.msnpn#` for measurements from PHASENOISE analysis when using SN to obtain the steady state solution
- `.msnptn#` for measurements in PTDNOISE analysis

For more information about `.MEASURE` statements, see HSPICE Netlist Commands in the *HSPICE Reference Manual: Commands and Control Options*.

Optimization

To perform optimization, create an input netlist file that specifies:

- Optimization parameters with upper and lower boundary values along with an initial guess.
- A HB or HBOSC optimization statement
- An optimization model statement
- Optimization measurement statements for optimization parameters

If you provide the input netlist file, optimization specifications, limits, and initial guess, then the optimizer reiterates the simulation until it finds an optimized solution.

Usage Notes and Examples

- Optimization works for HB, HBOSC, and HBAC analyses.
- You can add the `GOAL` options in every meaningful `.MEASURE` statement, like `FIND-WHEN`, `FIND-AT`, and so forth.
- A data sweep does not need definition in the `.HB` statement for HB optimization to use the measured result from `.MEASURE HBNOISE`, or `PHASENOISE` statements. Therefore, this type of optimization does not support parameter sweep.
- Optimize multiple parameters with multiple goals by selecting `.MODEL OPT LEVEL=0` (modified Lavenberg-Marquardt method).
- Optimize single parameters in single measurement situations by selecting `.MODEL OPT LEVEL=1` (bisection method).
- Examples
  - Setting optimization parameters

    ```
    .param W=opt1(231u, 100u, 800u)
    .param Rs=opt1(10,8,20)
    ```
  - Optimization analysis statement

    ```
    .HB tones=2.25g 2.5g nharms=6,3
     + sweep Pin_dbm -30 0 2
     + sweep optimize = opt1
     + results = gain $measure result to tune the parameters
     + model= optmod1
    ```
  - Selecting an optimization model

    ```
    .model optmod1 opt level=1 $Bisection method
     + itropt=40 relin=1e-4 relout=1e-6 $ accuracy settings
    ```
  - Measurement statements to tune the optimization parameters

    ```
    .measure HB vif find vdb(if+)[-1,1] at 10e-6
    .measure HB vrf find vdb(rf+)[0,1] at 10e-6
    .measure HB gain=param('vif-vrf') goal=-2
    ```
  - Measurement statement to find the fundamental frequency from HB analysis:

    ```
    .measure HB frequency_max FIND ‘HERTZ[1]’ at=0
    ```

Optimizing with HB Measurements

The required statements are:

- Analysis statement

```
.HB TONES=f1[f2 ... fn] [NHARMS=h1 [,h2 ... hn]]
+ SWEEP parameter_sweep OPTIMIZE=OPTxxx RESULT=measname
+ MODEL=mname
```

- Measure statement

```
.MEASURE HB measname FIND out_var1 AT=val GOAL=val
```

Optimizing with HBNOISE or PHASENOISE Measurements

The required statements are:

- Analysis statement

```
.HB TONES=f1[f2...fn] [NHARMS=h1 [,h2...hn]]
+ SWEEP OPTIMIZE=OPTxxx RESULT=measname MODEL=mname
```

For example:

```
.HBOSC tone=1g nharms=5 PROBENODE=out,gnd,0.8
+ SWEEP OPTIMIZE=opt1 RESULT=y1,y2 MODEL=m1
.MODEL m1 OPT level=0
.PHASENOISE v(out) DEC 1 1k 1G
.MEAS PHASENOISE y1 FIND PHNOISE at 10k goal=-150dBc
.MEAS PHASENOISE y2 RMSJITTER PHNOISE units=sec goal=1p
```

- Measure statement

```
.MEASURE HBNOISE measname FIND out_var1 AT=val GOAL=val
.MEASURE PHASENOISE measname FIND out_var1 AT=val
+ GOAL=val
```

Using CHECK Statements

The CHECK statements in HSPICE advanced analog analyses offer the following instrumentation:

- Setting Global Hi/Lo Levels
- Slew, Rise, and Fall Conditions

- Edge Timing Verification
- Setup and Hold Verification
- IR Drop Detection

The results of these statements appear in a file with an `.err` extension. To prevent creating unwieldy files, HSPICE advanced analog analyses reports only the first 10 violations for a particular check in the .err file.

Setting Global Hi/Lo Levels

You use the `.CHECK GLOBAL_LEVEL` statement to globally set the desired high and low definitions for all CHECK statements. For example,

```
.CHECK GLOBAL_LEVEL (hi lo hi_th lo_th)
```

This statement defines values for hi, lo, and the thresholds.

For syntax and description of this statement, see .CHECK GLOBAL_LEVEL in the *HSPICE Reference Manual: Commands and Control Options*.

Slew, Rise, and Fall Conditions

You use the `.CHECK SLEW` statement to verify that a slew rate occurs within the specified window of time. For example,

```
.CHECK SLEW (min max) node1 [node2 ...] [(hi lo hi_th lo_th)]
```

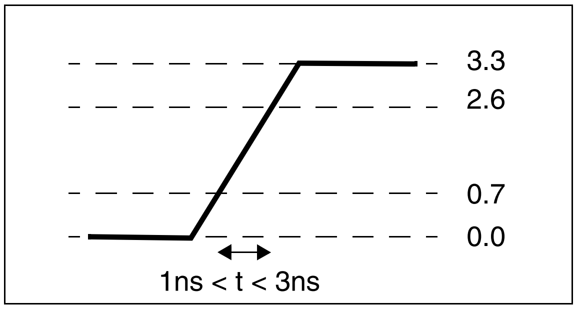


Figure 1 SLEW Example

For syntax and description of this statement, see .CHECK SLEW in the *HSPICE Reference Manual: Commands and Control Options*.

You use the `.CHECK RISE` statement to verify that a rise time occurs within the specified window of time. For example,

```
.CHECK RISE (min max) node1 [node2 ...] [(hi lo hi_th lo_th)]
```

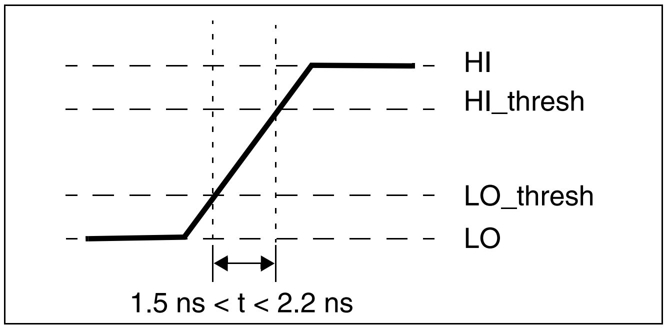


Figure 2 RISE Time Example

For syntax and description of this statement, see .CHECK RISE in the *HSPICE Reference Manual: Commands and Control Options*.

You use the `.CHECK FALL` statement to verify that a fall time occurs within the specified window of time. For example,

```
.CHECK FALL (min max) node1 [node2 ...>] [(hi lo hi_th lo_th)]
```

For syntax and description of this statement, see .CHECK FALL in the *HSPICE Reference Manual: Commands and Control Options*.

Edge Timing Verification

The edge condition verifies that a triggering event provokes an appropriate RISE or FALL action, within the specified time window. You use the `.CHECK EDGE` statement to verify this condition. For example,

```
.CHECK EDGE (ref RISE|FALL min max RISE|FALL)
+ node1 [node2 . . . ] [(hi lo hi_th low_th)]
```

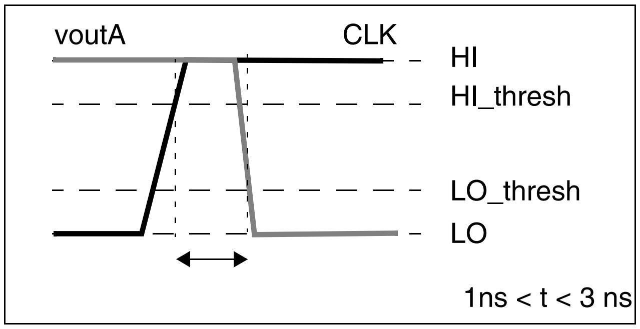


Figure 3 EDGE Example

For syntax and description of this statement, see .CHECK EDGE in the *HSPICE Reference Manual: Commands and Control Options*.

Setup and Hold Verification

You use the `.CHECK SETUP` and `.CHECK HOLD` statements to ensure that specified signals do not switch for a specified period of time. For example,

```
.CHECK SETUP (ref RISE|FALL duration RISE|FALL) node1
+[node2 . . . ] [(hi lo hi_th low_th)]
.CHECK HOLD (ref RISE|FALL duration RISE|FALL) node1
+[node2 . . . ] [(hi lo hi_th low_th)]
```

- For a SETUP condition, this is the minimum time before the triggering event, during which the specified nodes cannot rise or fall.

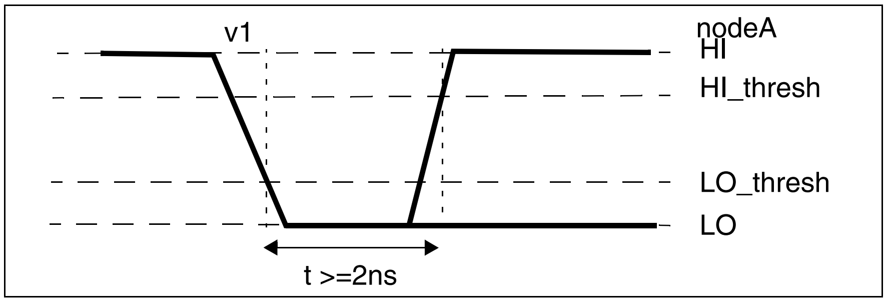


Figure 4 SETUP Example

For syntax and description of this statement, see .CHECK SETUP in the *HSPICE Reference Manual: Commands and Control Options*.

- For a HOLD condition, specify the minimum time after the triggering event, before the specified nodes can rise or fall.

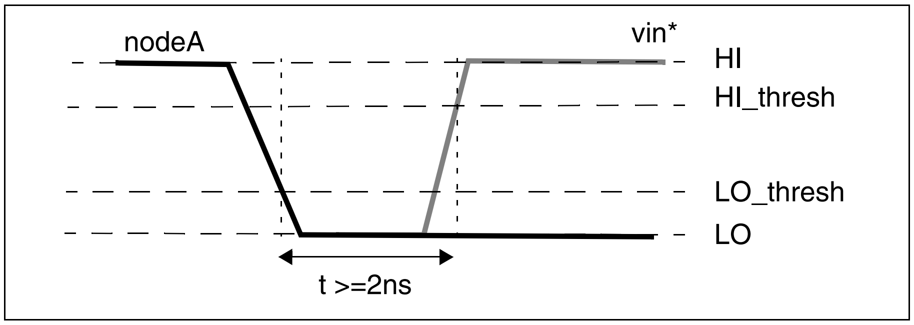


Figure 5 HOLD Example

For syntax and description of this statement, see .CHECK HOLD in the *HSPICE Reference Manual: Commands and Control Options*.

IR Drop Detection

You use the `.CHECK IRDROP` statement to verify that the IR drop does not exceed, or does not fall below, a specified value for a specified duration. For example,

```
.CHECK IRDROP ( volt_val time ) node1 [node2 . . . ]
+ [( hi lo hi_th low_th )]
```

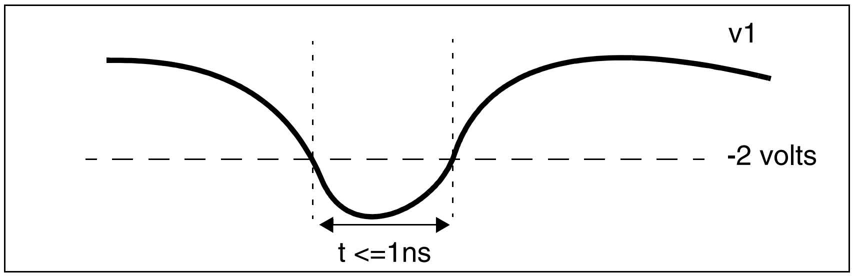


Figure 6 IR Drop Example

For syntax and description of this statement, see .CHECK IRDROP in the *HSPICE Reference Manual: Commands and Control Options*.

POWER DC Analysis

You use the `.POWERDC` (standby current) statement to calculate the DC leakage current of a design hierarchy. For example,

```
.POWERDC keyword subckt_name1...
```

This statement creates a table that lists the measurements of the AVG, MAX, and MIN values for the current of every instance in the subcircuit. This table also lists the sum of the power of each port in the subcircuit.

You use the `SIM_POWERDC_HSPICE` option to increase the accuracy of operating point (OP) calculations.

Or for even higher accuracy in operating point calculations, you use the `SIM_POWERDC_ACCURACY` option.

For syntax and description of this statement and options, see .POWERDC, .OPTION SIM_POWERDC_ACCURACY, or .OPTION SIM_POWERDC_HSPICE in the *HSPICE Reference Manual: Commands and Control Options.*

This section covers the following topics:

- Power DC Analysis Output Format
- POWER Analysis

Power DC Analysis Output Format

```
*** Leakage Current Result ***
Subckt Name=XXX
Instance Name       Port   Max(A)   Min(A)   Avg(A)
Total Power          Max(W)   Min(W)   Avg(W)
NOTE:    Power=Sum{Ii * Vi}
Subckt Name=XXX
Instance Name       Port   Max(A)   Min(A)   Avg(A)
Total Power          Max(W)   Min(W)   Avg(W)
```

Example:

```
.global vdd vss
.powerdc all
x1 in1 mid1 inv
x2 mid1 out1 inv
.subckt inv in out
mn out in vss vss nch
mp vdd in out vdd pch
.ends
.end
```

Output:

```
*** Leakage Current Result ***
Subckt Name=Top Level
Instance Name       Port    Max(A)    Min(A)    Avg(A)
   x1   in      .......
   x1   out      .......
   x2   in      .......
   x2   out      .......
   Total Power        .......
Subckt Name=inv
Instance Name       Port    Max(A)    Min(A)    Avg(A)
   mn   d      .......
   mn   g      .......
   mn   s      .......
   mn   b      .......
   mp   d      .......
   mp   g      .......
   mp   s      .......
   mp   b      .......
   Total Power        .......
```

POWER Analysis

The `.POWER` statement in HSPICE creates a table, which by default contains the measurements for `AVG`, `RMS`, `MAX`, and `MIN` for every signal specified. For example,

```
.POWER signals [REF=vname FROM=start_time TO=end_time]
```

By default, the scope of these measurements are from 0 to the maximum time point specified in the `.TRAN` statement.

For syntax and description of `.POWER` statement, see .POWER in the *HSPICE Reference Manual: Command and Control Options*.

In the following example, no simulation start and stop time exists for the `x1.in` signal, so the simulation scope for this signal runs from the start (0ps) to the last `.tran` time (`100ps`).

```
.power x1.in
.tran 4ps 100ps
```

In the following example, you can use the `FROM` and `TO` times to specify a separate measurement start and stop time for each signal. In this example:

- The scope for simulating the `x2.in` signal is from 20ps to 80ps.
- The scope for simulating the `x0.in` signal is from 30ps to 70ps.

```
.param myendtime=80ps
.power x2.in REF=a123 from=20ps to=80ps
.power x0.in REF=abc from=30ps to='myendtime - 10ps'
```

Setting Default Start and Stop Times

In addition to using `FROM` and `TO` times in a `.POWER` statement, you can also use the `SIM_POWERSTART` and `SIM_POWERSTOP` options with `.POWER` statements to specify default start and stop times for measuring signals during simulation. These times apply to all signals that do not have their own defined `FROM` and `TO` measurement times. For example,

```
.OPTION SIM_POWERSTART=time
.OPTION SIM_POWERSTOP=time
```

These options control the power measurement scope; the default is for the entire run.

For syntax and description of these options, see .OPTION SIM_POWERSTART or .OPTION SIM_POWERSTOP in the *HSPICE Reference Manual: Command and Control Options*.

Controlling Power Analysis Waveform Dumps

Use the `SIM_POWERPOST` option to dump the control-power analysis waveform. For example,

```
.OPTION SIM_POWERPOST=ON|OFF
```

When you consider the potentially enormous number of signals, there is no waveform dump by default for the signals in the `.POWER` statement. Setting `SIM_POWERPOST=ON` turns on power analysis waveform dumping.

Detecting and Reporting Surge Currents

The `.SURGE` statement in HSPICE advanced analog analyses automatically detects and reports a current surge that exceeds the specified surge tolerance. For example,

```
.SURGE surge_threshold surge_width node1 [node2 .... noden]
```

This statement reports any current surge that is greater than *surge_threshold* for a duration of more than *surge_width*.

For additional information, see .SURGE in the *HSPICE Reference Manual: Commands and Control Options*.

Advanced Analog Demonstration Input Files

The following is a listing of shipped demonstration files for illustrating HSPICE advanced analog analyses functionality. All of these example files are available at:

`$installdir/demo/hspice/rf_examples`

File Name	Description
`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
`gsmlna.sp`	LNA Linear analysis example
`gsmlnaIP3_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
`pfdcpGain.sp`	Shooting Newton analysis example
`phasefreqdet.sp`	Shooting Newton and noise analysis example
`ringoscSN.sp`	Shooting Newton and Phase Noise analysis example
`tsmc018.m`	Transistor model library used by `ringoscSN.sp`

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 in *HSPICE User Guide: Basic Simulation and Analysis*). Attempting to copy and paste from the book or help documentation may present unexpected results, as text used in formatting may include hidden characters, white space, etc. for visual clarity only.

Part 1: Time Domain, Steady-State Analysis

Part 1 presents the following chapters /topics:

- Chapter 2, Steady-State Harmonic Balance Analysis
- Chapter 3, Steady-State Shooting Newton Analysis

2 Steady-State Harmonic Balance Analysis

Describes how to use harmonic balance analysis for frequency-driven and steady-state analysis.

This chapter provides an introduction to Harmonic Balance (`.HB`) analysis command. You can use steady-state analysis on a circuit if it contains only DC and periodic sources. These analyses assume that all “start-up” transients have completely died out with only the steady-state response remaining. These analyses treat sources that are not periodic or DC as zero-valued. For more information, see Chapter 3, Steady-State Shooting Newton Analysis.

The following sections discuss these topics:

- Harmonic Balance Analysis
- Steady-State HB Sources
- Phase Differences Between HB and SIN Sources
- Tutorial Examples Harmonic Balance Analysis
- References

Harmonic Balance Analysis

Harmonic balance analysis (HB) is a frequency-domain, steady-state analysis technique. Use this analysis technique on a circuit excitable by DC and periodic sources of one or more fundamental tones. The solution that HB finds is a set of phasors for each harmonic signal in the circuit. You can think of this solution as a set of truncated Fourier series. HSPICE advanced analog analyses allows you to specify the solution spectrum to use in an analysis. HB analysis then finds the set of phasors at these frequencies that describes the circuit response. The result is a set of complex valued Fourier series coefficients that represent the waveforms at each node in the circuit.

Evaluation of linear circuit elements takes place in the frequency domain, while evaluation of nonlinear elements occurs in the time domain. The nonlinear response then transforms to the frequency domain where it adds to (or “balances” with) the linear response. The resulting composite response satisfies KCL and KVL (Kirchoff's current and voltage laws) with the circuit solution.

Typical applications include performing intermodulation analysis, oscillator analysis, and gain compression analysis, on amplifiers and mixers. HB analysis also serves as a starting point for periodic AC and noise analyses.

For more information on control options, see .HB command in the *HSPICE Reference Manual: Commands and Control Options*.

This section covers the following topics:

- Features Supported
- Harmonic Balance Equations
- Harmonic Balance Output Measurements
- HB .PRINT and .PROBE Output Syntax
- HB Output Data Files and Examples
- HB Error and Warning Messages
- Calculating Power Measurements After HB Analyses
- Calculations for Time-Domain Output
- Using .MEASURE with .HB Analyses

Features Supported

HB supports the following features:

- All existing HSPICE advanced analog analyses models.
- Unlimited number of independent input tones.
- Sources with multiple HB specifications.
- `SIN`, `PULSE`, `VMRF`, and `PWL` sources with `TRANFORHB=1`.

Prerequisites and Limitations

The following prerequisites and limitations apply to HB:

- Requires one `.HB` statement.
- Treats sources without a DC, HB, or TRANFORHB description as a zero-value for HB unless the sources have a transient description, in which case, the `time=0` value becomes a DC value.

Harmonic Balance Equations

We can write Kirchoff's current law in the time domain as:

Equation 1

$$f(v,t) = i(v(t)) + \frac{d}{dt}q(v(t)) + \int_{-\infty}^{t} y(t-\tau)v(\tau)d\tau + i_s(t) = 0$$

- `i(v(t))` represents the resistive currents from nonlinear devices
- `q` represents the charges from nonlinear devices
- `y` represents the admittance of the linear devices in the circuit
- i_s represents the vector of independent current sources
- `v` is a variable that represents the circuit unknowns, both node voltages and branch currents, and `f(v,t)` is an error term that goes to zero to satisfy Kirchoff's current law.

Transforming this equation to the frequency domain results in:

Equation 2

$$F(V) = I(V) + \Omega Q(V) + Y(\omega)V + I_s = 0$$

Note: Time-differentiation transforms to multiplication by $j\omega$ terms (which make up the Ω matrix) in the frequency domain. The convolution integral transforms to a simple multiplication. The Y matrix is the circuit's modified nodal admittance matrix.

All terms above are vectors that represent the circuit response at each analysis frequency.

The following equation shows the vector of (complex-valued) unknowns in the frequency domain for a circuit with *K* analysis frequencies and *N* unknowns.

Equation 3

$$V = \left[V_{(1,0)}\ V_{(1,1)}\ \cdots\ V_{(1,K-1)}\ V_{(2,0)}\ \cdots\ V_{(N,K-1)}\right]$$

HSPICE advanced analog analyses finds the unknown vector (*V*), to satisfy the system of nonlinear equations in the equation above. The Newton-Raphson technique uses either a direct solver to factor the Jacobian matrix, or an indirect solver. The HSPICE advanced analog analyses indirect solver is in the Generalized Minimum Residual (GMRES) Solver, a Krylov technique, and uses a matrix-implicit algorithm.

Harmonic Balance Output Measurements

This section explains the harmonic balance output measurements you receive after HSPICE runs an HB simulation.

The HB cosine sources interpret in real/imaginary and polar formats according to Equation 4:

Equation 4

$$\begin{aligned} v(t) &= A\cos(\alpha t + \phi) = Re\{Ae^{j(\alpha t + \phi)}\} = Re\{Ae^{j\phi}e^{j\omega t}\} \\ &= Re\{Ae^{j\phi}[\cos(\alpha t) + j\sin(\alpha t)]\} \\ &= Re\{[V_R + jV_I][\cos(\alpha t) + j\sin(\alpha t)]\} \\ &= V_R\cos(\alpha t) - V_I\sin(at) \\ &= A\cos(\phi)\cos(\alpha t) - A\sin(\phi)\sin(\alpha t) \end{aligned}$$

Note that this equation relates real/imaginary and polar formats with the standard convention:

Equation 5

$$\begin{aligned} V_R + jV_I &= Ae^{j\phi} \\ V_R &= A\cos(\phi) \\ V_I &= A\sin(\phi) \\ A &= \sqrt{V_R^2 + V_I^2} \\ \tan\phi &= \frac{V_I}{V_R} \end{aligned}$$

The result of HB analysis is a complex voltage (current) spectrum at each circuit node (or specified branch). Let `a[i]` be the real part and `b[i]` be the imaginary part of the complex voltage at the `ith` frequency index. The Fourier series expansion gives the conversion to a steady-state time-domain waveform is as in Equation 6:

Equation 6

$$\begin{aligned} v(t) = a[0] &+ a[1]*\cos(2\pi f[1]*t) - b[1]*\sin(2\pi f[1]*t) \\ &+ a[2]*\cos(2\pi f[2]*t) - b[2]*\sin(2\pi f[2]*t) \\ &+ a[3]*\cos(2\pi f[3]*t) - b[3]*\sin(2\pi f[3]*t) \\ &+ \ldots \\ &+ a[N]*\cos(2\pi f[N]*t) - b[N]*\sin(2\pi f[N]*t) \end{aligned}$$

Where:

- `v(t)` is the resulting time domain waveform.
- `N+1` is the total number of harmonics (including DC) in the frequency domain spectrum of the `*.hb0` file (the zero-th data point represents DC).
- `a[i]` is the real component at the `ith` frequency
- `b[i]` is the imaginary component at the `ith` frequency
- `f[i]` is the `ith` frequency value (with i=0 representing the zero frequency DC term). These frequencies do not need a harmonic relationship.

This frequency domain (Fourier coefficient) representation converts to a steady-state time domain waveform when you use the `.PRINT` or `.PROBE HBTRAN` output option or you invoke the **To Time Domain** function on complex spectra within Custom WaveView.

HB .PRINT and .PROBE Output Syntax

This section describes the syntax for the HB `.PRINT` and `.PROBE` statements.

```
.PRINT HB TYPE(NODES|ELEMENTS)[INDICES]
```

```
.PROBE HB TYPE(NODES|ELEMENTS)[INDICES]
```

Parameter	Description
TYPE	Specifies a harmonic type node or element. TYPE can be one of the following: ■ Voltage type: V = voltage magnitude and phase in degrees VR = real component VI = imaginary component VM = magnitude VP - Phase in degrees VPD - Phase in degrees VPR - Phase in radians VDB - dB units VDBM - dB relative to 1 mV ■ Current type: I = current magnitude and phase in degrees IR = real component II = imaginary component IM = magnitude IP - Phase in degrees IPD - Phase in degrees IPR - Phase in radians IDB - dB units IDBM - dB relative to 1 mV ■ Power type: P = Power in Watts or Pdbm = Power in dBm ■ Frequency type: ‘HERTZ[i]’ (for single tone analysis), ‘HERTZ[i][j]’ (for two-tone analysis), ‘HERTZ[i][j][k]’ (for 3-tone analysis), etc. You must specify the harmonic index integer for the HERTZ keyword. The frequency of the specified harmonics results.
INDICES	Index to tones in the form [n1, n2,..., nN], where nj is the index of the HB tone and the HB statement contains N tones. Wildcards are illegal if you use the INDICES keyword.

TYPE can be one of the following:

- Voltage type – a single node name (n1), or a pair of node names, (n1,n2)
- Current type – an element name (elemname)
- Power type – a resistor (resistorname) or port (portname) element name.

Use the following output syntax to transform HB data into the time domain:

```
.PRINT hbtran ov1 [ov2...]
.PROBE hbtran ov1 [ov2...]
```

Where `ov1 ...` are the output variables to print or probe.

Outputting Phase Noise Source as ASCII Data Files Using *.printpn0

HB phase noise and phase noise analyses can output simulation results as ASCII data in `*.printpn0` files for HBOSC and HBNOISE. Extend the E- and

G- voltage-controlled source syntax to make the phase noise data in ASCII phase noise files double as input for specifying behavioral noise sources.

Usage Model

The syntax for the voltage controlled voltage (E) or current (G) source is as follows:

```
Exxx node1 node2 noise file='filename' [mname='measname']
Gxxx node1 node2 noise file='filename' [mname='measname']
```

Where `file='filename'` is the name of the ASCII phase noise data file. The file '*design*.printpn0', name is typically designates an .HBOSC phase noise analysis or .HBNOISE analysis output file.

Use `mname='measname'` to select the appropriate noise measurement name from the `*.printpn0` 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 .HBNOISE or .SNNOISE analysis

HB Output Data Files and Examples

The results of an HB analysis are complex spectral components at each frequency point. The a[i] is the real part, and b[i] is the imaginary part of the complex voltage at frequency index i. The conversion to a steady state time-domain is then given by the Fourier series expansion.

An HB analysis produces these output data files:

- Output from the `.PRINT HB` statement is written to a `.printhb#` file.
  - The header contains the large signal fundamental frequencies.
  - The columns of data are labeled as `HERTZ`, followed by frequency indices, and then the output variable names.
  - The sum of the frequency indices, multiplied by the corresponding fundamental frequencies, add up to the frequency in the first column.

- Output from the `.PROBE HB` statement is written to a `.hb#` file. It is in the same format as the HSPICE transient analysis `.tr#` file. Besides the output waveform, it contains the information of harmonic indices and basic tone frequencies.
- Output from the `.PRINT HBTRAN` statement is written to a printer file. The format is identical to a `.print#` file.
- Output from the `.PROBE HBTRAN` statement is written to a `.hr#` file. The format is identical to a *`.tr#`* file.
- Reported performance log statistics are written to a `.lis` file.

.HB Output Example

```
.PRINT HB P(rload)       $ RMS power (spectrum)
                         $ dissipated at the rload resistor
.PROBE HB V(n1,v2)       $ Differential voltage (spectrum)
                         $ between the n1,n2 nodes
.PRINT HB VP(out)[1]     $ Phase of voltage at the out
                         $ node, at the fundamental
                         $ frequency
.PROBE HB P(Pout)[2,-1]  $ RMS power delivered to the Pout
                         $ port, at third-order intermod
.PRINT HBTRAN V(n1)      $ Voltage at n1 in time domain
.PROBE HBTRAN V(n1, n2)  $ Differential voltages between n1
                         $ and n2 node in time domain.
```

HB Error and Warning Messages

HB Analysis Error Messages

Following are the error messages issued by HSPICE simulator when using the .HB analysis:

File	Description
HB_ERR.1	Harmonic numbers must be positive non-zero.
HB_ERR.2	No .hb frequencies given.
HB_ERR.3	Negative frequency given.
HB_ERR.4	Number of harmonics should be greater than zero.
HB_ERR.5	Different number of tones, nharms.
HB_ERR.6	Bad probe node format for oscillator analysis.

File	Description
HB_ERR.7	Bad format for FSPTS.
HB_ERR.8	Bad .hb keyword.
HB_ERR.9	Tones must be specified for .hb analysis.
HB_ERR.10	Nharms or intmodmax must be specified for .hb analysis.
HB_ERR.11	Source harmonic out of range.
HB_ERR.12	Source named in the tones list is not defined.
HB_ERR.13	Source named in the tones list does not have TRANFORHB specified.
HB_ERR.14	Source named in the tones list has no transient description.
HB_ERR.15	Source named in the tones list must be HB, SIN, PULSE, PWL, or VMRF.
HB_ERR.16	Tone specification for the source is inconsistent with its frequency.
HB_ERR.17	HB oscillator analysis has reached the NULL solution.
HB_ERR.18	Bad subharms format.
HB_ERR.19	Modtone may not be set to the same value as tone.

HB Analysis Error Messages

Following are the warning messages issued by HSPICE when using the .HB analysis:

File	Description
HB_WARN.1	.hb multiply defined. Last one will be used.
HB_WARN.2	Tone specified for V/I source not specified in .HB command.
HB_WARN.3	HB convergence not achieved.
HB_WARN.4	Source specifies both HB and transient description. HB description will be used.
HB_WARN.5	Source specifies exponential decay. HB will ignore it.
HB_WARN.6	Source specifies a non-positive frequency.
HB_WARN.7	Source does not fit the HB spectrum.

File	Description
HB_WARN.8	Source cannot be used with the TRANFORHB option.
HB_WARN.9	Frequency not found from transient analysis
HB_WARN.10	.hb/.hbosc will be ignored due to .env/.envosc.
HB_WARN.11	HBTRANINIT does not support more than one input tone.

Calculating Power Measurements After HB Analyses

Two types of power measurements are available: dissipated power in resistors and delivered power to port elements. The following describes the subtle differences between these two measurements:

Power Dissipated in a Resistor

All power calculations make use of the fundamental phasor power relationship given as the following equation, where voltage V and current I are complex phasors given in peak values (not rms, nor peak-to-peak):

Equation 7

$$P_{rms} = \frac{1}{2}Re\{VI^*\}$$

In the case of a simple resistor, its current and voltage relate to each other according to $V_n = I_n R$. Equation 8 gives the power dissipated in a resistor of (real) value R at frequency index n:

Equation 8

$$P_{rms}(resistor)[n] = \frac{|V_n|^2}{2R}$$

Power Delivered to a Port Element

The port element can be either a source or sink for power. You can use a special calculation that computes the power flowing into a port element even if the port element itself is the source of that power. In Figure 7 a Port element connects to a circuit (the Port element may or may not include a voltage source).

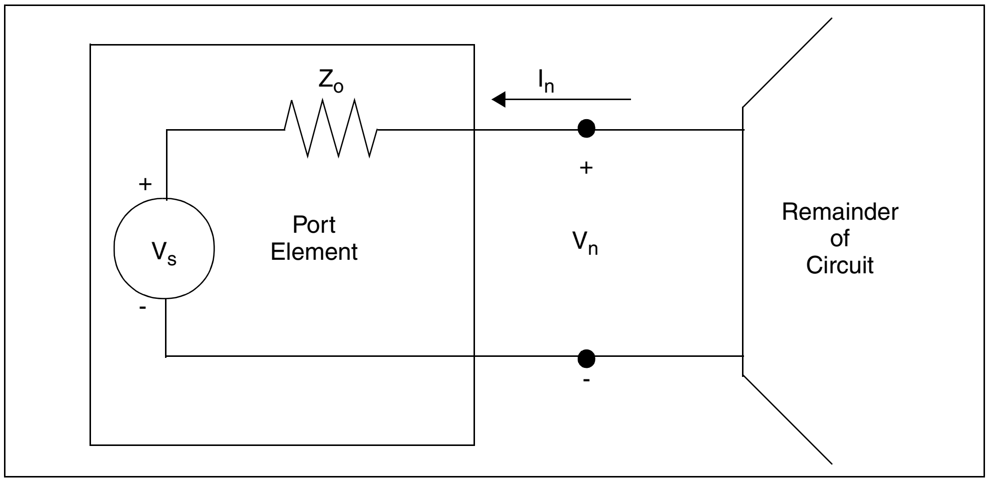


Figure 7 Port Element

Let V_n be the (peak) voltage across the terminals of the port element (at frequency index n). Let I_n be the (peak) current into the (1st) terminal of the port element (at frequency index n). Let Z_o be the impedance value of the z0 port element. Then, you can compute the power wave flowing into the terminals of the port element (at frequency index *n*) according to:

Equation 9

$$P_{in}[n] = \frac{1}{2}\left|\frac{V_n + Z_o I_n}{2\sqrt{Z_o}}\right|^2$$

This power expression remains valid whether or not the port element includes an internal voltage source at the same frequency. If the port element includes a voltage source at the same frequency, you can use this power calculation to compute the magnitude of the related large-signal scattering parameters.

If you expand the preceding formula, you can determine the power delivered to a port element with (real) impedance Z_o:

Equation 10

$$P_{rms}(port)[n] = \frac{1}{2}\left\{\frac{|V_n|^2 + Z_o^2\,|I_n|^2}{4Z_o} + \frac{1}{2}Re\{V_n I_n^*\}\right\}$$

This power value represents the power incident upon and delivered to the port element's load impedance (Zo) due to other power sources in the circuit, and due to reflections of its own generated power.

If you use the port element as a load resistor (no internal source), the preceding equation reduces to that for the simple resistor.

If you use the port element as a power source (with non-zero available power, i.e. a non-zero V_S) and it terminates in a matched load (Zo), the port power-measurement returns 0 W, because no power is reflected.

You can request power measurements in the form of complete spectra or in the form of scalar quantities that represent power at a particular element. To request a complete power spectrum, use the following syntax.

```
.PRINT HB P(Elem)
.PROBE HB P(Elem)
```

To request a power value at a particular frequency tone, use the following syntax:

```
.PRINT HB P(Elem)[<n1<,n2<n3,...>>>]
.PROBE HB P(Elem)[<n1<,n2<,n3,...>>>]
```

The `Elem` is the name of either a Resistor (R) or Port (P) element, and *n1*,*n2*, and *n3* are integer indices used for selecting a particular frequency in the Harmonic Balance output spectrum.

Example 1 Prints a table of the RMS power (spectrum) dissipated by resistor R1.

```
.PRINT HB P(R1)
```

Example 2 Outputs the RMS power dissipated by resistor R1 at the fundamental HB analysis frequency following a one-tone analysis.

```
.PROBE HB P(R1)[1]
```

Example 3 Prints the power dissipated by resistor R1 at DC following a one-tone analysis.

```
.PRINT HB P(R1)[0]
```

Example 4 Outputs the RMS power dissipated by resistor R1 at the (low-side) 3rd order intermodulation product after an HB two-tone analysis.

```
.PROBE HB P(R1)[2,-1]
```

Example 5 Prints the RMS power dissipated by resistor R1 at the (high-side) 3rd order intermodulation product after an HB two-tone analysis.

```
.PRINT HB P(R1)[-1,2]
```

Example 6 Outputs the RMS power (spectrum) delivered to port element Pload.

```
.PROBE HB P(Pload)
```

Example 7 Prints the RMS power delivered to port element Pload at the fundamental HB analysis frequency following a one-tone analysis.

```
.PRINT HB P(Pload)[1] $
```

Example 8 Outputs the RMS power delivered to port element Pload at the (low-side) 3rd order intermodulation product after an HB two-tone analysis.

```
.PROBE HB P(Pload)[2,-1]
```

Calculations for Time-Domain Output

In addition to a frequency-domain output, HB analysis also supports a time-domain output. The simulation generates an equivalent time-domain waveform according to the Fourier series expansion by way of

Equation 11

$$V(n1)\text{@time } t = \text{SUM}_{\text{OVER}\,m}(\text{REAL}\,V(n1)[m]) \bullet \cos""(\Omega[m] \bullet t) - \text{IMAG}(\text{V}(n1)[m]\sin\Omega[m \bullet t]t$$

Where *m* starts from 0 to the number of frequency points in the HB simulation.

The output syntax is

```
.PRINT [HBTRAN | HBTR] V(n1)
.PROBE [HBTRAN | HBTR] V(n1)
```

The output time ranges from 0 to twice the period of the smallest frequency in the HB spectra.

Minimizing Gibbs Phenomenon

You can use the HB_GIBBS option for HBTRAN output to minimize Gibbs' phenomenon that may occur in transforming a square-wave signal from the frequency domain to the time domain. The syntax is `.OPTION HB_GIBBS=n` (defaults to zero, which is equivalent to not using it at all). The result is that the A $(\sin c(x))^N$ function filters HBTRAN waveforms before they transform to the time domain via FFT. This option applies only to single-tone output. For example:

```
.option hb_gibbs = 2
...
.print hbtran v(2)
```

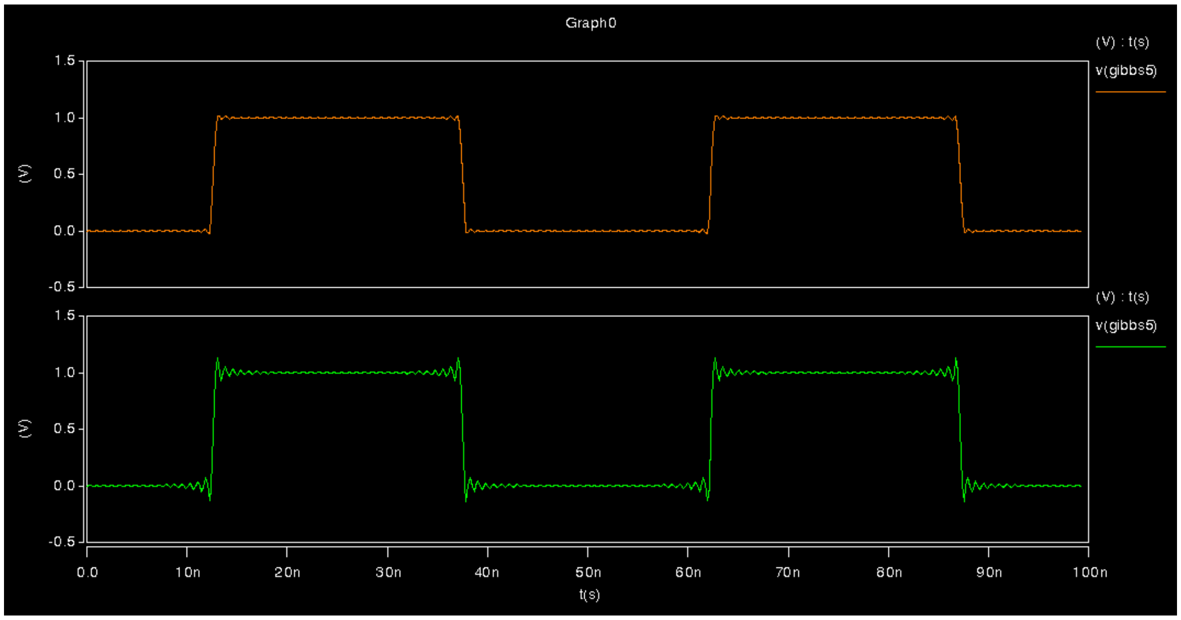


Figure 8 Upper square-wave signal shows HB_GIBBS = 2, while the lower shows the option = 0

Using .MEASURE with .HB Analyses

- For transient analysis (TRAN), the independent variable for calculating `.MEASURE` is time.
- For AC analysis, the independent variable for calculating `.MEASURE` is frequency.
- However, as with DC analysis, the use of a `.MEASURE` command is peculiar for HB analysis, because it has no obvious independent variable.

In HSPICE advanced analog analyses, the independent variable for HB `.MEASURE` analysis is the first swept variable specified in the `.HB` simulation control statement. This variable can be anything: frequency, power, voltage, current, a component value, and so on.

Example 9 *For the following `.HB` simulation control statement, the independent variable is the swept tone frequency, and the `.MEASURE` command values return results based on this frequency sweep:*

```
* HARMONIC BALANCE tone-frequency sweep for amplifier
.param freq1=1.91e9 power=1e-3
.HB tones=freq1 nharms=10 sweep freq1 LIN 10 1.91e9 2.0e9
.MEASURE HB Patf0 FIND P(Rload)[1] AT=1.95e9 $ Power at
+ f0=1.95Ghz
.MEASURE HB Frq1W WHEN P(Rload)[1]=1. $ freq1 @ 1 Watt
.MEASURE HB BW1W TRIG AT=1.92e9 TARG P(Rload)[1] VAL=1.
+ CROSS=2 $ 1 Watt bandwidth
.MEASURE HB MaxPwr MAX P(Rload)[1] FROM=1.91e9 TO=2.0e9
+ $ Finds max output power
.MEASURE HB MinPwr MIN P(Rload)[1] FROM=1.91e9 TO=2.0e9
+ $ Finds min output power
```

Example 10 *In the following example, the independent variable is the* power *variable, and the `.MEASURE` values return results based on the power sweep. Units are in Watts.*

```
* HARMONIC BALANCE power sweep for amplifier
.param freq1=1.91e9 power=1e-3
.HB tones=freq1 nharms=10 sweep power DEC 10 1e-6 1e-3
.MEASURE HB Pat1uW FIND P(Rload)[1] AT=1e-6 $ Pout at 1uW
.MEASURE HB Pin1W WHEN P(Rload)[1]=1. $ Pin @ 1 Watt Pout
.MEASURE HB Prange1W TRIG AT=1.92e9 TARG P(Rload)[1] VAL=1.
+ CROSS=2    $ 1W oper. range

.MEASURE HB ssGain DERIV P(Rload)[1] AT=1e-5
+ $ relative power gain at 10uW input
.MEASURE HB Gain3rd DERIV P(Rload)[3] AT=1e-5
+ $ 3rd harmonic gain at 10uW input
.MEASURE HB PAE1W FIND ‘(P(Rload)[1]-power)/P(Vdc)[0]’
+ WHEN P(Rload)[1]=1 $ PAE at 1 Watt output
```

Example 11 *In this example, the independent variable is again the* power *variable, and the* `.MEASURE` *values return results based on the power sweep. This is a two-tone sweep, where both input frequency sources are at the same power level in Watts.*

```
* HARMONIC BALANCE two-tone sweep for amplifier
* An IP3 calculation is made at 10uW in the sweep
.param freq1=1.91e9 freq2=1.91e9 power=1e-3
.HB tones=freq1,freq2 nharms=6,6 sweep power DEC 10 1e-6 1e-3
.MEASURE HB Pf1dBm FIND '10.*LOG(P(Rload)[1,0]/1.e-3)'
+ AT=1e-5 $ P(f1) at 10uW input
.MEASURE HB P2f1_f2dBm FIND '10.*LOG(P(Rload)[2,-1]/1.e-3)'
+ AT=1e-5 $ P(2f1-f2) at 10uW input
.MEASURE HB OIP3dBm PARAM = '0.5*(3.*Pf1dBm-P2f1_f2dBm)'
.MEASURE HB IIP3dBm PARAM = 'OIP3dBm-Pf1dBm+20.0'
.MEASURE HB AM2PM DERIV VP(outp,outn)[1] AT=1e-5
+ $ AM to PM Conversion in Deg/Watt
```

If you do not specify an HB sweep, then `.MEASURE` assumes a single-valued independent variable sweep.

You can apply the measurements to current, voltage, and power waveforms. The independent variable for measurements is the swept variable (such as power), not the frequency axis that corresponds to a single HB steady state point.

HSPICE advanced analog analyses also supports the `.MEASURE [HBTRAN | HBTR] ...` syntax. Similar to the `.PROBE` and `.PRINT HBTR` statements in the section Calculations for Time-Domain Output, you apply a `.MEASURE HBTR` statement on the signals obtained in the same way. Moreover, like a `.MEASURE` statement in transient analysis, the independent variable in a `.MEASURE HBTR` statement is time.

HSPICE advanced analog analyses optimization can read the data from `.MEASURE HB` and `.MEASURE HBTR` statements. Due to the difference in the independent variable between the `.MEASURE HB` and `.MEASURE HBTR` statements, you cannot mix these two types of measurements in an HSPICE advanced analog analyses optimization. But you can combine a `.MEASURE HBTR` statement with a `.MEASURE PHASENOISE` statement (see Measuring Phase Noise with .MEASURE PHASENOISE) and a `.MEASURE HBNOISE` statement (see Measuring HBNOISE Analyses with .MEASURE) in an HSPICE advanced analog analyses optimization flow.

Steady-State HB Sources

The fundamental frequencies used with harmonic balance analysis are specified with the `.HB TONES` command. These frequencies can then be referenced by their integer indices when specifying steady-state signal sources. For example, the .HB specification given by the following line:

.HB TONES=1900MEG,1910MEG INTMODMAX=5

This specifies two fundamental frequencies: $f[tone = 1] = 1.9GHz$ and $f[tone = 2] = 1.91GHz$. Their mixing product at 10 MHz can then be referenced using indices as $|f[2] - f[1]|$, while their 3rd order intermodulation product at 1.89 GHz can be referenced as $|2f[1] - f[2]|$.

Steady-state voltage and current sources are identified with the HB keyword according to

[HB [mag [phase [harm [tone [mod harm [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 12 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 13 *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 14 *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 12

$$A\sin(\omega t + \phi)$$

while the HB sources follow

Equation 13

$$A\cos(\omega t + \phi)$$

In order for the two sources to yield identical results it is necessary to align them by setting their phase values accordingly using:

Equation 14

$$A\cos(\omega t + \phi) = A\sin(\omega t + \phi + 90°)$$

Equation 15

$$A\sin(\omega t + \phi) = A\cos(\omega t + \phi - 90°)$$

Example 15 *To specify sources with matching phase for HB and TRAN analysis, use a convention similar to:*

```
** 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
```

Tutorial Examples Harmonic Balance Analysis

The following sections present these tutorial examples:

- Example 1: Using HB Analysis for a Power Amplifier
- Example 2: Using HB Analysis for a Low Noise Amp

Example 1: Using HB Analysis for a Power Amplifier

The `.HB` command computes periodic steady-state solutions of circuits. This analysis uses the Harmonic Balance (HB) technique for computing such solutions in the frequency domain. The circuit can be driven by a voltage, power, or current source, or it may be an autonomous oscillator. The HB algorithm represents the circuit's voltage and current waveforms as a Fourier series, that is, a series of sinusoidal waveforms.

To set up a periodic steady-state analysis, the HSPICE input netlist must contain:

- An `.HB` command to activate the analysis. The `.HB` command specifies the base frequency (or frequencies, also called tones) for the analysis, and the number of harmonics to use for each tone. The `.HB` command can specify base tones so that the circuit solution is represented as a multi-dimensional Fourier series. The number of terms in the series are determined by the number of harmonics; more harmonics result in higher accuracy, but also longer simulation times and higher memory usage.
- One or more signal sources for driving the circuit in HB analysis, if the circuit is driven. In the case of autonomous oscillator analysis, no signal source is required. Signal sources are specified using the HB keyword on the voltage or current source syntax. Power sources are specified by setting the power switch on voltage/current sources to 1; in this case, the source value is treated as a power value in Watts instead of a voltage or current.

Optionally, the netlist can also contain a set of control option for optimizing HB analysis performance.

The following example shows how to set up a Harmonic Balance analysis on an NMOS Class C Power Amplifier. The example compares transient analysis results to Harmonic Balance results.

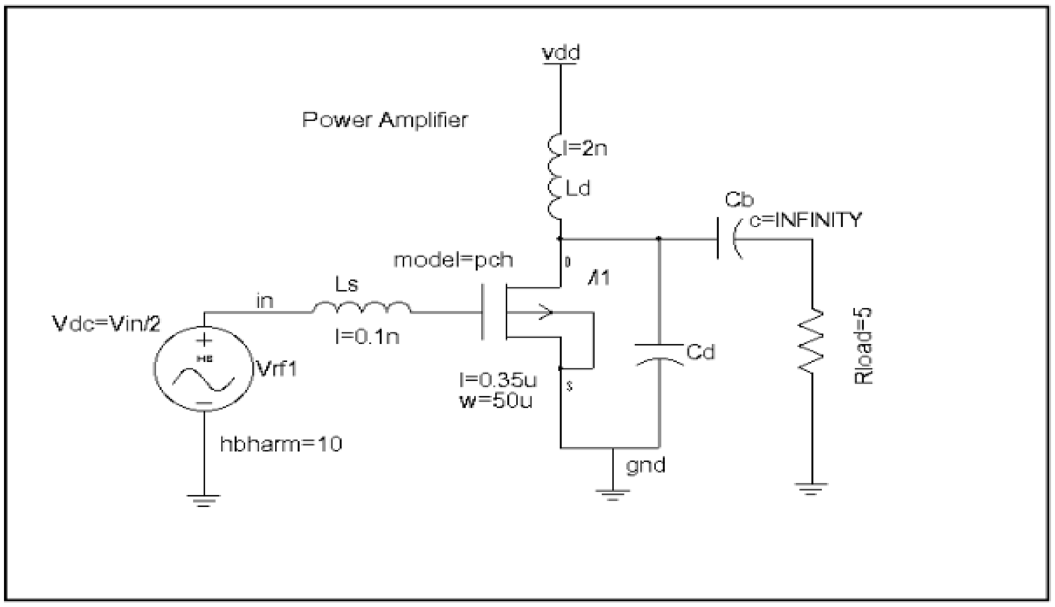


Figure 9 Power Amplifier

The following netlist performs both a transient and a Harmonic Balance analysis of the amplifier driven by a sinusoidal input waveform. The `accurate` option is set to ensure sufficient number of time points for comparison with HB. This example is included with the HSPICE advanced analog analyses distribution as `pa.sp` and is available in directory `$`*`installdir`*`/demo/hspice/rf_examples/`.

```
.options POST accurate
.param f0=950e6 PI=3.1415926 Ld=2e-9 Rload=5 Vin=3.0
.param Lin=0.1n Vdd=2 Cd='1.0/(4*PI*PI*f0*f0*Ld)'
M1 drain gt 0 0 CMOSN L=0.35u W=50u AS=100p AD=100p
PS=104u PD=104u M=80
Ls in gt   Lin $ gate tuning
Ld drain vdd Ld $ drain tuning
Cd drain 0    Cd
Cb drain out   INFINITY $ DC block
Rload out   0   Rload
Vdd vdd 0   DC    Vdd
Vrf1 in   0   DC 'Vin/2.0'
+ SIN ('Vin/2' 'Vin/2' 'f0' 0 0 90)
+ HB 'Vin/2' 0.0 1 1
.hb tones=f0 nharms=10
.tran 10p 10n
.probe hb p(Rload)
.probe tran p(Rload)
.include cmos49_model.inc
.end
```

An HB analysis uses the following:

- An `.HB` command:

  ```
  .hb tones=f0 nharms=10
  ```

  For a single tone analysis with base frequency 950 MHz and 10 harmonics.

- The HB source in Vrf1:

  ```
  HB ‘Vin/2’ 0.0 1 1.
  ```

  This creates a sinusoidal waveform matching the transient analysis one. The amplitude is Vin/2=1.5 V, and it applies to the first harmonic of the first tone, 950 MHz.

- A `.PROBE` command for plotting the output power:

  ```
  .probe hb p(Rload)
  ```

To run this netlist, type the following command:

```
hspicerf pa.sp
```

This produces two output files named `pa.tr0` and `pa.hb0`, containing the transient and HB output, respectively. To view and compare the output:

1. Type **wv** at the prompt to invoke Custom WaveView.
2. Use **File** > **Import Waveform File** and select the pa.tr0 and pa.hb0 files from the Open: Waveform Files dialog box.
3. Select the v(out) signal from the `pa.hb0` in the signal browser. Double-click on the signal name or drag and drop the signal in the waveform. The histogram shows lines at 950MHz, and multiples thereof, up to 9.5GHz.
4. In the waveform, right click in the name area of the panel containing the signal v(out), left-click on the waveform label for v(out) from the pa.hb0 file. From the **Panel** menu, choose **Signal 'v(out)'** > **To Time-Domain**.
5. In the Convert to Time Domain window, change the X-End (sec) value to 10n.
6. Click **OK** to accept the settings.
7. The new waveform shows a new time domain waveform named IFT.0|v(out).
8. Select the v(out) signal from the `pa.tr0` in the signal browser. Drag and drop the signal in the waveform containing IFT.0|v(out). This should overlay the IFT.0|v(out) and v(out) signals on the same waveform. Zoom into the transitions to see the slight differences between the waveforms.

Example 2: Using HB Analysis for a Low Noise Amp

This example performs a simulation on a Low Noise Amplifier circuit using two closely-spaced steady-state tones to study the compression and third order distortion properties of the amplifier. The example file `gsmlnaIP3_A.sp` is located at: `$`*`installdir`*`/demo/hspice/rf_examples/`.

See Figure 10 for the schematic view.

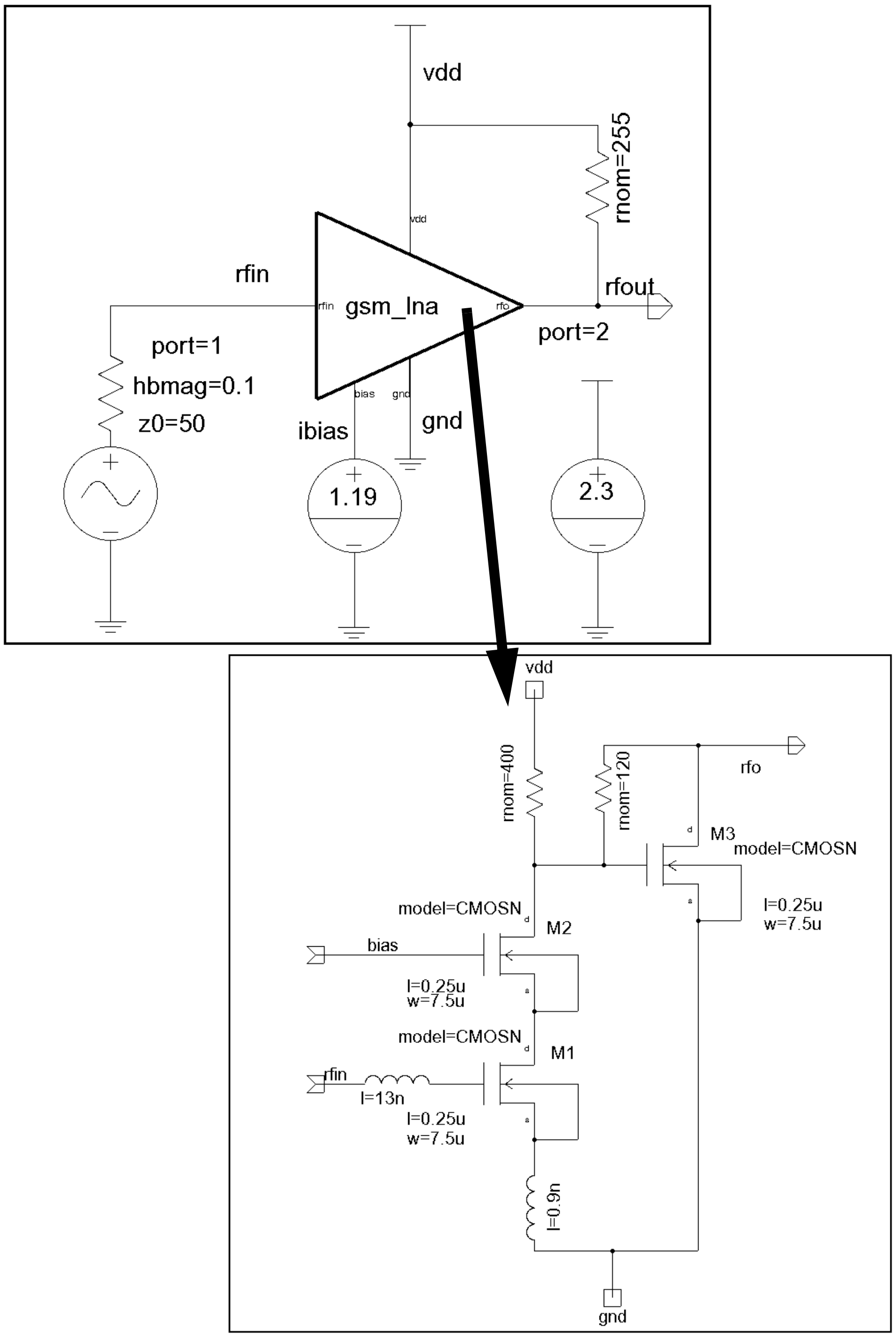


Figure 10 Schematic showing instantiation for Low Noise Amplifier

```
** NMOS 0.25um Cascode LNA for GSM applications
** setup for s-parameter and noise parameter measurements
.temp 27
.options post
.param Vdd=2.3
**
** Cascode LNA tuned for operation near 1 GHz
**
M1 _n4 _n3 _n5 _n5 CMOSN l=0.25u w=7.5u
+ as=15p ad=15p ps=19u pd=19u m=80
M2 _n6 _n1 _n4 _n4 CMOSN l=0.25u w=7.5u
+ as=15p ad=15p ps=19u pd=19u m=80
M3 rfo _n6 gnd gnd CMOSN l=0.25u w=7.5u
+ as=15p ad=15p ps=19u pd=19u m=40
r1 _vdd _n6 400
l1 _n5 gnd l=0.9nH
l2 rfin _n3 l=13nH
lchk rfin rfinb INFINITY
cblk rfin rfind INFINITY
vvb _n1 gnd dc=1.19 $ bias for common base device
vinb rfinb gnd dc=0.595
vvdd _vdd gnd dc=Vdd
rfb rfo _n6 120 $ feedback
**
**
** Two-tone input source (DC blocked at this time)
**
Vin rfind gnd dc=0 power=1 z0=50 $ 50 Ohm src
+ HB Pin 0 1 1 $ tone 1
+ HB Pin 0 1 2 $ tone 2
Rload rfo _vdd R=255
**
** HB test bench to measure IP3 and IP2
.HB tones=900MEG, 910MEG nharms=11 11 intmodmax=7
+ SWEEP Pin dec 10 1e-8 1e-3
.print HB P(Rload) P(Rload)[1,0] P(Rload) [2,0] P(Rload)[2,-1]
.probe HB P(Rload) P(Rload)[1,0] P(Rload) [2,0] P(Rload)[2,-1]
**

**
```

Figure 11 Low Noise Amplifier netlist, Part 1

```
** Approximate parameters for TSMC 0.25 Process (MOSIS run T17B)
**
.MODEL CMOSN NMOS ( LEVEL = 49
+VERSION = 3.1 TNOM = 27 TOX = 5.8E-9
+XJ = 1E-7 NCH = 2.3549E17 VTH0 = 0.3819327
+K1 = 0.477867 K2 = 2.422759E-3 K3 = 1E-3
+K3B = 2.1606637 W0 = 1E-7 NLX = 1.57986E-7
+DVT0W = 0 DVT1W = 0 DVT2W = 0
+DVT0 = 0.5334651 DVT1 = 0.7186877 DVT2 = -0.5
+U0 = 289.1720829 UA = -1.300598E-9 UB = 2.3082E-18
+UC = 2.841618E-11 VSAT = 1.482651E5 A0 = 1.6856991
+AGS = 0.2874763 B0 = -1.833193E-8 B1 = -1E-7
+KETA = -2.395348E-3 A1 = 0 A2 = 0.4177975
+RDSW = 178.7751373 PRWG = 0.3774172 PRWB = -0.2
+WR = 1 WINT = 0 LINT = 1.88839E-8
+XL = 3E-8 XW = -4E-8 DWG = -1.2139E-8
+DWB = 4.613042E-9 VOFF = -0.0981658 NFACTOR = 1.2032376
+CIT = 0 CDSC = 2.4E-4 CDSCD = 0
+CDSCB = 0 ETA0 = 5.128492E-3 ETAB = 6.18609E-4
+DSUB = 0.0463218 PCLM = 1.91946 PDIBLC1 = 1
+PDIBLC2 = 4.422611E-3 PDIBLCB = -0.1 DROUT = 0.9817908
+PSCBE1 = 7.982649E10 PSCBE2 = 5.200359E-10 PVAG = 9.31443E-3
+DELTA = 0.01 RSH = 3.7 MOBMOD = 1
+PRT = 0 UTE = -1.5 KT1 = -0.11
+KT1L = 0 KT2 = 0.022 UA1 = 4.31E-9
+UB1 = -7.61E-18 UC1 = -5.6E-11 AT = 3.3E4
+WL = 0 WLN = 1 WW = 0
+WWN = 1 WWL = 0 LL = 0
+LLN = 1 LW = 0 LWN = 1
+LWL = 0 CAPMOD = 2 XPART = 0.5
+CGDO = 5.62E-10 CGSO = 5.62E-10 CGBO = 1E-12
+CJ = 1.641005E-3 PB = 0.99 MJ = 0.4453094
+CJSW = 4.179682E-10 PBSW = 0.99 MJSW = 0.3413857
+CJSWG = 3.29E-10 PBSWG = 0.99 MJSWG = 0.3413857
+CF = 0 PVTH0 = -8.385037E-3 PRDSW = -10
+PK2 = 2.650965E-3 WKETA = 7.293869E-3 LKETA = -6.070E-3)
*
.meas hb fund1_mag max v(rfo) [1,0]
.meas hb fund2_mag max v(rfo) [0,1]
.meas hb harm1_mag max v(rfo) [2,-1]
.meas hb harm2_mag max v(rfo) [-1,2]
.meas hb fund1 param ='20*log10(fund1_mag)'
.meas hb fund2 param ='20*log10(fund2_mag)'
.meas hb harm1 param ='20*log10(harm1_mag)'
.meas hb harm2 param ='20*log10(harm2_mag)'
.meas hb IIP31 param ='fund1+(fund1-harm1)/2'
.meas hb IIP32 param ='fund1+(fund2-harm2)/2'
*
.END
```

Figure 12 Low Noise Amplifier netlist, Part 2

First, a voltage source element is used as a two-tone power source by setting the power flag and a source impedance of 50 ohms is specified. The `HB` keyword is used to identify the amplitude (interpreted as Watts with the power flag set to 1), phase, harmonic index, and tone index for each tone.

```
Vin rfind gnd dc=0 power=1 z0=50 $ 50 Ohm src
+ HB Pin 0 1 1 $ tone 1
+ HB Pin 0 1 2 $ tone 2
```

Second, the `.HB` command designates the frequencies of the two tones and establishes the power sweep. The `intmodmax` parameter has been set to 7 to include intermodulation harmonic content up to 7th order effects.

```
.HB tones=900MEG,910MEG nharms=11 intmodmax=7
+ SWEEP Pin dec 10 1e-8 1e-3
```

Last, the HSPICE advanced analog analyses ability to specify a specific harmonic term is used in the `.PRINT` and `.PROBE` statements to pull out the signals of particular interest. Notice the three different formats:

1. The following reference dumps a complete spectrum in RMS Watts for the power across resistor `Rload`.

```
.PRINT HB P(Rload)
```

2. The following reference selectively dumps the power in resistor `Rload` at the first harmonic of the 1st tone.

```
.PRINT HB P(Rload)[1,0]
```

3. The following reference selectively dumps the power in resistor `Rload` at the 3rd intermodulation product frequency (890 MHz).

```
.PRINT HB P(Rload)[2,-1]
```

To run this simulation, type the following at the command line:

```
hspice -i gsmlnaIP3_A.sp -o
```

Viewing Results using Custom WaveView

For this analysis, the `.PRINT` statement will generate a `gsmlnaIP3_A.printhb0` file. Assume you want to find the output power through the load resistor at the first tone, when the input power is 0.1mW.

To view the file:

1. Type **wv** *at the prompt* to invoke Custom WaveView.
2. Use **File** > **Import Waveform File** and select the `gsmlnaIP3_A.hb0` file.

3. Select the signal `Pr(rload) [1,0]` in the signal browser. Drag and drop the signal in the waveform. The X-axis is the input power and the Y-axis is the output power. The output power on the Y-axis is displayed in dBm but the input power on the X-axis is displayed in watts. To make the 1dB compression point measurement, it is necessary to change the X-axis scale to dBm.
4. To change the X-axis scale to dBm, right-click the X-axis and select **Attributes** from the menu. At the bottom right of the X-axis Attributes menu, select **dBm10** from the type pull-down menu and click **Apply**. Close the menu by clicking **Close**.
5. To measure the 1dB compression point of the amplifier, select **Measurement** from the tools menu. In the Measurement Tools window, the available measurements are listed by category on the left side of the window. Scroll down to the RF measurements and select **P1dB**. Click **OK**; this places a dynamic meter in the waveform.
6. Move the dynamic meter near the signal, the meter will show the 1dB compression point, the linear gain of the amplifier, and the input power where the 1dB compression point is measured. For best results, the asymptotic line drawn by the dynamic meter should overlay the linear portion of the amplifier power curve.

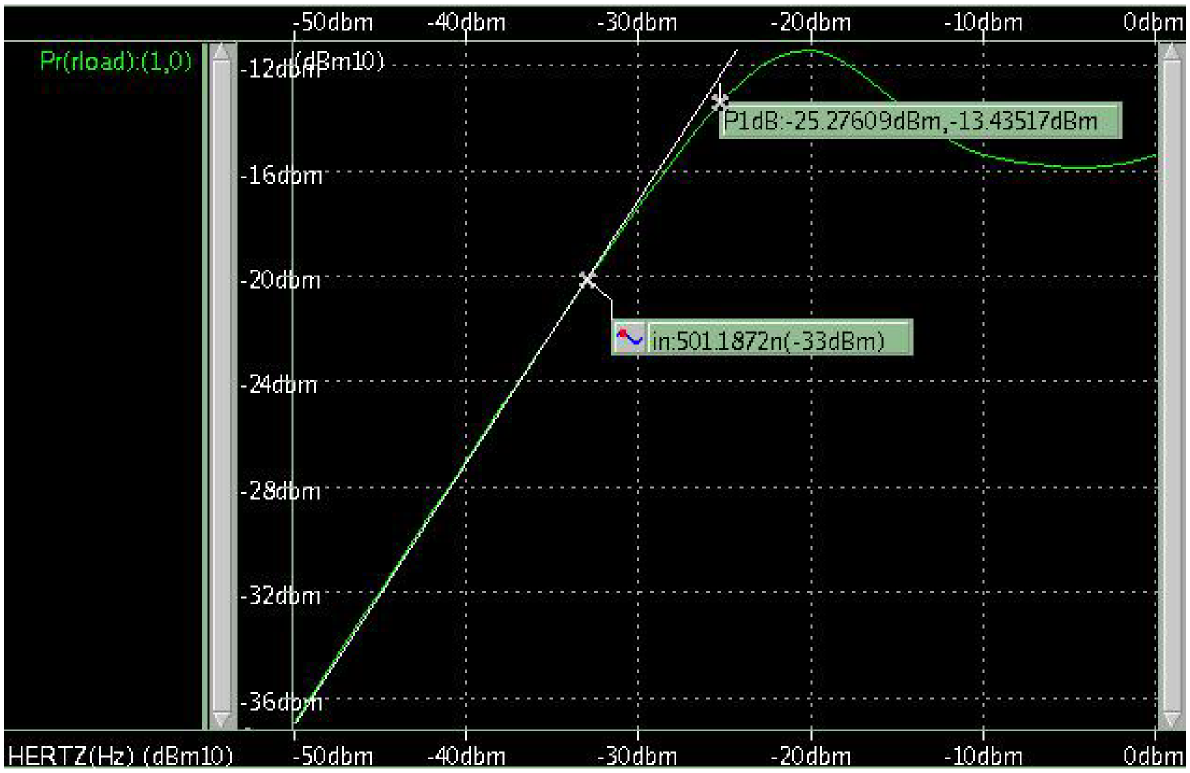


Figure 13 1dB Compression Point

7. Use **Waveview** > **New** to open a new waveform.
8. Select the signals `Pr(rload)[1,0]`, `Pr(rload)[2,-1]`, and `Pr(rload)[2,0]` in the signal browser. Drag and drop the signals in the waveform. Change the X-axis scale to dBm as described in step 4.
9. The 3rd order intercept point is also measured by using the measurement tool. In the measurement section, select **IP3/SFDR**. Click **OK**; this will place a dynamic meter in the waveform.
10. Move the dynamic meter near the `Pr(rload)[1,0]` signal, the meter will show the 3rd order intercept point and the input power where the 3rd order intercept point is measured. For best results, the asymptotic lines drawn by the dynamic meter should overlay the linear portion of the `Pr(rload)[1,0]` and `Pr(rload)[2,-1]` signals.

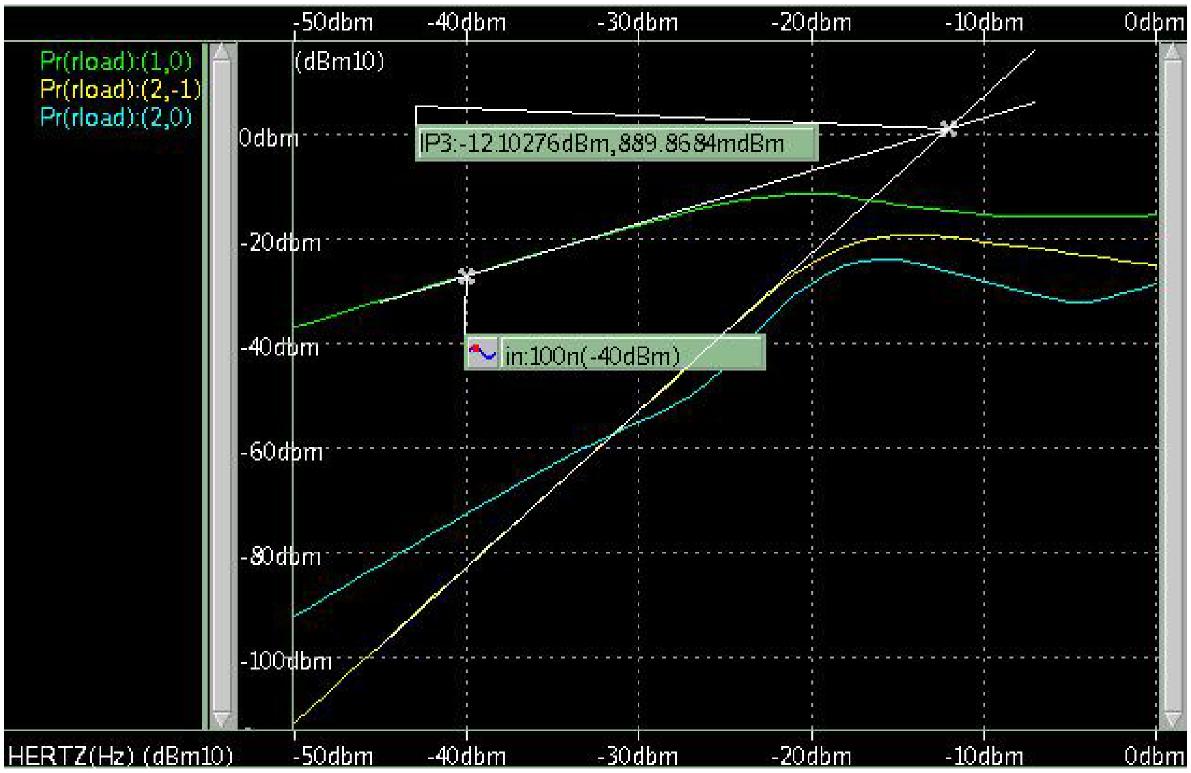


Figure 14 3rd Order Intercept Point

Device Model Cards

The following is an NMOS model in `cmos49_model.inc` file used in the power amplifier example. It is available in directory `$installdir/demo/hspice/rf_examples`.

```
**
** NMOS IC Quadrature VCO circuit for GPS local oscillator

**
** Twin differential negative resistance VCOs
** using NMOS transistors for varactors, coupled
** to produce quadrature resonances.
** Design based on 0.35um CMOS process.
**
** References:
**  >P. Vancorenland and M.S.J. Steyaert, "A 1.57-GHz fully
**    integrated very low-phase-noise quadrature VCO,"
**    IEEE Trans. Solid-State Circuits, May 2002, pp.653-656.
**  >J. van der Tang, P. van de Ven, D. Kasperkovitz, and A.
Roermund,
**   "Analysis and design of an optimally coupled 5-GHz quadrature
**    LC oscillator,"  IEEE Trans. Solid-State Circuits, May 2002,
**     pp.657-661.
** >F. Behbahani, H. Firouzkouhi, R. Chokkalingam, S. Delshadpour,
**    A. Kheirkhani, M. Nariman, M. Conta, and S. Bhatia,

**   "A fully integrated low-IF CMOS GPS radio with on-chip analog
**  image rejection," IEEE Trans. Solid-State Circuits, Dec. 2002,
**    pp. 1721-1727.
**
** Setup for Harmonic Balance Analysis
**
** Oscillation Frequency: ~ 1575 MHz (GPS L1 frequency)
** Amplitude:  ~5 Volts peak-to-peak (zero to 5V)
** Vdd: 2.5 V
**
**
** Simulation Options :
.option POST
**
.param Vtune=2.0  $ Failures: vtune=1
.param Cval=0.2p
*---------------------------------
Vtune vc gnd  DC Vtune
Vdd vdd gnd 2.5
*---------------------------------
* First oscillator section
** Low-Q resonator with Vdd at center tap of inductors
R1a IP ri 100k  $ These R's set the Q
R1b ri IN 100k
L1 IP vdd 16.5nH
L2 vdd IN 16.5nH
Cc1 IP gnd Cval $ I to Q
Cc2 IN gnd Cval $ -I to Q
** Differential fets
M1 IP IN cs gnd NMOS l=0.35u w=15u
M2 IN IP cs gnd NMOS l=0.35u w=15u
** Bias fet - bias at Vdd -- too high?
Mb cs  vdd gnd gnd NMOS l=0.35u w=15u
** fets used as varactors
Mt1 vc IP vc gnd NMOS l=0.35u w=2u M=50
Mt2 vc IN vc gnd NMOS l=0.35u w=2u M=50
*---------------------------------
** Second oscillator section
** Low-Q resonator with Vdd at center tap of inductors
R1a_b QP ri_b 100k  $ These R's set the Q
R1b_b ri_b QN 100k
L1_b QP vdd 16.5nH
```

```
L2_b vdd QN 16.5nH
Cc1_b QP gnd Cval $ -Q to -I
Cc2_b QN gnd Cval $ -Q to I
** Differential fets
M1_b QP QN cs_b gnd NMOS l=0.35u w=15u
M2_b QN QP cs_b gnd NMOS l=0.35u w=15u
** Bias fet - bias at Vdd -- too high? 2nd in parallel
Mb_b cs_b  vdd gnd gnd NMOS l=0.35u w=15u
** fets used as varactors

Mt1_b vc QP vc gnd NMOS l=0.35u w=2u M=50
Mt2_b vc QN vc gnd NMOS l=0.35u w=2u M=50
*
*-------------------------------
* Differentiators Coupling transistors for quadrature
*
.param Cdiff=0.14p difMsize=50u
vidiff  dbias gnd 1.25
viqdiff vdcdif gnd 1.75
Midiff1 dQP dbias gnd gnd NMOS l=0.35u w=difMsize
Midiff2 dQN dbias gnd gnd NMOS l=0.35u w=difMsize
Midiff3 dIN dbias gnd gnd NMOS l=0.35u w=difMsize
Midiff4 dIP dbias gnd gnd NMOS l=0.35u w=difMsize
Cdiff1  dQP QP Cdiff
Cdiff2  dQN QN Cdiff
Cdiff3  dIN IN   Cdiff
Cdiff4  dIP IP   Cdiff
Mc_QP1 IP vdcdif dQP gnd NMOS l=0.35u w=difMsize
Mc_QN2 IN vdcdif dQN gnd NMOS l=0.35u w=difMsize
Mc_QN3 QP vdcdif dIN gnd NMOS l=0.35u w=difMsize
Mc_QP4 QN vdcdif dIP gnd NMOS l=0.35u w=difMsize
*-------------------------------
* Transient Analysis Test Bench

* Use to show oscillator start up
* 2mA pulse used to start oscillator
*iosc IP IN PULSE ( 0 2m .01n .01n .01n 10n 1u )
*.probe tran v(IP) v(IN)
*.print tran v(IP) v(IN)
*.TRAN .01n 10n
*------------------------------
* Harmonic Balance Test Bench
*
.sweepblock vtune_sweep

+ 0 5 0.2
+ 2 3 0.1
.HBOSC tones=1550e6 nharms=12
```

```
+ PROBENODE=IP,QN,4
+ sweep Vtune sweepblock=vtune_sweep
**
.phasenoise dec 10 100 1e7
.print phasenoise phnz
.probe phasenoise phnz
.print hb v(IP,IN) v(IP,IN)[1] v(QP,QN) v(QP,QN)[1]
.probe hb v(IP,IN) v(IP,IN)[1] v(QP,QN) v(QP,QN)[1]
.probe hb hertz[1][1]
*
* NMOS Device from MOSIS 0.35um Process
*
* BSIM3 VERSION 3.1 PARAMETERS
*
* DATE: Mar  8/00
* LOT: n9co                  WAF: 07
* Temperature_parameters=Default
*
.MODEL NMOS NMOS (                              LEVEL   = 49
+VERSION = 3.1            TNOM    = 27          TOX = 7.9E-9
+XJ      = 1.5E-7        NCH     = 1.7E17        VTH0 = 0.5047781
+K1     = 0.5719698     K2      = 0.0197928     K3      = 33.4446099
+K3B    = -3.1667861    W0      = 1E-5          NLX     = 2.455237E-7
+DVT0W  = 0              DVT1W   = 0              DVT2W   = 0
+DVT0   = 2.8937881     DVT1    = 0.6610934     DVT2    = -0.0446083
+U0    = 421.8714618   UA     = -1.18967E-10  UB     = 1.621684E-18
+UC     = 3.422111E-11  VSAT   = 1.145012E5    A0      = 1.119634
+AGS     = 0.1918651     B0      = 1.800933E-6    B1      = 5E-6
+KETA    = 3.313177E-3    A1      = 0              A2      = 1
+RDSW   = 984.149934    PRWG   = -1.133763E-3  PRWB   = -7.19717E-3
+WR     = 1            WINT   = 9.590106E-8   LINT    = 1.719803E-8
+XL     = -5E-8         XW     = 0             DWG    = -2.019736E-9
+DWB     = 6.217095E-9    VOFF    = -0.1076921     NFACTOR = 0
+CIT     = 0              CDSC    = 2.4E-4         CDSCD   = 0
+CDSCB  = 0            ETA0   = 0.0147171     ETAB   = -7.256296E-3
+DSUB   = 0.3377074     PCLM   = 1.1535622     PDIBLC1 = 2.946624E-4
+PDIBLC2 = 4.171891E-3   PDIBLCB = 0.0497942    DROUT  = 0.0799917
+PSCBE1  = 3.380501E9    PSCBE2  = 1.69587E-9    PVAG   = 0.4105571
+DELTA   = 0.01           MOBMOD  = 1              PRT     = 0
+UTE     = -1.5           KT1     = -0.11          KT1L    = 0
+KT2    = 0.022         UA1    = 4.31E-9       UB1    = -7.61E-18
+UC1     = -5.6E-11       AT      = 3.3E4          WL      = 0
+WLN     = 1             WW      = -1.22182E-15   WWN     = 1.1657
+WWL     = 0              LL      = 0              LLN     = 1
+LW      = 0              LWN     = 1              LWL     = 0
+CAPMOD  = 2             XPART   = 0.4           CGDO    = 3.73E-10
+CGSO   = 3.73E-10      CGBO    = 1E-11         CJ     = 8.988141E-4
+PB    = 0.8616985     MJ     = 0.3906381     CJSW   = 2.463277E-10
+PBSW   = 0.5072799    MJSW   = 0.1331717    PVTH0  = -0.0143809
+PRDSW  = -81.683425   WRDSW  = -107.8071189  PK2    = 1.210197E-3
+WKETA  = -1.00008E-3  LKETA  = -6.1699E-3    PAGS   = 0.24968
+AF      = 1.0           KF      = 1.0E-30     )
*
.END
```

The following is the BJT model file, `bjt.inc` used in oscillator example. It is available in directory `$`*`installdir`*`/demo/hspice/rf_examples/`.

```
* RF Wideband NPN Transistor die SPICE MODEL
.MODEL RF_WB_NPN   NPN
+ IS    = 1.32873E-015   BF    = 1.02000E+002
+ NF    = 1.00025E+000   VAF   = 5.19033E+001
+ EG    = 1.11000E+000   XTI   = 3.00000E+000
+ CJE   = 2.03216E-012   VJE   = 6.00000E-001
+ MJE   = 2.90076E-001   TF    = 6.55790E-012
+ XTF   = 3.89752E+001   VTF   = 1.09308E+001
+ ITF   = 5.21078E-001   CJC   = 1.00353E-012
+ VJC   = 3.40808E-001   MJC   = 1.94223E-001
```

References

[1] S. Maas, *Nonlinear Microwave Circuits*, Chapter 3, IEEE Press, 1997.

[2] R. Gilmore and M.B. Steer, “Nonlinear Circuit Analysis Using the Method of Harmonic Balance - A Review of the Art, Part I, Introductory Concepts.” *International Journal of Microwave and Millimeter-wave Computer-Aided Engineering*, Volume 1, No. 1, pages 22-37, 1991.

[3] R. Gilmore and M.B. Steer, “Nonlinear Circuit Analysis Using the Method of Harmonic Balance - A Review of the Art. Part II. Advanced Concepts.” *International Journal of Microwave and Millimeter-wave Computer-Aided Engineering*, Volume 1, No. 2, pages 159-180, 1991.

[4] V. Rizzoli, F. Mastri, F. Sgallari, G. Spaletta, “Harmonic-Balance Simulation of Strongly Nonlinear Very Large-Size Microwave Circuits by Inexact Newton Methods,” *MTT-S Digest*, pages 1357-1360, 1996.

[5] S. Skaggs, *Efficient Harmonic Balance Modeling of Large Microwave Circuits*, Ph.D. thesis, North Carolina State University, 1999.

[6] R.S. Carson, *High-Frequency Amplifiers*, 2nd Edition, John Wiley & Sons, 1982

[7] S.Y. Liao, *Microwave Circuit Analysis and Amplifier Design*, Prentice-Hall, 1987.

[8] J. Roychowdhury, D. Long, P. Feldmann, “Cyclostationary Noise Analysis of Large RF Circuits with Multitone Excitations”, *IEEE JSCC*, volume 33, number 3, March 1998.

[9] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, 1995.

[10] J. Roychowdhury, D. Long, and P. Feldmann, “Cyclostationary Noise Analysis of Large RF Circuits with Multitone Excitations,” *IEEE Journal of Solid-State Circuits*, volume 33, pages 324–336, March 1998.

[11] K. Kurakawa, “Power waves and the Scattering Matrix,” IEEE Trans. *Microwave Theory Tech*., vol. MTT-13, pp. 194-202, March 1965.

3

Steady-State Shooting Newton Analysis

Describes HSPICE advanced analog analyses steady-state time domain analysis based on a Shooting-Newton algorithm.

These topics are covered in the following sections:

- Shooting Newton Steady-State Time Domain Analysis (.SN)
- Shooting Newton with Fourier Transform (.SNFT)
- Shooting Newton Analysis — Tutorial Example

Shooting Newton Steady-State Time Domain Analysis (.SN)

An advanced Shooting Newton (SN) algorithm provides additional performance and functionality to HSPICE advanced analog analyses for time-domain, steady-state analysis.

Shooting-Newton adds analysis capabilities for PLL components, digital circuits/logic, such as ring oscillators, frequency dividers, phase/frequency detectors (PFDs), switched capacitor filters, and for other digital logic circuits and RF components that require steady-state analysis, but operate with waveforms that are more square wave than sinusoidal.

For more information on control options, see .SN command in the *HSPICE Reference Manual: Commands and Control Options*.

The Shooting-Newton algorithm effectively analyzes applications including:

- Ring oscillators (see Chapter 4, Oscillator and Phase Noise Analysis)
- Frequency dividers (prescalers)

- Mixer conversion gain
- Phase-frequency detectors (PFDs)
- Mixer noise figure

Functionality includes:

- Both driven and oscillator (autonomous) analyses
- Time Domain or Frequency analysis based on advanced Shooting Newton algorithm
- Shooting Newton with Fourier Transform (.SNFT)
- Shooting Newton AC Analysis (.SNAC)
- Shooting Newton Oscillator Analysis(.SNOSC)
- Oscillator and Phase Noise Analysis

This section covers the following topics:

- SN .PRINT and .PROBE Output Syntax

SN .PRINT and .PROBE Output Syntax

The output from .SN analysis is generated in both time and frequency domains.

The time domain output variables are the same as for standard transient analysis:

- individual nodal voltages: `V(`*`n1`*` [,`*`n2`*`])`
- branch currents: `I(`*`Vxx`*`)`
- element power dissipation: `In(`*`element`*`)`

It is also possible to output the results from Shooting Newton analysis in terms of complex, frequency-domain output variables. This output format is activated by using the “SNFD” keyword in the output syntax.

For output in the frequency domain, the syntax is identical to the Harmonic Balance output syntax:

```
.PRINT SNFD TYPE (NODES|ELEMENTS)[INDICES]
```

```
.PROBE SNFD TYPE (NODES|ELEMENTS)[INDICES]
```

Parameter	Description
SNFD TYPE	Specifies a harmonic type node or element. TYPE can be one of the following: ■ Voltage type: V = voltage magnitude and phase in degrees VR = real component VI = imaginary component VM = magnitude VP - Phase in degrees VPD - Phase in degrees VPR - Phase in radians VDB - dB units VDBM - dB relative to 1 mV ■ Current type: I = current magnitude and phase in degrees IR = real component II = imaginary component IM = magnitude IP - Phase in degrees IPD - Phase in degrees IPR - Phase in radians IDB - dB units IDBM - dB relative to 1 mV ■ Power type – P ■ Frequency type: hertz[index], hertz[index1, index2, ...]. You must specify the harmonic index for the hertz variable. The frequency of the specified harmonics is dumped.
INDICES	Is the harmonic index of the SNFD tone. Index is limited to the single tone associated with the SN analysis.

SNFD TYPE can be one of the following:

- Voltage type – a single node name (n1), or a pair of node names, (n1,n2)
- Current type – an element name (elemname)
- Power type – a resistor (resistorname) or port (portname) element name.

Output Files

The time domain data are output to `printsn0` and `.sn0` files. Frequency domain data are output to .`printsnf0` and .`snf0` files.

Output Format

The format for time domain output is the same as standard transient analysis. For frequency domain output, the format is similar to HB. The main difference is that Shooting Newton output in the frequency domain is single tone only.

The results of an SN analysis are complex spectral components at each frequency point. The a[i] is the real part, and b[i] is the imaginary part of the complex voltage at frequency index i. The conversion to a steady state time-domain is then given by the Fourier series expansion.

An SN analysis produces these output data files:

- Output from the `.PRINT SN` statement is written to a `.printsn#` file.
  - The header contains the large signal fundamental frequencies.
  - The columns of data are labeled as `HERTZ`, followed by frequency indices, and then the output variable names.
  - The sum of the frequency indices, multiplied by the corresponding fundamental frequencies, add up to the frequency in the first column.
- Output from the `.PROBE SN` statement is written to a `.sn#` file in the same format as the HSPICE transient analysis .tr# file. It contains the information of harmonic indices and basic tone frequencies plus the output waveform.
- Reported performance log statistics are written to a `.lis` file:
  - Name of SN data file.
  - Simulation time:

    DC operating point (op) time

    SN time

    Total simulation time
  - Memory used
  - Size of matrix (nodes * harmonics)
  - Final SN residual error

Shooting Newton with Fourier Transform (.SNFT)

The .SNFT command is to the .SN analysis what .FFT is to the TRAN analysis, a means to provide spectrum analysis. Spectrum analysis represents a time-domain signal, within the frequency domain. .SNFT uses the Fourier transform: a Discrete Fourier Transform (DFT) uses sequences of time values to determine the frequency content of analog signals, in circuit simulation. The `.SNFT` statement uses the internal time point values.

By default, the `.SNFT` statement uses a second-order interpolation to obtain waveform samples, based on the number of points that you specify.

Note: True distributed components (such as ideal delays or transmission lines) are not supported; components with hidden states are not supported.

.SN analysis assumes that all stimuli are periodic with period T. If the circuit is driven with more than one periodic stimulus, then the frequencies must be all co-periodic and T must match the common period or some integer multiple of it. The .SN analysis only supports .tran (time-domain) periodic signal sources. (Refer to the .tran analysis for a detailed documentation on transient signal sources).

You can use windowing functions to reduce the effects of waveform truncation on the spectral content. You can also use the `.SNFT` command to specify:

- output format
- frequency
- number of harmonics
- total harmonic distortion (THD)

For more information, see .SNFT command in the *HSPICE Reference Manual: Commands and Control Options*

This section covers the following topics:

- Other Shooting Newton Analyses
- SN Error and Warning Messages

Other Shooting Newton Analyses

The following Shooting Newton Analyses are also supported by HSPICE advanced analog analyses.

- `.SNFT` is equivalent to the `.FFT` command in transient (`.TRAN`) analysis. `.SNFT` uses Fourier transform to represent a time domain signal in the frequency domain. For more information, see Shooting Newton with Fourier

Transform (.SNFT).

- `.SNAC` is used to perform a linear analysis of a driven (or non-autonomous) circuit, where the linear coefficients are modulated by a periodic, steady-state signal. The functionality is similar to the `.HBAC` command. For more information, see Shooting Newton AC Analysis (.SNAC).
- `.SNXF` is used to calculate transfer functions from an arbitrary number of small signal sources to a designated output in a circuit under periodic steady state conditions. For more information, see Shooting Newton Transfer Function Analysis (.SNXF).

SN Error and Warning Messages

Error messages are displayed with convergence recommendations in cases of non-convergence within the maximum number of Shooting-Newton iterations.

Error messages are displayed for software errors such as segmentation violations, and abort conditions such as:

- unrecognized format, i.e., unrecognized V/I source.
- faulty input values, i.e., wrong sign, out of range value.
- unspecified values, i.e., unspecified tone.
- inconsistent values, i.e., non-commensurable tones.
- duplicate values, i.e., same entry, given more than one, the last one is always taken.

Shooting Newton Analysis — Tutorial Example

While the Harmonic Balance (HB) algorithm represents the circuit's voltage and current waveforms as a Fourier series (a series of sinusoidal waveforms), the Shooting Newton (SN) algorithm provides analysis capability for digital logic circuits and advanced analog components that require steady-state analysis, but operate with waveforms that tend to be square instead of sinusoidal.

This section covers the following topics:

- Shooting Newton — Analysis Setup
- Driven Phase Frequency Detector Example
- Ring Oscillator Example

Shooting Newton — Analysis Setup

To set up a time-domain, steady-state analysis, the HSPICE input netlist must contain:

- A `.SN` command to activate the analysis. The `.SN` command specifies:
- The expected period of the steady-state waveforms, which must match the period of any input waveforms. The period is specified in time domain units (seconds). Alternatively, you may specify a frequency in Hz.
- A time resolution, which is analogous to the transient analysis (`.TRAN`) command's TSTEP parameter and will affect the time step size selection. It also affects the number of frequency terms used in small-signal analyses, such as periodic AC or noise analysis. The time resolution is typically specified in seconds but, alternatively, may be specified in the frequency domain as a number of harmonics.
- A transient initialization time that is used by HSPICE advanced analog analyses to run a basic transient simulation of this length before attempting Newton-Raphson iterations to converge on a steady-state solution. This parameter is optional. If it is not specified, the specified period is used as the initialization time. The initial transient analysis is used for circuit stabilization before the steady state solution is found. Larger initialization values typically result in convergence that is more robust.
- For oscillator circuits, a `.SNOSC` command is used to activate the analysis. The `.SNOSC` command specifies:
- The approximate frequency of oscillation specified either as a frequency (in Hertz) or as the time domain period.
- The number of high frequency harmonics. Alternatively, a time resolution in seconds can be specified.
- A transient initialization time that is used by HSPICE advanced analog analyses to run a basic transient simulation of this length before attempting Newton-Raphson iterations to converge on a steady state solution. This parameter is optional. If it is not specified, the period of the specified

frequency of oscillation is used as the initialization time. For oscillators we recommend specifying a transient initialization time since the default initialization time is usually too short to effectively stabilize the circuit.

- A node at which to probe for oscillation conditions.
- If the tuning curve of a VCO is to be analyzed, the optional parameter MAXTRINITCYCLES can be specified.
- One or more signal sources for driving the circuit in SN analysis, if the circuit is driven. In the case of autonomous oscillator analysis, no signal source is required. The sources are required to be time domain sources and must match the period specified in the `.SN` command.

Driven Phase Frequency Detector Example

This example demonstrates the Shooting Newton-based analysis of a driven phase-frequency detector. Extracted portions of the input file are presented below. The complete `phasefreqdet.sp` input file for this example is located at *`$installdir`*`/demo/hspice/rf_examples/`.

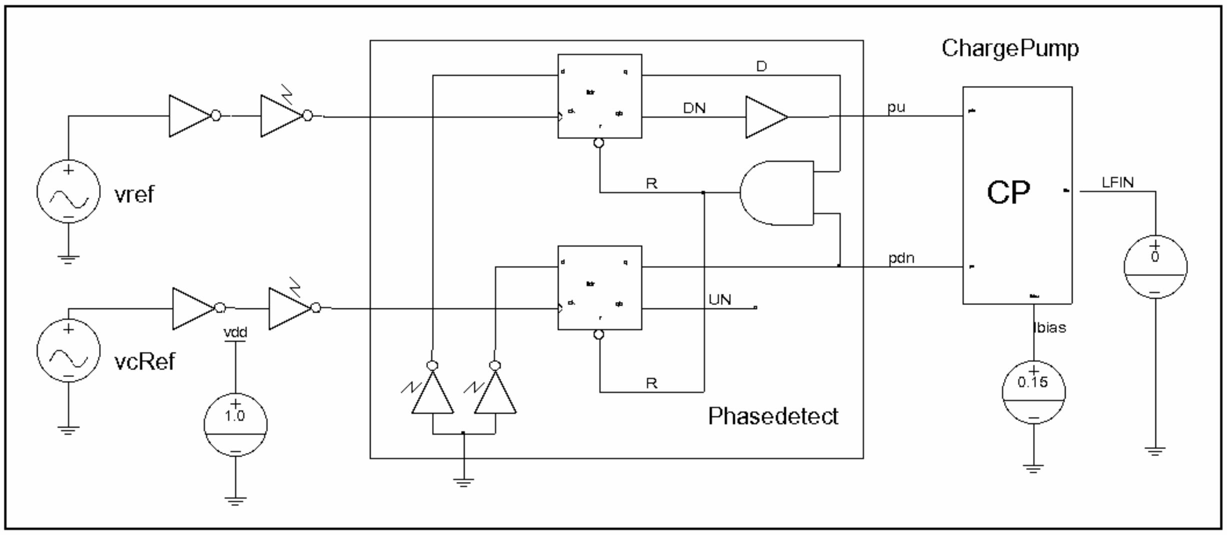


Figure 15   Driven Phase Frequency Detector

```
* Phase Frequency Detector Example
*
.global vdd gnd
.options wl post
```

```
* DC sources
vsup vdd 0 DC 1.0

* Reference signal (sine wave)
vref xin gnd DC 0 sin( 0.5 0.5 0.5g)
* Input buffers (square up Ref sine wave)
xfin1 xin fin1 inv
xfin2 fin1 FIN inv3

* Compare signal (sine wave)
vcRef cin gnd DC 0 sin( 0.5 0.5 0.5g 0.0 0.0 phase)
$ phase shift
* Input buffers (square up compare sine wave)
xcfin1 cin cfin1 inv
xcfin2 cfin1 cFIN inv3
*
** Phase/frequency detector
xPFD cFIN FIN pdn pu phasedet
** Chargepump
xCP LFIN Ibias pdn pu chargepump
** Bias voltage
vIbias Ibias gnd 0.15     $ Sets charge pump bias!
Vload LFIN 0 0

* Harmonic Balance Test Bench
*
.param phase=0.0     $ phase shift in degrees

.opt snaccuracy=30
.SN tres=10p period=2n SWEEP phase POI 5 0.0 22.5 45.0 67.5 90.0
.SNNOISE V(pu,pdn) Vref
+ DEC 21 100 10MEG      $ offset frequency sweep
+ [0,1]                 $ Take low frequency noise
*
.probe sn v(fin) v(cfin) v(pu) v(pdn) v(lfin) i(vibias)
.print snfd i(vload) i(vload) [0]
.probe snfd i(vload) i(vload) [0]
.probe SNNOISE  ONOISE
.print SNNOISE  ONOISE
.end
```

During the analysis, the phase of the input signal is swept between 0 degrees and 90 degrees using five equally spaced steps. This enables you to measure the phase detector gain at the output load. The `.SN` command syntax specifies the expected period of the steady-state waveforms (2nS) and the time resolution (10pS) in the time domain.

A periodic, time-varying AC noise analysis based on the Shooting Newton algorithm is performed using the `.SNNOISE` command. The `.SNNOISE` analysis requires an output node (v(pu, pdn)) where the noise is to be measured, an input noise source (Vref) which serves as the reference for the noise computation and, a frequency sweep for the noise analysis. Optionally, an index term can be defined. The index term specifies the output frequency band at which the noise is evaluated. For this case, you will evaluate the low frequency noise of the phase frequency detector.

The time-domain signals v(cfin), v(fin), v(pu) and v(pdn), and v(lfin) are probed. The gain of the phase frequency detector can be found by probing the frequency domain value of v(lfin) at DC (frequency indices 0).

During the simulation, the simulation status is displayed on the screen. In addition to the screen display, more detailed status, cpu time, and memory usage information is also written to the `phasefreqdet.lis` file.

Viewing Results in Custom WaveView

You can view the time-domain, `phasefreqdet.sn0` file, the frequency domain, `phasefreqdet.snf0` file, and the noise results, `phasefreqdet.snpn0` file in Custom WaveView:

1. Enter **wv** *at the prompt* to start Custom WaveView.
2. The time domain results are used to show the input and output waveforms of the phase frequency detector. Use File > Import Waveform File to open the `phasefreqdet.sn0` file.
3. Select the input signals, v(cfin) and v(fin), and the output signals, v(pu) and v(pdn) from the signal browser. Drag and drop the selected signals in the waveform. Figure 16 on page 71 shows the waveforms for the selected input and output signals.

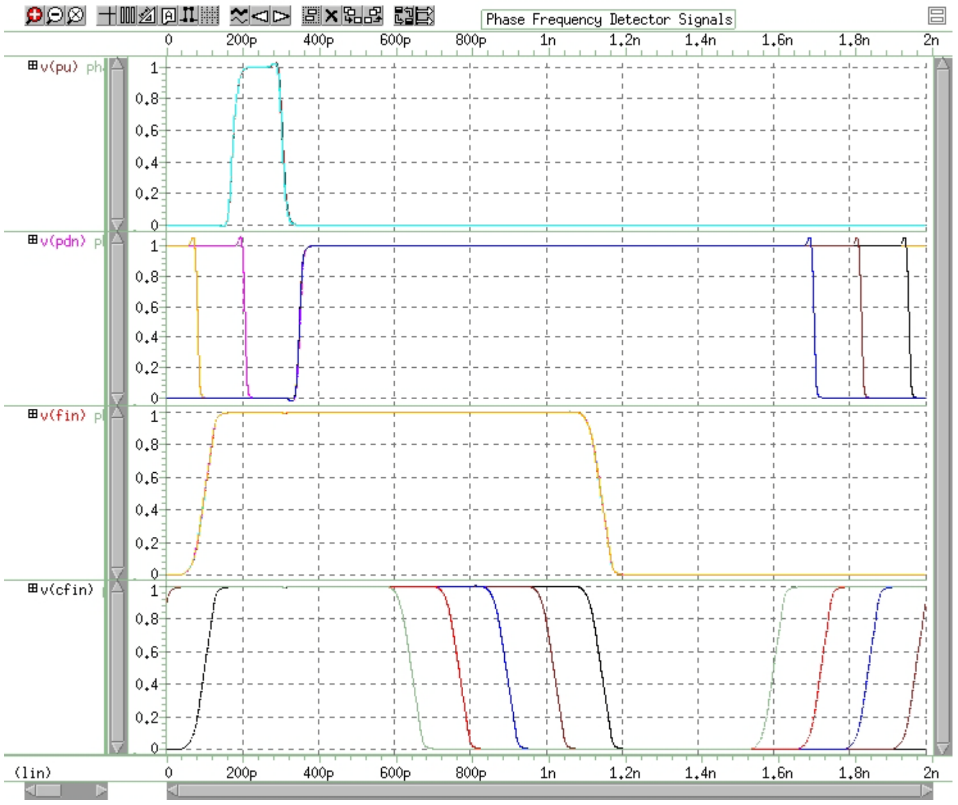


Figure 16 Phase Frequency Detector Signals

4. The frequency domain results are used to show the gain of the phase frequency detector. Use **File** > **Import Waveform** File to open the `phasefreqdet.snf0` file.
5. Use Waveview > New to open a new waveform.
6. Select the signal i(vload):(0) from the signal browser. Drag and drop the signal i(vload):(0) in the waveform. The signal is the DC component of the i(vload) signal spectrum. By default, the magnitude and phase of the load current are plotted. To measure the gain of the phase frequency detector verses phase, only the magnitude is required. Figure 17 on page 72 shows the gain of the phase frequency detector.

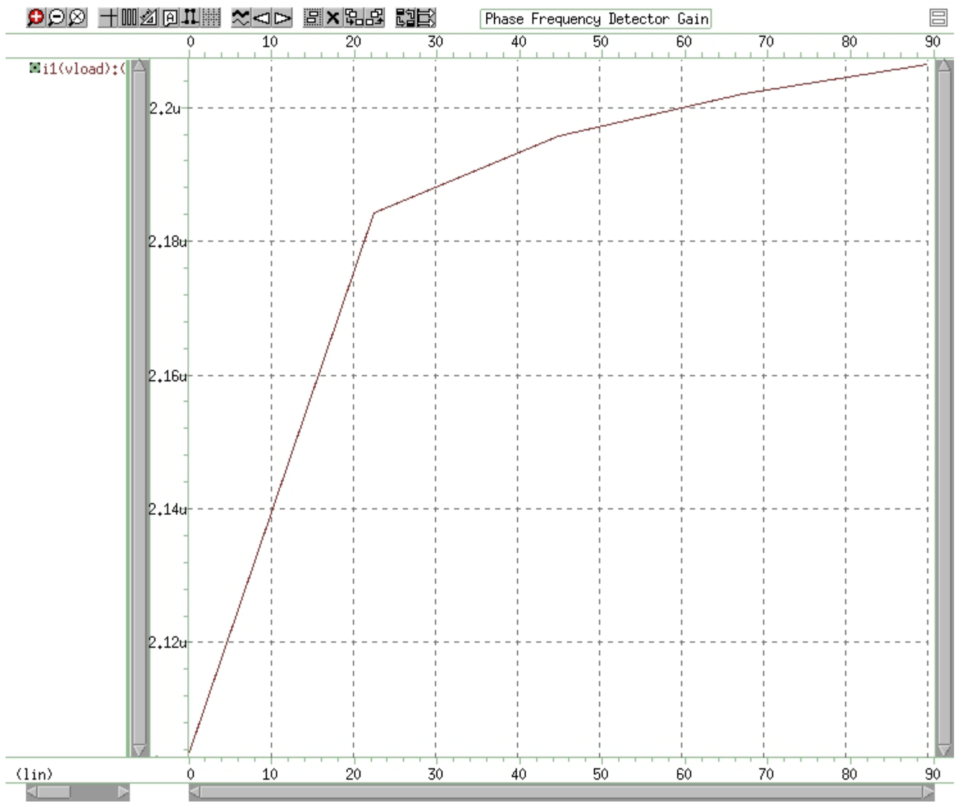


Figure 17 Phase Frequency Detector Gain

7. Next, plot the output noise of the phase frequency detector. Use **File** > **Import Waveform** File to open the `phasefreqdet.snpn0` file.`phasefreqdet.snpn0` file.
8. Use **Waveview** > **New** to open a new waveform.
9. Select the signal onoise() from the signal browser. Drag and drop the signal onoise() in the waveform. The noise results are shown in Figure 18 on page 73. This displays the noise at the output, v(pu, vpd) at each phase value swept in the `.SN` command.
10. Change the X-axis scale to log by left clicking on the X-axis and selecting **Log Scale** from the **X-Axis** menu.

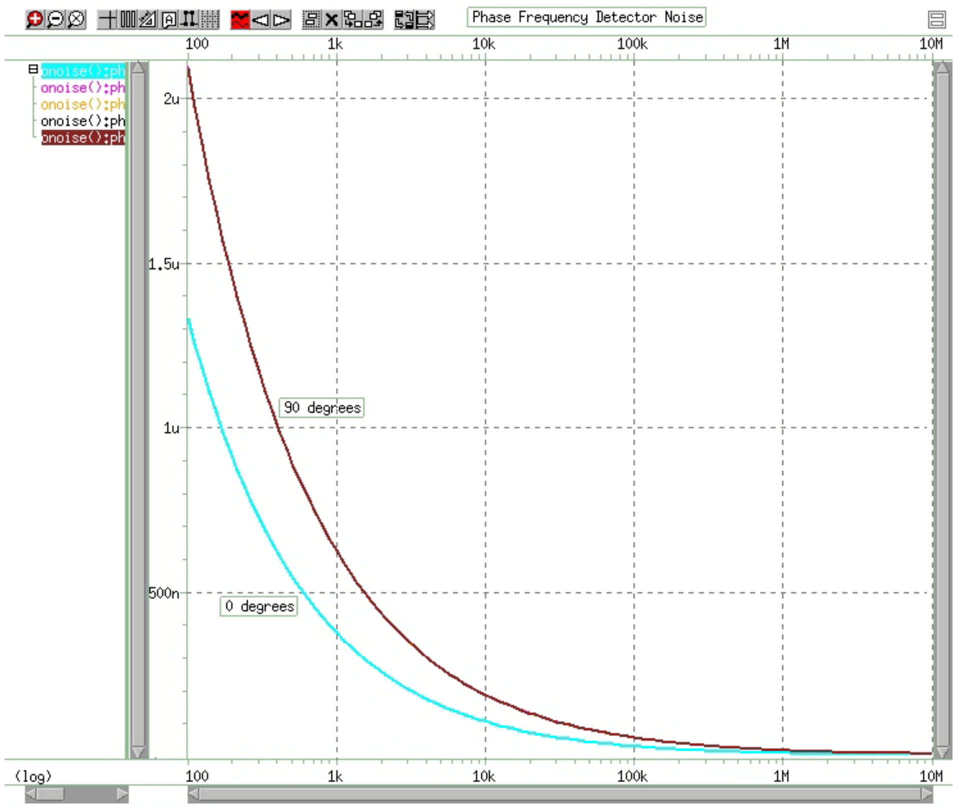


Figure 18   Phase Detector Output Noise

Ring Oscillator Example

The Shooting Newton algorithm provides fast and effective analysis for ring oscillators. The `ringoscSN.sp` input file for this example is located at

`$installdir/demo/hspice/rf_examples/`.

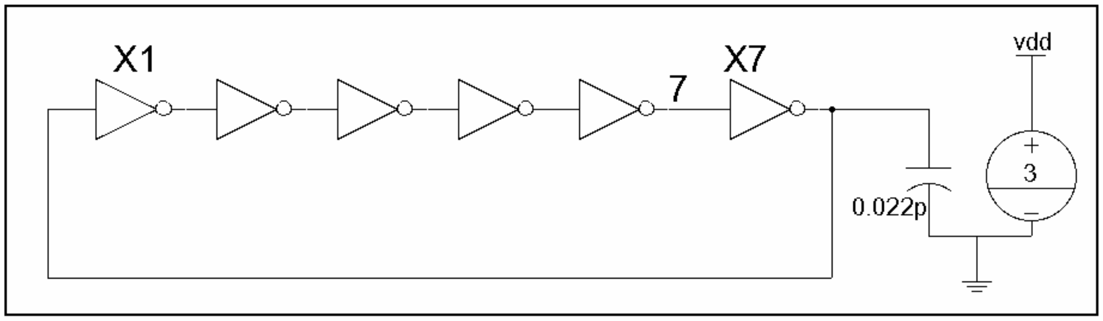


Figure 19 Ring Oscillator

```
.title ringosc

.subckt inv in out vdd
mn1 out in 0 0 nmos l=0.25u w=2u
mp1 out in vdd vdd pmos l=0.25u w=6u
.ends

vdd vdd 0 3
x1 1 2 vdd inv
x2 2 3 vdd inv
x3 3 4 vdd inv
x4 4 5 vdd inv
x5 5 6 vdd inv
x6 6 7 vdd inv
x7 7 1 vdd inv
c1 1 0 0.022p
.ic v(1)=3
.options post
.options snaccuracy=50

.snosc tones=335meg nharms=10 oscnode=1 trinit=10n
.phasenoise v(7) dec 10 100 10meg

.probe sn v(7)
.probe snfd v(7)
.print phasenoise phnoise v(7)
.probe phasenoise phnoise v(7)

.end
```

This analysis finds the oscillation frequency of the ring oscillator. Since the circuit is an oscillator, no input source is required. The oscillator is started by setting an initial condition at the input of the ring (node 1). In the `.SNOSC` command, the node that the analysis will probe for oscillation conditions is specified, as well as the approximate frequency of oscillation. The number of harmonics to include in the analysis is specified, as well.

The phase noise characteristics of the oscillator are analyzed by using the `.PHASENOISE` command. The `.PHASENOISE` command requires that an output node, pair of nodes, or a two-port element and a frequency sweep be specified. The frequency sweep is used to calculate the phase noise analysis at the specified offset frequencies, measured from the oscillation carrier frequency. For this example phase noise analysis, the default Nonlinear Perturbation (NLP) method is used.

The signals v(7) will be probed in both the frequency and time domain. The measure statement is used to measure the fundamental frequency of the oscillator.

Simulation Status Output

During the simulation, the simulation status is displayed on the screen. In addition to the screen display, more detailed status, cpu time and memory usage information is also written to the `ringoscSN.lis` file.

Viewing Results in Custom WaveView

You can view the time-domain, `ringoscSN.sn0` file, the frequency domain, `ringoscSN.snf0` file, and the phase noise, `ringoscSN.snpn0` file in Custom WaveView.

1. Enter **wv** *at the prompt* to start Custom WaveView.
2. Use **File** > **Import Waveform File** to open the `ringoscSN.sn0` file.
3. Select the signal v(7) from the signal browser. Drag and drop the signal v(7) to the right side of waveform so that panels are opened in row / column format. The time domain trace shown at the right side of Figure 20 on page 76.
4. Use **File** > **Import Waveform File** to open the `ringoscSN.snf0` file.
5. Select the signal v(7) from the signal browser. Drag and drop the signal v(7) to the right side of waveform so that panels are opened in row / column format. The frequency domain spectrum is shown at the left side of Figure 20 on page 76.

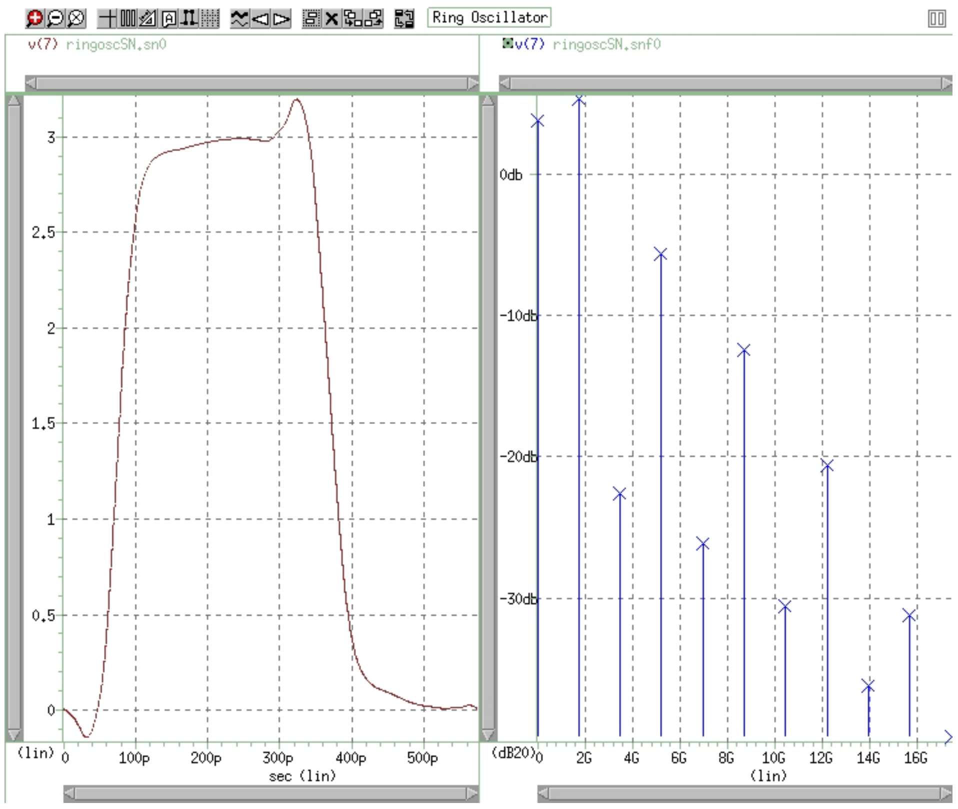


Figure 20 Ring Oscillator Output

6. Use **File** > **Import Waveform File** to open the `ringoscSN.snpn0` file.
7. Use **Waveview** > **New** to open a new waveform.
8. Select the signal nlp_l(f) from the signal browser. Drag and drop the signal nlp_l(f) signal in the waveform. Figure 21 on page 77 shows the resulting phase noise results for the oscillator.

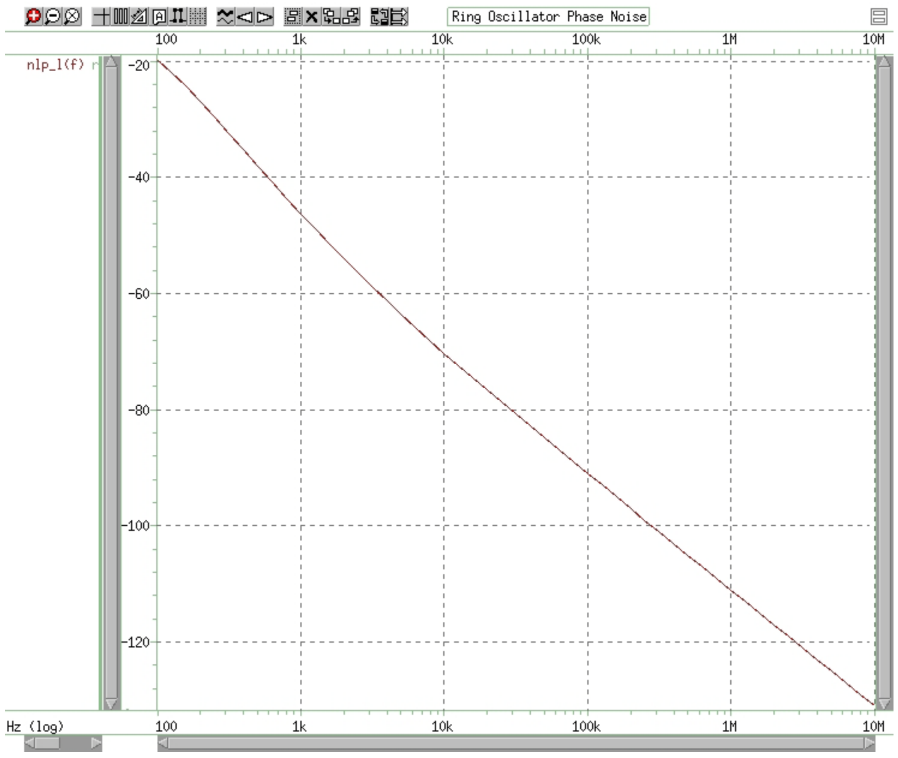


Figure 21 Ring Oscillator Phase Noise

Part 2: Oscillator and PLL Analysis

The following chapters/topics are included in this Part:

- Chapter 4, Oscillator and Phase Noise Analysis
- Chapter 5, Large Signal Periodic AC, Transfer Function, and Noise Analyses
- Chapter 6, S-parameter Analyses
- Chapter 7, Envelope Analysis
- Chapter 8, Post-Layout Analysis

4

Oscillator and Phase Noise Analysis

Describes how to use HSPICE advanced analog functions to perform oscillator and phase noise analysis on oscillator circuits.

Two main groups categorize oscillators:

- Ring oscillators: These oscillators tend to have low Q and operate based on delay of digital cells such as inverters. Ring oscillators have strong nonlinear behavior and output signals are often square-wave-like. You can analyze Ring oscillators in either the frequency domain using Harmonic Balance analysis or in the time domain using Shooting Newton analysis.
- Harmonic oscillators: Common harmonic oscillators are LC and crystal oscillators. These oscillators tend to have a high Q, making it difficult to find the oscillation frequency. Their behavior tends to be only mildly nonlinear and their output signals tend to be close to purely sinusoidal. Harmonic Balance analysis is most effective for analyzing harmonic oscillators.

HSPICE advanced analog analyses includes special analysis algorithms for finding the steady-state solution for oscillator circuits. No driving sources set the frequencies of operation in oscillators. The fundamental oscillation frequency is one of the unknowns that the simulator solves for. HSPICE advanced analog analyses provides two approaches: harmonic balance analysis or a Shooting Newton algorithm-based analysis.

This section covers the following topics:

- Harmonic Balance Oscillator Analysis (.HBOSC)
- Shooting Newton Oscillator Analysis(.SNOSC)
- Phase Noise Analysis (.PHASENOISE)
- Accumulated Jitter Measurement for Closed Loop PLL Analysis
- Clock Source with Random Jitter

- Small-Signal Phase-Domain Noise Analysis (.ACPHASENOISE)
- Behavioral Noise Sources
- References

Harmonic Balance Oscillator Analysis (.HBOSC)

Because the frequencies of driving sources do not determine the frequency of oscillation, the simulator solves for a slightly different set of nonlinear equations as in the following equation:

Equation 16

$$F(V, \omega_0) = I(V, \omega_0) + \Omega Q(V, \omega_0) + Y(\omega_0)V + I_s$$

HSPICE harmonic balance oscillator analysis (`.HBOSC`) adds the fundamental frequency of oscillation to the list of unknown circuit quantities. To accommodate the extra unknown, HSPICE sets the phase to zero (or equivalently, the imaginary part of one unknown variable — generally a node voltage). The phases of all circuit quantities are relative to the phase, at this reference node (referred to as the “PROBENODE”).

Additionally, the HBOSC analysis tries to avoid the “degenerate solution,” where all non-DC quantities are zero. Although this is a valid solution of the above equation (it is the correct solution, if the circuit does not oscillate), HBOSC analysis might find this solution incorrectly, if the algorithm starts from a bad initial solution.

The HBOSC analysis follows a technique similar to that described by Ngoya, et al, which uses an internally-applied voltage probe to find the oscillation voltage and frequency. The source resistance of this probe is a short circuit at the oscillation frequency, and an open circuit otherwise. HSPICE advanced analog analyses uses a two-tier Newton approach to find a non-zero probe voltage, which results in zero probe current.

HB analysis of the oscillator circuit uses the DC solution as a starting point. This analysis requires, in addition to the DC solution, initial values for both the oscillation frequency and the probe voltage. HBOSC analysis calculates the small-signal admittance that the voltage probe sees over a range of frequencies in an attempt to find potential oscillation frequencies. Oscillation is likely to occur where the real part of the probe current is negative, and the imaginary part is zero. You can use the `FSPTS` parameter to specify the frequency search. You must also supply an initial guess for the large signal probe voltage. A value of one-half the supply voltage is often a good starting point.

For more information on control options, see .HBOSC command in the *HSPICE Reference Manual: Commands and Control Options*.

The following sections discuss these topics:

- HB Simulation of Ring Oscillators
- .HBOSC Output Syntax
- Using the .MEASURE Command with .HBOSC
- Troubleshooting Convergence Problems
- Tutorial Examples Using HBOSC Analysis

HB Simulation of Ring Oscillators

Ring oscillators require a slightly different simulation approach in HB. Since their oscillation is due to the inherent delay in the inverters of the ring, they are best modeled in the time domain and not in the frequency domain.

In addition, ring oscillator waveforms frequently approach square waves, which require a large number of harmonics in the frequency domain. An accurate initial guess is important for an accurate HB simulation.

The HSPICE advanced analog HBOSC analysis typically starts from the DC solution and looks for potential resonances in the linear portion of the circuit to determine the initial guess for the oscillation frequency. However, these resonances generally do not exist in ring oscillators, which do not contain linear resonant elements.

HB analysis provides a second method of obtaining a good initial guess specifically intended for ring oscillators for the oscillation frequency. Instead of starting from the results of a DC analysis, this method starts from the result of a transient analysis. This method,*Transient Initialization*, also provides a good initial guess for all the voltages and currents in the circuit.

The recommended setup for ring oscillators is therefore:

- Set up .HBOSC without FSPTS.
- Choose one of the nodes in the ring as the PROBENODE.

- Recommendation: since ring oscillators tend to have square-wave-like output signals which have significant high frequency content, a relatively large value, perhaps 50, for nharms. Ring oscillators with more stages tend to need more harmonics.
- Set HBTRANINIT to a value that represents ~5-10 oscillator periods, and make sure that you include an .ic command or other transient analysis setup to start the oscillator in transient simulation. Longer HBTRANINIT times may result in faster HBOSC convergence, at the expense of additional CPU time spent on HBTRANINIT.

.HBOSC Output Syntax

The output syntax for .HBOSC analysis is identical to that for HB analysis (see Chapter 2, Steady-State Harmonic Balance Analysis). To output the final frequency of oscillation, use the `HERTZ` keyword. For example, `HERTZ[1]` identifies the fundamental frequency of oscillation.

Note: For `PROBENODE = n1 n2 vp`, where `vp` is a voltage, units must be in volts.

See also Outputting Phase Noise Source as ASCII Data Files Using *.printpn0.

Using the .MEASURE Command with .HBOSC

Since `.HBOSC` requires an `.HB` analysis, the measure statements for this analysis are the same as for `.HB` analysis. For example,

```
.MEASURE HB result FIND out_var AT=val
```

Troubleshooting Convergence Problems

This section lists the most common causes of convergence problems, how to recognize them, and resolve them.

The HSPICE advanced analog harmonic balance oscillator analysis consists of a two-tier iterative analysis of *inner loop* and *outer loop* iterations. In the outer loop iteration, HBOSC iterates to reduce any reported “probe errors” for each outer loop iteration. Each outer loop iteration involves a non-autonomous Harmonic Balance (HB) circuit solution; this non-autonomous solve is the *inner loop iteration*.

If HBOSC has inner loop convergence problems, the simulation may hang on the first outer loop iteration or you may see warning messages such as:

```
Warning: HB_WARN.3: Final HB residual value > HB_TOL.
Rank of HB Jacobian = 155
Warning: HB_WARN.3: HB convergence failure in non-autonomous HB.
```

The simulation lists the probe voltage and probe frequency for each outer loop iteration. If an outer loop convergence problem occurs, you may see the following:

- Decreasing probe voltage values.
- Wildly fluctuating values of probe frequency.

  ```
  Osc probe : voltage = 0.218234 frequency =
  6.240794122744832e+09
  ```

- A warning message which indicates that the oscillator simulation has reached a non-oscillating DC solution.

  ```
  Warning: HB_ERR.18: HB oscillator analysis has reached the
  NULL solution.
  ```

The following sections discuss these topics:

- General Convergence Issues
- Outer Loop Convergence
- Inner Loop Convergence

General Convergence Issues

The following sections discuss these topics:

- Probe Node Location
- Incorrect Source Values
- GMRES Convergence
- Accuracy of Initial Guess

Probe Node Location

Since convergence is sensitive to the probe node location, you can often track convergence problems to this setting.

A common scenario is that the oscillator's output signal passes through one or more buffers, and a designer may think of the buffer output as the oscillator output. A frequent mistake places the probe node at the output of the buffer, but this causes HB convergence problems. In this case, move the probe node to the oscillator part of the circuit. Often, it is necessary to select an internal node of a subcircuit to achieve this.

The most typical symptom of this problem is inner loop convergence failure. For ring oscillators, always choose a node that is part of the ring, i.e., connecting two stages of the ring; for harmonic oscillators, choose a node close to the oscillator.

Incorrect Source Values

If the original netlist simulates the oscillator in transient analysis, some voltage or current sources may have transient descriptions (e.g., PWL) to start the oscillator. For example, you can ramp a voltage supply to simulate a power-up to start the oscillator:

```
Vvdd vdd 0 PWL (0 0 1n 3)
```

In this case, the user would like HBOSC to use 3 as the voltage source value, but HBOSC uses 0 because harmonic balance uses the explicit DC value of the source. HSPICE advanced analog analyses tries to interpret your sources intelligently but, in some cases it may not be able to determine what you intended.

For the above example, there are a few ways to ensure that HSPICE advanced analog analyses correctly interprets the source.

- Remove the explicit DC value. If you only provide a transient description, HB uses the time=infinity value of the source.
- Add TRANFORHB=1

  ```
  Vvdd vdd 0 PWL (0 0 1n 3) TRANFORHB=1
  ```

  The TRANFORHB=1 keyword causes HB to use the transient analysis description in HB and HBOSC.
- Add an explicit HB value

  ```
  Vvdd vdd 0 PWL (0 0 1n 3) HB 3 0 0
  ```

  This causes HB to treat the source as a 3V DC source since the HB value specifies the 0th harmonic). If you provide an HB value, HSPICE advanced analog analyses ignores the PWL description and uses “HB 3 0 0” (amplitude=3, phase=0, harmonic=0) instead. The PWL description is still applies for HBTRANINIT.

Incorrect source values usually result in the following:

- High residual value after HBTRANINT. Usually, HBTRANINIT should produce a good starting point for HB or HBOSC. Typical residuals after HBTRANINIT are 1e-4 or 1e-5. If the initial residual printed immediately after HBTRANINIT completes is high, there may be a source problem. In the VDD ramping example above, you might see a residual value of 3.Outer loop may converge to DC solution because incorrect source values result in a non-oscillatory circuit:

  ```
  Warning: HB_ER.18: HB oscillator analysis has reached the NULL
  solution.
  ```

- In some cases, inner loop non-convergence may occur.

GMRES Convergence

When you set the default value for .option HBSOLVER (=1), HSPICE advanced analog analyses uses a GMRES iterative solver to solve the linear systems that arise on each inner loop Newton-Raphson step. If GMRES does not solve the linear systems accurately enough, then the inner loop may not converge.

The GMRES solver is controlled by two options:

- HBKRYLOVTOL: relative tolerance for GMRES solver. Default is 0.01, or 1%. For some circuits, setting this option helps inner loop convergence:

  ```
  .option HBKRYLOVTOL=1e-3
  ```

- HBKRYLOVDIM: dimension of Krylov subspace to use in GMRES iteration. Also controls maximum number of GMRES iterations. The HSPICE advanced analog analyses `.lis` file lists the number of GMRES iterations taken for each Newton-Raphson step. If that number is equal to HBKRYLOVDIM, you may improve convergence by increasing HBKRYLOVDIM. Example:

  ```
  .option HBKRYLOVDIM=80
  ```

The symptom for GMRES convergence difficulty is always inner loop convergence failure, or slow inner loop convergence. If this problem occurs, the inner loop convergence is often good until the residual reaches a fairly low value like 1e-8 or 1e-7, and then stagnates.

Accuracy of Initial Guess

Both inner loop and outer loop convergence improves significantly if the starting point or initial guess of the iterative method is good.

Outer Loop Convergence

For outer loop convergence, the initial guess consists simply of the oscillation frequency and first harmonic amplitude at the probe node location. If inner loop convergence is successful but outer loop convergence is not, then you may need to provide a better frequency or amplitude guess.

If you use HBTRANINIT, then you can improve the accuracy of the initial guess by use of one of the following methods:

- Increase the HBTRANINIT time, simply by increasing the value of the HBTRANINIT option.
- Increase the HBTRANINIT accuracy. You can increase the transient analysis accuracy by setting .option DELMAX or .option SIM_ACCURACY. For example, you may set

  ```
  .option SIM_ACCURACY=10 HBTOL=1e-8
  ```

  SIM_ACCURACY simultaneously tightens transient and HB accuracy tolerances. If you want HB accuracy to remain unaffected, you may also want to set HBTOL as in the example above.
- Increase accuracy of time domain to frequency domain conversion of HBTRANINIT results, by increasing HBTRANPTS or equivalently, decreasing HBTRANSTEP. For example:

  ```
  .option HBTRANSTEP=1p
  ```

If you are using FSPTS, you can increase the number of points. Sometimes, it is best to supply a guess manually by removing FSPTS and adjusting the TONES value.

If you do not use HBTRANINIT, you may be able to improve convergence by manually adjusting the PROBENODE amplitude guess.

To evaluate the effectiveness of your option settings, look at the “probe error” reported after the first outer loop iteration:

```
Iteration 1
Osc probe : voltage = 0.2 frequency = 5.980000000000000e+09
hb residual = 7.628260e-10
Rank of HB Jacobian = 9102
Probe error = 0.000154462
dv = -0.0411324 df =-2.30464430e+08
```

A smaller probe error value indicates a better initial guess.

Inner Loop Convergence

If inner loop convergence is a problem, it may be because the initial voltage waveform values are not close to the solution. The only way to improve the voltage values is by using HBTRANINIT. While this does not work well for harmonic oscillators, it does work well for ring oscillators. You can improve the accuracy of HBTRANINIT as described in the outer loop convergence section above.

If the initial residual is large after HBTRANINIT, you may want to check to make sure that the voltage and current sources are consistent between HB and transient analysis.

The following section discuss these topics:

- Insufficient Number of Harmonics
- Presence of Frequency Divider

Insufficient Number of Harmonics

If the number of harmonics specified is too small to represent the signals present in the circuit, you may see either convergence problems in either the inner or outer loop, or the solution may converge to an unreasonable frequency value.

It is difficult to know when the number of harmonics is insufficient, but if you suspect an insufficiency, it is a simple experiment to increase the value of NHARMS. If you achieved convergence and the number of harmonics is large enough, then the magnitude of the spectral data for all signals should significantly decay with increasing frequency. If the spectral data for node voltages has not decayed at the highest harmonics included in the simulation, increase the value of NHARMS.

Presence of Frequency Divider

If a frequency divider is present and not accounted for by the SUBHARMS setting, convergence is not possible because the Harmonic Balance spectrum does not include the necessary low frequency components. As a result, inner loop convergence fails. When debugging HBOSC convergence problems, it is necessary to rule out the possibility of presence of frequency dividers early in the process.

If a frequency divider is present, you can simulate the circuit if you set SUBHARMS to the largest frequency division present in the circuit. If a frequency divider is present, it is almost always necessary to use the HBTRANINIT option to achieve convergence.

To get optimal performance, it is recommend that you set

```
.option HBSOLVER=2
```

This activates a hybrid time/frequency-domain preconditioner which is particularly effective on frequency dividers.

Tutorial Examples Using HBOSC Analysis

The following tutorial examples illustrate:

- Example 1— Colpitts Oscillator
- Example 2 — Using HBOSC for a CMOS GPS VCO

Example 1— Colpitts Oscillator

This section demonstrates HSPICE advanced analog oscillator analysis by using a single transistor oscillator circuit. Oscillator analysis is an extension of Harmonic Balance in which you solve for the base frequency. In oscillator analysis, the user supplies a guess at the base frequency, and it requires no voltage or current source stimulus.

To activate oscillator analysis, include a `.HBOSC` command with:

- The `TONE` parameter set to a guess of the oscillation frequency.
- The `PROBENODE` parameter set to identify an oscillating node or pair of nodes. Always specify a pair of nodes; if only one node oscillates, specify ground as the second node. To speed up the simulation, also supply a guess at the magnitude of the oscillating voltage across these nodes.
- The `FSPTS` parameter set to a frequency range and number of search points. When you set `FSPTS`, HSPICE advanced analog analyses precedes the HBOSC analysis with a frequency search in the specified range to obtain an optimal initial guess for the oscillation frequency. This can accelerate the HB oscillator convergence.

In conjunction with oscillator analysis, HSPICE advanced analog analyses can perform phase noise analysis. Phase noise analysis measures the effect of transistor noise on the oscillator frequency. You activate phase noise analysis by using the `.PHASENOISE` command; this command sets a set of frequency points for phase noise analysis. The `.PRINT` and `.PROBE` commands output phase noise values.

The following netlist, `osc.sp`, simulates an oscillator, and performs phase noise analysis. This example file is available in the HSPICE distribution in the directory `$`*`installdir`*`/demo/hspice/rf_examples/`.

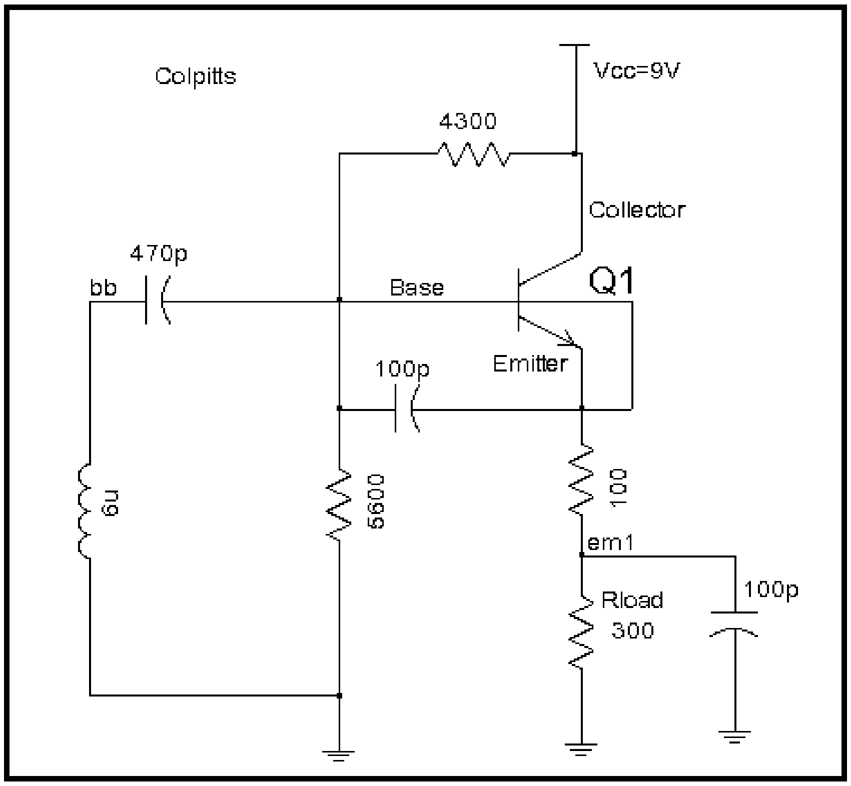


Figure 22 Colpitts Oscillator

Use the `.HBOSC` command with the `PROBENODE` and `FSPTS` parameters set.

```
PROBENODE=emitter,0,4.27
```

Identifies the emitter node as an oscillating node, and provides a guess value of 4.27 volts for the oscillation amplitude at the emitter node.

```
FSPTS=40,9e6,1.1e7
```

Causes an initial frequency search using 40 equally-spaced points between 9 and 11 MHz.

In the `.PHASENOISE`, `.PRINT`, and `.PROBE` commands:

```
.PHASENOISE V(emitter) dec 10 10k 1meg
```

Runs phase noise analysis at the specified offset frequencies, measured from the oscillation carrier frequency. The frequency points specified here are on a logarithmic scale, 10 points per decade, 10 kHz to 1 MHz.

- `.PROBE PHASENOISE PHNOISE` and the similar `.PRINT` command instruct HSPICE advanced analog analyses to output phase noise results to the `osc.pn0` and `osc.printpn0` files.

```
**
** Uses emitter resistor limiting to keep output sinusoidal.
** Output can be taken at the emitter (eml node).
**
*----------------------------------------------------------
* Options for Oscillator Harmonic Balance Analysis...
*
.OPTIONS post sim_accuracy=100 hbsolver=0
*----------------------------------------------------------
* Bias NPN transistor for 5V Vce, 10mA Ic
* Emitter follower Colpitts design
Vcc collector 0      9V
Q1 collector base emitter emitter RF_WB_NPN
Re1   emitter   eml     100
RLoad eml        0      300
Rb1   collector base 4300
Rb2   base       0     5600
*
*----------------------------------------------------------
* Capacitive feedback network
Ce    0    eml      100pF
Cfb base eml      100pF
Cbb base bb       470pF
Lb bb     0      6uH
*----------------------------------------------------------
* Simulation control for automated oscillator analysis
*
.HBOSC tones=1.0e7 nharms=15
+PROBENODE=emitter,0,4.27
+FSPTS=40,9.e6,1.1e7
*
.PHASENOISE V(emitter) DEC 10 10K 1MEG
+METHOD=0 CARRIERINDEX=1
*
.print hbosc vm(eml) vp(eml) vr(emitter) vi(emitter)
.print hbosc vm(emitter) vp(emitter) P(Rload)
.print phasenoise phnoise
.probe phasenoise phnoise
```

```
.probe hbosc v(emitter) v(eml)
.include bjt.inc
.END
```

After you run this netlist, examine the `osc.printhb0` file.

- At the top is the oscillator frequency (about 10.14 MHz) and the `.PRINT HBOSC` output.
- The first 2 lines show that the eml node oscillates around 3 V with an amplitude of about 2.85 V.
- The emitter node oscillates around 4V with an amplitude of about 4.27 V.

Also examine the `osc.printpn0` file, which contains the phase noise results in text form.

You can view the `osc.hb0` and `osc.pn0` files in Custom WaveView.

1. Type **wv** *at the prompt* to invoke Custom WaveView.
2. Use **File > Import Waveform File** and select the `osc.hb0` file from the Open: Waveform Files dialog box.
3. Select the v(emitter) signal in the signal browser. Double-click on the signal name or drag and drop the signal in the waveform.
4. In the waveform, right-click in the name area of the panel containing the signal v(emitter), left-click on the waveform label for v(emitter) in the waveform. From the Panel menu, choose Signal 'v(emitter)' > To Time-Domain.
5. To accept the defaults for range and interval, click OK in the Convert to Time domain window.
6. In a new waveform, you should now see a time domain waveform named IFT.0|v(emitter).

To run a transient simulation for comparison:

1. Use the `.TRAN 1n 10u` command.
2. Add `ic=10n` to the `Lb` inductor.

   The resulting waveforms should be the same as those from HB oscillator analysis.

Example 2 — Using HBOSC for a CMOS GPS VCO

This second oscillator analysis example involves two negative resistance oscillators coupled at 90 degrees. The MOS capacitors double as varactors.

This VCO topology is common for GPS applications and produces quadrature LO outputs near 1550 MHz. The purpose of this example is to generate the VCO tuning curve (output level and frequency as a function of tuning voltage), as well as its phase noise characteristics as a function of tuning voltage.

You activate the oscillator analysis by using the `.HBOSC` command:

- The `TONE` parameter sets an oscillation frequency (near 1550 MHz).
- The `NHARMS` parameter sets the harmonic content to 11th order.
- The `PROBENODE` parameters identify the drain pins across the first oscillator section as the pair of oscillating nodes. This is a differential oscillator, and the approximate value for this differential amplitude is 6.1 V.
- The `FSPTS` parameters set the search frequency range between 1500 and 1600 MHz.
- The `SWEEP` parameters set a tuning voltage sweep from 2.0 to 3.2 V.

The following example uses the demonstration netlist `gpsvco.sp`, which is available in directory `$`*`installdir`*`/demo/hspice/rf_examples/`. This netlist simulates the oscillator schematic Figure 23 and performs phase noise analysis.

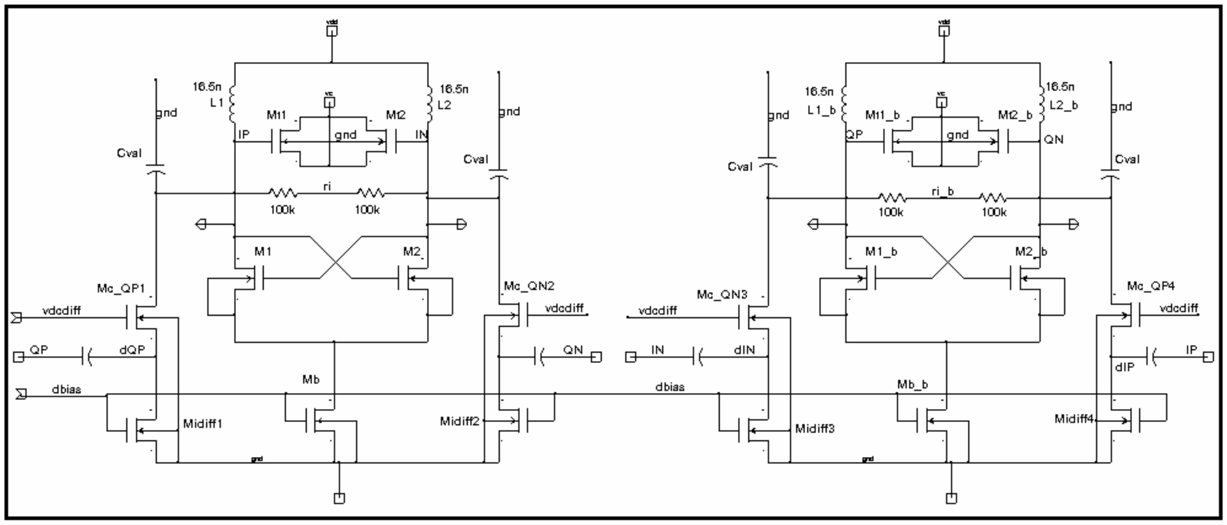


Figure 23 VCO Schematic

```
**
** NMOS IC Quadrature VCO circuit for GPS local oscillator
**
** Twin differential negative resistance VCOs using NMOS
** transistors for varactors, coupled to produce quadrature
** resonances.
** Design based on 0.35um CMOS process.
**
** References:
**  >P. Vancorenland and M.S.J. Steyaert, "A 1.57-GHz fully
**    integrated very low-phase-noise quadrature VCO,"
**    IEEE Trans. Solid-State Circuits, May 2002, pp.653-656.
**  >J. van der Tang, P. van de Ven, D. Kasperkovitz, and A.
Roermund,
**   "Analysis and design of an optimally coupled 5-GHz quadrature
**    LC oscillator,"  IEEE Trans. Solid-State Circuits, May 2002,
**     pp.657-661.
**  >F. Behbahani, H. Firouzkouhi, R. Chokkalingam, S. Delshadpour,
**    A. Kheirkhani, M. Nariman, M. Conta, and S. Bhatia,
**   "A fully integrated low-IF CMOS GPS radio with on-chip analog
**  image rejection," IEEE Trans. Solid-State Circuits, Dec. 2002,
**    pp. 1721-1727.
** Setup for Harmonic Balance Analysis
** Oscillation Frequency: ~ 1575 MHz (GPS L1 frequency)
** Amplitude:  ~5 Volts peak-to-peak (zero to 5V)
** Vdd: 2.5 V
**
** HSPICE Simulation Options:
*.option delmax=1n ACCURATE LIST NODE
**
** HSPICE advanced analog simulation options :
.option sim_accuracy=10
**
*.option savehb='a.hbs' loadhb='a.hbs'
.option POST
.param Vtune=2.0  $ Failures: vtune=1
.param Cval=0.2p
*--------------------------------
Vtune vc gnd  DC Vtune
Vdd vdd gnd 2.5
*--------------------------------
* First oscillator section
** Low-Q resonator with Vdd at center tap of inductors
R1a IP ri 100k  $ These R's set the Q
R1b ri IN 100k
L1 IP vdd 16.5nH
L2 vdd IN 16.5nH
Cc1 IP gnd Cval $ I to Q
```

```
Cc2 IN gnd Cval $ -I to Q
** Differential fets
M1 IP IN cs gnd NMOS l=0.35u w=15u
M2 IN IP cs gnd NMOS l=0.35u w=15u
** Bias fet - bias at Vdd -- too high?
Mb cs  vdd gnd gnd NMOS l=0.35u w=15u
** fets used as varactors
Mt1 vc IP vc gnd NMOS l=0.35u w=2u M=50
Mt2 vc IN vc gnd NMOS l=0.35u w=2u M=50
*--------------------------------
** Second oscillator section
** Low-Q resonator with Vdd at center tap of inductors
R1a_b QP ri_b 100k  $ These R's set the Q
R1b_b ri_b QN 100k
L1_b QP vdd 16.5nH
L2_b vdd QN 16.5nH
Cc1_b QP gnd Cval $ -Q to -I
Cc2_b QN gnd Cval $ -Q to I
** Differential fets
M1_b QP QN cs_b gnd NMOS l=0.35u w=15u
M2_b QN QP cs_b gnd NMOS l=0.35u w=15u
** Bias fet - bias at Vdd -- too high? 2nd in parallel
Mb_b cs_b  vdd gnd gnd NMOS l=0.35u w=15u
** fets used as varactors
Mt1_b vc QP vc gnd NMOS l=0.35u w=2u M=50
Mt2_b vc QN vc gnd NMOS l=0.35u w=2u M=50
*
*------------------------------
* Differentiators Coupling transistors for quadrature
*
.param Cdiff=0.14p difMsize=50u
vidiff  dbias gnd 1.25
viqdiff vdcdif gnd 1.75
Midiff1 dQP dbias gnd gnd NMOS l=0.35u w=difMsize
Midiff2 dQN dbias gnd gnd NMOS l=0.35u w=difMsize
Midiff3 dIN dbias gnd gnd NMOS l=0.35u w=difMsize
Midiff4 dIP dbias gnd gnd NMOS l=0.35u w=difMsize
Cdiff1  dQP QP Cdiff
Cdiff2  dQN QN Cdiff
Cdiff3  dIN IN   Cdiff
Cdiff4  dIP IP   Cdiff
Mc_QP1 IP vdcdif dQP gnd NMOS l=0.35u w=difMsize
Mc_QN2 IN vdcdif dQN gnd NMOS l=0.35u w=difMsize
Mc_QN3 QP vdcdif dIN gnd NMOS l=0.35u w=difMsize
Mc_QP4 QN vdcdif dIP gnd NMOS l=0.35u w=difMsize
*------------------------------
* Transient Analysis Test Bench
* stimulate oscillation with 2mA pulse
```

```
*iosc IP IN PULSE ( 0 2m .01n .01n .01n 10n 1u )
*.probe tran v(IP) v(IN)
*.print tran v(IP) v(IN)
*.TRAN .01n 10n
*-----------------------------
* Harmonic Balance Test Bench
*
.sweepblock vtune_sweep
+ 0 5 0.2
+ 2 3 0.1
.HBOSC tones=1550e6 nharms=12
+ PROBENODE=IP,QN,4
+ sweep Vtune sweepblock=vtune_sweep
**
.phasenoise v(IP,IN)dec 10 100 1e7
.print phasenoise phnoise
.probe phasenoise phnoise
.print hb v(IP,IN) v(IP,IN)[1] v(QP,QN) v(QP,QN)[1]
.probe hb v(IP,IN) v(IP,IN)[1] v(QP,QN) v(QP,QN)[1]
.probe hb hertz[1]
*
* NMOS Device from MOSIS 0.35um Process
*
* BSIM3 VERSION 3.1 PARAMETERS
*
* DATE: Mar  8/00
* LOT: n9co                WAF: 07
* Temperature_parameters=Default
*
.MODEL NMOS NMOS (                              LEVEL   = 49
+VERSION = 3.1          TNOM    = 27           TOX     = 7.9E-9
+XJ     = 1.5E-7       NCH     = 1.7E17       VTH0    = 0.5047781
+K1     = 0.5719698     K2     = 0.0197928     K3     = 33.4446099
+K3B    = -3.1667861    W0     = 1E-5         NLX    = 2.455237E-7
+DVT0W   = 0              DVT1W   = 0              DVT2W   = 0
+DVT0   = 2.8937881     DVT1   = 0.6610934     DVT2   = -0.0446083
+U0    = 421.8714618   UA    = -1.18967E-10  UB    = 1.621684E-18
+UC     = 3.422111E-11  VSAT   = 1.145012E5    A0     = 1.119634
+AGS     = 0.1918651      B0      = 1.800933E-6    B1      = 5E-6
+KETA    = 3.313177E-3    A1      = 0              A2      = 1
+RDSW   = 984.149934    PRWG   = -1.133763E-3  PRWB   = -7.19717E-3
+WR     = 1            WINT   = 9.590106E-8   LINT   = 1.719803E-8
+XL     = -5E-8         XW     = 0            DWG    = -2.019736E-9
+DWB     = 6.217095E-9    VOFF    = -0.1076921     NFACTOR = 0
+CIT     = 0              CDSC    = 2.4E-4         CDSCD   = 0
+CDSCB   = 0            ETA0   = 0.0147171     ETAB   = -7.256296E-3
+DSUB   = 0.3377074     PCLM   = 1.1535622     PDIBLC1 = 2.946624E-4
+PDIBLC2 = 4.171891E-3   PDIBLCB = 0.0497942     DROUT  = 0.0799917
+PSCBE1  = 3.380501E9    PSCBE2  = 1.69587E-9    PVAG    = 0.4105571
+DELTA    = 0.01           MOBMOD   = 1              PRT      = 0
+UTE      = -1.5           KT1      = -0.11          KT1L     = 0
+KT2     = 0.022         UA1     = 4.31E-9       UB1     = -7.61E-18
+UC1      = -5.6E-11       AT       = 3.3E4          WL       = 0
+WLN     = 1             WW      = -1.22182E-15   WWN     = 1.1657
+WWL      = 0              LL       = 0              LLN      = 1
+LW       = 0              LWN      = 1              LWL      = 0
+CAPMOD  = 2             XPART   = 0.4           CGDO    = 3.73E-10
+CGSO    = 3.73E-10      CGBO    = 1E-11        CJ      = 8.988141E-4
+PB     = 0.8616985     MJ     = 0.3906381     CJSW   = 2.463277E-10
+PBSW    = 0.5072799     MJSW    = 0.1331717     PVTH0   = -0.0143809
+PRDSW   = -81.683425    WRDSW   = -107.8071189  PK2    = 1.210197E-3
+WKETA   = -1.00008E-3   LKETA   = -6.1699E-3    PAGS    = 0.24968
+AF       = 1.0            KF       = 1.0E-30      )
*
.END
```

Figure 24 displays the results of the analysis, along with Figure 25 on page 99, Figure 26 on page 99, and Figure 27 on page 100 using Custom WaveView for VCO waveforms, tuning curves, and phase noise response.

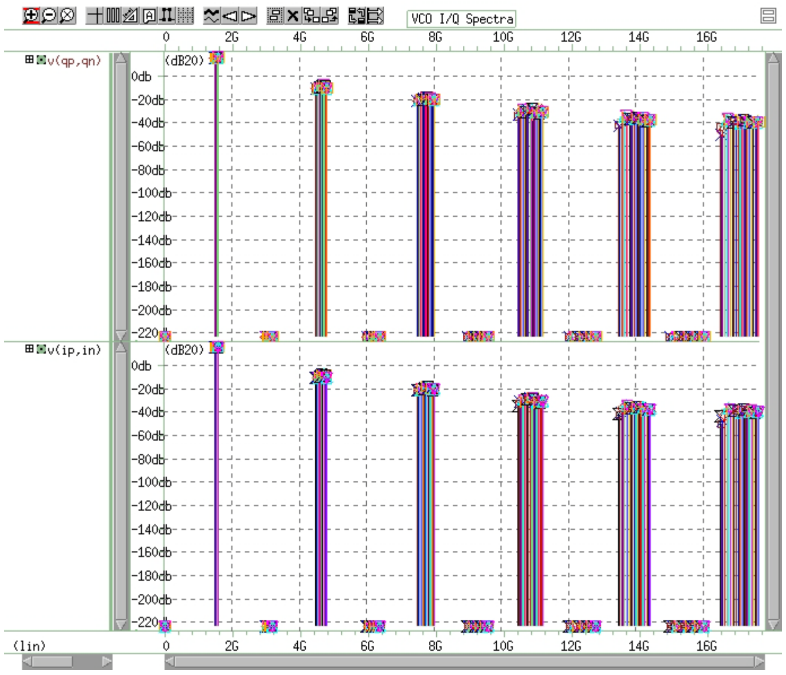


Figure 24   VCO Spectra Output

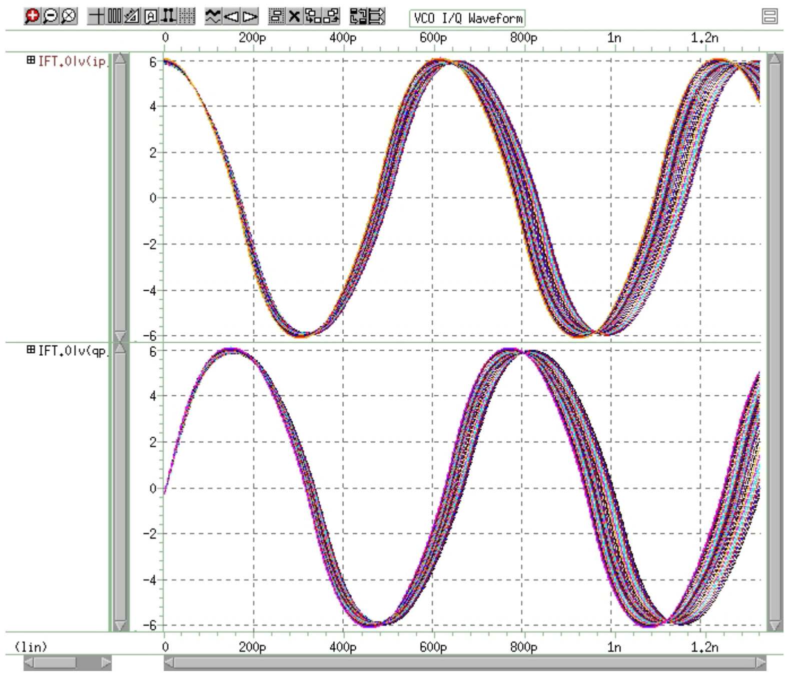


Figure 25 VCO Waveform Output

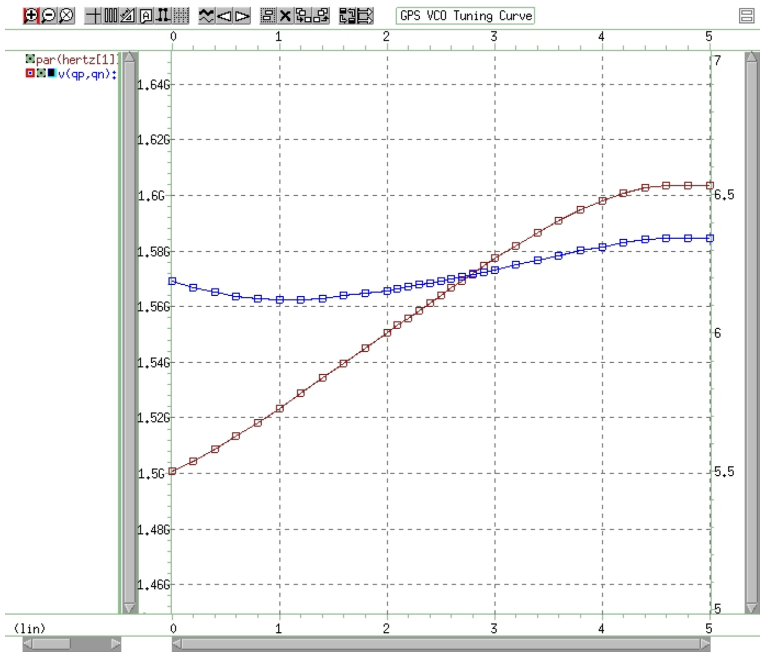


Figure 26 VCO Tuning Curves Output

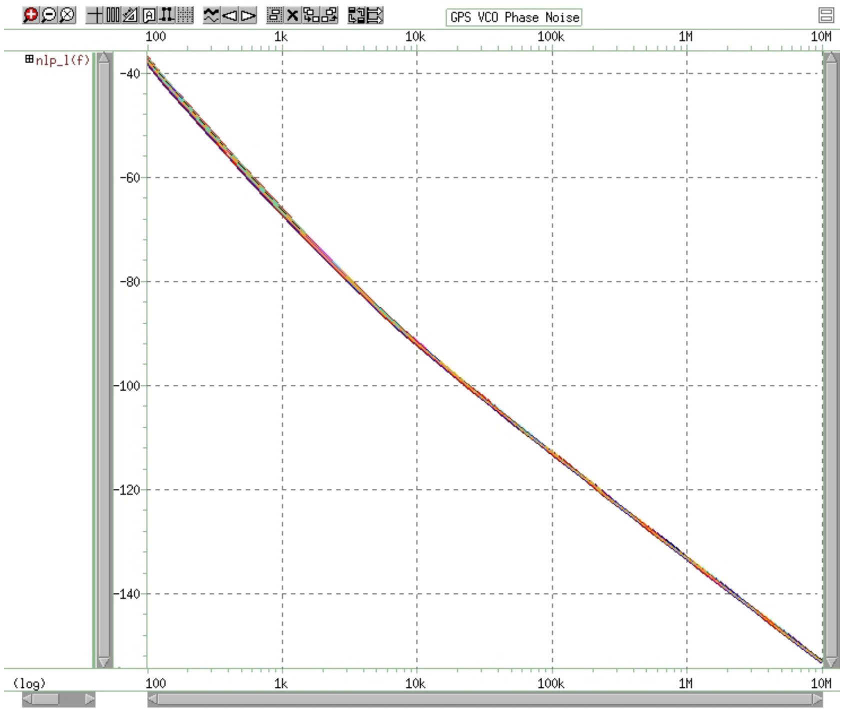


Figure 27 VCO Phase Noise Response

Shooting Newton Oscillator Analysis(.SNOSC)

The analysis described in Chapter 3, Steady-State Shooting Newton Analysis also provides a very effective means for finding the steady-state for oscillator circuits.

Ring oscillators are best suited for time domain analysis by using Shooting Newton because they tend to:

- have a low Q
- operate based on digital delays
- have strongly nonlinear behavior
- output signals that are piece-wise-linear or square-wave-like

HBOSC is superior for sinusoidal waveforms. As with the Harmonic Balance approach, the goal is to solve for the additional unknown oscillation frequency. Shooting Newton accomplishes this by considering the period of the waveform as an additional unknown, and solving the boundary conditions at the waveform endpoints that coincide with steady-state operation. As with regular Shooting Newton analysis, you can specify input in terms of time or frequency values.

For more information on control options, see .SNOSC command in the *HSPICE Reference Manual: Commands and Control Options*.

.SNOSC Output Syntax

The output syntax for .SNOSC analysis is identical to that for SN analysis (see Chapter 2, Steady-State Harmonic Balance Analysis). To output the final frequency of oscillation, use the `HERTZ` keyword. For example, `HERTZ[1]` identifies the fundamental frequency of oscillation.

See also Using Noise Analysis Results as Input Noise Source.

Phase Noise Analysis (.PHASENOISE)

.PHASENOISE analysis autonomous oscillators. Phase Noise analysis requires first running either harmonic balance (HBOSC) or Shooting Newton (SNOSC) analysis, and then PHASENOISE analysis. The PHASENOISE analysis itself is identical whether you run SNOSC or HBOSC.

For more information on control options, see .PHASENOISE command in the *HSPICE Reference Manual: Commands and Control Options*.

The following figure shows a simple free-running oscillator, which includes a port with injected current.

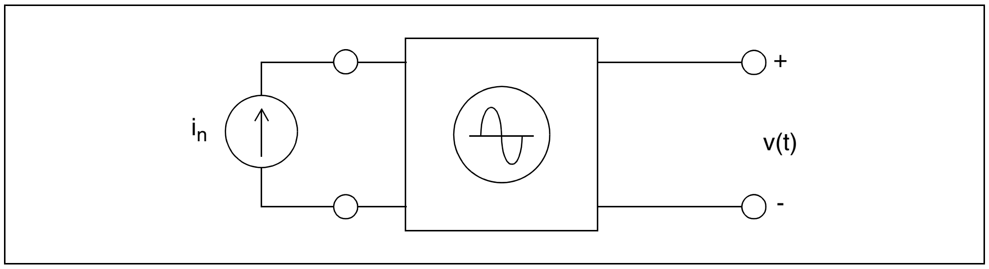


Figure 28 Oscillator with Injected Current

The following equation presents how an ideal oscillator would be insensitive to perturbations with a fixed amplitude, frequency:

Equation 17

$$v(t) = A\cos[\omega_0 t + \phi_0]$$

A noisy oscillator has amplitude and phase fluctuations we can write as:

Equation 18

$$v(t) = A(t)\cos[\omega_0 t + \phi(t)]$$

In the preceding equation:

- *A(t)* is the time varying amplitude for the noisy oscillator.
- $\phi(t)$ is the time varying phase for the noisy oscillator.
- ω_0 is the frequency of oscillation.

In most applications, the phase noise is of particular interest, because it represents frequency fluctuations about the fundamental, which you cannot remove. These fluctuations are random processes. Typical expressions are in terms of their power spectral density. For most oscillators, the phase noise is a low-frequency modulation that creates sidebands in the oscillator's spectrum, about ω_0 .

For example, the following equation represents a simple sinusoidal variation in the phase:

Equation 19

$$v(t) = A\cos[\omega_0 t + \theta_p \sin\omega_m t]$$

- θ_p is the peak phase deviation, specified as $\theta_p = \Delta\omega/\omega_m$
- $\Delta\omega$ is the peak angular frequency deviation.

For $\theta_p \ll 1$, the following equation approximates the output:

Equation 20

$$v(t) = A\left\{\cos(\omega_0 t) - \frac{\theta_p}{2}[\cos(\omega_0 + \omega_p)t - \cos(\omega_0 + \omega_m)t]\right\}$$

That is, when the peak phase deviation is small, the result is frequency components on each side of the fundamental with amplitude $\theta_p/2$.

The Single-Sideband Phase Noise $L(f_m)$ is the ratio of noise power to carrier power in a 1Hz bandwidth, at offset $\omega_m = 2\pi f_m$, which in this case can be written as:

Equation 21

$$L(f_m) = \left(\frac{V_{sb}}{A}\right)^2 = \frac{\theta_p^2}{4} = \frac{\theta_{rms}^{\ 2}}{2}$$

This model for oscillator noise shows that sidebands about the fundamental, due to noise, directly relate directly to the spectrum of the phase fluctuations $\theta(t)$. The more general definition of phase noise relates it to the spectral density of phase fluctuations, i.

Equation 22

$$S_{\phi}(\omega_m) = \frac{\theta_p^2}{2} = 2L(f_m)$$

HSPICE advanced analog analyses uses several sophisticated analysis techniques for computing the power spectrum of the phase variations to yield the phase noise response. This information informs of the spectrum of the oscillator about the fundamental frequency, and informs of its random jitter characteristics.

Any `.PHASENOISE` analysis results in the calculation of a curve fit for a power-law model according to:

Equation 23

$$L(f) = 10 \cdot \log\left\{\frac{a3}{f^3} + \frac{a2}{f^2} + \frac{a1}{f} + a0\right\}$$

The `.lis` file reports the coefficients $a3$, $a2$, $a1$, and $a0$.

The `.lis` file includes a table which models the phase noise, including the behavioral model fit and its fit error.

```
|--------------------------------------------------------------|
| L(f) = 10*log10( a3/f^(2+ef) + a2/f^2 + a1/f^(ef) + a0 ) dBc/Hz|
| a3 = 0.000000e+00                                             |
| a2 = 3.165111e-02                                             |
| a1 = 0.000000e+00                                             |
| a0 = 0.000000e+00                                             |
| ef = 1.000000e+00                                             |
| Average fit error = 1.6185e+00 dB                             |
| Maximum fit error = 8.4862e+00 dB @ 1.0000e+07 Hz             |
```

This section covers the following topics:

- Identifying Phase Noise Spurious Signals
- Phase Noise Algorithms
- PHASENOISE Output Syntax
- Measuring Phase Noise with .MEASURE PHASENOISE
- Amplitude Modulation/Phase Modulation Separation

Identifying Phase Noise Spurious Signals

Realistic phase noise responses include spurs. Spurs are contributions to the phase noise that result from deterministic signals present within the circuit. In most cases, the spurs are very small signals and do not interfere with the steady-state operation of the oscillator, but do add energy to the output spectrum of the oscillator. You may need to include the energy that the spurs add in jitter measurements. The phase noise spurs feature adds an additional analysis option that can predict the spurious contributions to the jitter.

To activate the new phase noise spur analysis, use the `SPURIOUS` keyword in the `.PHASENOISE` command. An additional `.HBAC` analysis predicts the spurious contributions to the phase noise.

Use a voltage or current source can to add spurious signals to an oscillator circuit. The keyword `SPUR` identifies the spurious signal.

Syntax

```
Exxxx n1 n2 … [SPUR mag phase freq] … $ voltage spur
Gxxxx n1 n2 … [SPUR mag phase freq] … $ current spur
```

where:

- *`mag`* is the amplitude in volts or amps
- *`phase`* is the phase in degrees
- *`freq`* is the frequency in Hz

The source is equivalent to a steady-state sinusoidal source at the specified amplitude, phase, and frequency values and only the spurious analysis uses it. All other analyses ignore it. The SPUR keyword is combinable into a source that other analyses use. Recommendation: add SPUR sources as separate sources.

Phase Noise Algorithms

HSPICE advanced analog analyses provides three algorithms for oscillator phasenoise: nonlinear perturbation, periodic AC, and broadband calculations. The METHOD parameter to 0, 1, or 2, respectively, selects these algorithms.

Each algorithm has its regions of validity and computational efficiency, so some thought is necessary to obtain meaningful results from a PHASENOISE simulation. In each simulation, for each algorithm, the region of validity depends on the particular circuit. However, you can apply general rules that to oscillator types (that is, ring or harmonic) to identify a valid region. You can use techniques to check validity of your simulation results.

This section covers the following algorithms:

- Nonlinear Perturbation Algorithm
- Periodic AC Algorithm
- Broadband Phase Noise Algorithm

Nonlinear Perturbation Algorithm

The nonlinear perturbation (NLP) algorithm, which is the default selection, is typically the fastest computation, but is valid only in a region close to the carrier. Generally, you want to use this algorithm if you interested in phasenoise close to the carrier and do not need to determine a noise floor. NLP computation time is almost independent of the number of frequency points in the phasenoise frequency sweep.

Periodic AC Algorithm

The periodic AC (PAC) algorithm is valid in a region away from the carrier and is slower than the NLP algorithm. Use the PAC algorithm for getting phasenoise in the far carrier region and when you need to determine a noise floor.

The computation time for the PAC algorithm is approximately linearly dependent on the number of frequency points in the phasenoise frequency sweep. If you are using the PAC algorithm, you should try to minimize the number of points in the sweep.

Another issue is that the PAC algorithm becomes more ill-conditioned as you approach the carrier. This means that you may have to generate a steady-state solution with more harmonics to get an accurate simulation as you get closer to the carrier. So, if you find that the PAC is rolling off at close-in frequencies, you should rerun HB analysis with a larger number of harmonics. Although, typically, you do not see improvements in PAC accuracy beyond more than about 100-200 harmonics.

Early in your testing, the best way to verify that NLP and PAC are giving accurate results is to run both algorithms over a broad frequency range and check that the curves have some range in frequency where they overlap. Typically, you see the NLP curve rolling off at 20 to 30 dB/decade as frequency increases, characteristic of white noise or 1/f noise behavior. Also, the PAC curve at first is flat or even noisy close to the carrier. At some point though, you see this curve match the NLP roll-off.

The lowest frequency at which the curves overlap defines the point, f_{PAC} above which the PAC algorithm is valid. Sometimes, by increasing the number of HB harmonics, it is possible to move f_{PAC} to lower frequencies. The highest frequency at which the curves overlap defines the point, f_{NLP} below which the NLP algorithm is valid. A rough rule of thumb is that $f_{PAC} = f_o/Q$, where f_o is the carrier frequency and Q is the oscillator Q-value. This implies that for high-Q oscillators, such as crystal and some harmonic oscillators, that PAC is accurate to values quite close to the carrier.

Broadband Phase Noise Algorithm

The Broadband Phase Noise (BPN) algorithm allows phase noise simulation over a broad frequency range. The BPN algorithm runs both the NLP and PAC algorithms and then connects them in the overlap region to generate a single phase noise curve. This algorithm is ideal for verifying the NLP and PAC accuracy regions and when you require a phase noise response over a broad frequency range.

PHASENOISE Output Syntax

You can analyze element phase noise through the `.PRINT` and `.PROBE` statements. HSPICE supports the output of the phase noise and phase noise due to a specified element. In addition, by using specialized keywords, you can output phase noise due to noise source types. To generate output, you must enable the `listsources` option of the `.PHASENOISE` command (`=on`).

Whole Circuit and Specified Element Phase Noise

A single `phnoise` keyword specifies the phase noise for the whole circuit, and the `phnoise(`*`element_name`*`)` specifies the phase-noise value of a specified element in the circuit.

```
.PRINT PHASENOISE phnoise phnoise(element_name)
.PROBE PHASENOISE phnoise phnoise(element_name)
```

In this syntax, the standalone `phnoise` is the phase noise parameter. For example:

```
.PROBE PHASENOISE phnoise
```

The `.PHASENOISE` statement outputs raw data to the `*.pn#` and `*.printpn#` files. HSPICE advanced analog analyses outputs the `phnoise` data in decibels, relative to the carrier signal, per hertz, across the output nodes in the `.PHASENOISE` statement (Units: dBc/Hz). The data plot is a function of the offset frequency.

HSPICE advanced analog analyses outputs `phnoise` to the `.pn#` file if you set `.OPTION POST`.

Frequency-Dependent and Frequency-independent Sources

- The `phnoise_fdep` keyword variable collects all *frequency-dependent* noise sources' contributions to the phase noise.
- The `phnoise_findep` keyword variable collects all *frequency independent* noise sources' contributions.

```
.print phasenoise phnoise_fdep
.print phasenoise phnoise_findep
```

Frequency and Bias Dependencies

Table 1 Summary of Noise Type Dependences

Noise type	frequency-dependent	bias-dependent
`phnoise_stationary`	No	No
The following syntax is frequency-independent and bias-independent: `.print phasenoise phnoise_stationary`		
`phnoise_cyclostationary` or `phnoise_cyclo`	No	Yes

The following syntax is frequency-independent and bias-dependent:
`.print phasenoise phnoise_cyclo`
or
`.print phasenoise phnoise_cyclostationary`
Where: `cyclo` or `cyclostationary` means anything bias-dependent.		

`phnoise_flicker`	Yes	No
The following syntax is bias-independent and frequency-dependent: `.print phasenoise phnoise_flicker`		
`phnoise_cycloflicker`	Yes	Yes

The following syntax is frequency-dependent and bias-dependent:
`.print phasenoise phnoise_cycloflicker`
or
`.print phasenoise phnoise_cyclostationaryflicker`		

`phnoise_fdep`	is the union of `phnoise_Flicker` and `phnoise_cycloflicker` noise types	
`phnoise_findep`	is the union of `phnoise_stationary` and `phnoise_cyclostationary` noise types	

Example 1

This example performs an oscillator analysis, by searching for frequencies in the vicinity of 900 MHz, followed by a phase noise analysis at frequency offsets from 100 Hz to 10 MHz.

```
.HBOSC TONE=900MEG NHARMS=9
+ PROBENODE=gate,gnd,0.65
.PHASENOISE V(gate,gnd) DEC 10 100 1.0e7
+ METHOD=0 CARRIERINDEX=1 $use NLP algorithm
+ listsources=on
.PROBE PHASENOISE phnoise
.PRINT PHASENOISE phnoise(X1)
```

Example 2

This example performs a VCO analysis, by searching for frequencies in the vicinity of 2.4 GHz. This example uses eleven harmonics and sweeps the VCO tuning voltage from 0 to 5 V. HSPICE advanced analog analyses uses the nonlinear perturbation (NLP) algorithm to perform a phase noise analysis based on the fundamental frequency for each tuning voltage value.

```
.HBOSC TONE=2400MEG NHARMS=11
+ PROBENODE=drainP,drainN,1.0
+ FSPTS=20,2100MEG,2700MEG
+ SWEEP Vtune 0.0 5.0 0.2
.PHASENOISE V(drainP,drainN) DEC 10 100 1.0e7
+ METHOD=0 CARRIERINDEX=1 $use NLP algorithm
+ listsouces=on
.PROBE PHASENOISE phnoise
.PRINT PHASENOISE phnoise(X2)
```

See Also

- Using Noise Analysis Results as Input Noise Sources.

Measuring Phase Noise with .MEASURE PHASENOISE

The HSPICE advanced analog optimization flow can read the measured data from a `.MEASURE PHASENOISE` analysis. You can combine this flow in the HSPICE advanced analog analyses optimization routine with a `.MEASURE HBTR` analysis (see Using .MEASURE with .HB Analyses) and a `.MEASURE HBNOISE` analysis (see Measuring HBNOISE Analyses with .MEASURE). The `.MEASURE PHASENOISE` syntax supports the following measurements:

- FIND

  ```
  .MEASURE PHASENOISE result FIND phnoise at = IFB_value
  ```

  — yields the result of a variable value at a specific input frequency band (IFB) point. For example:

  ```
  .MEASURE PHASENOISE np1 find PHNOISE at=100K
  ```

- WHEN

  ```
  .MEASURE PHASENOISE result WHEN phnoise=value
  ```

  —yields the input frequency point at a specific phnoise value. For example:

  ```
  .MEASURE PHASENOISE fcorn1 when PHNOISE=-120
  ```

- RMS, average, min, max, and peak-to-peak

  ```
  .MEASURE PHASENOISE result func phnoise
  + [FROM = IFB1] [TO = IFB2]
  ```

  —yields the average, RMS, minimum, maximum, or peak-to-peak value of the phase noise from frequency `IFB1` to frequency `IFB2`, where the value of `func` can be RMS, AVG, MIN, MAX or PP. If you do not specify `FROM` and `TO`, HSPICE calculates the value is over the frequency range that you specify in the `.PHASENOISE` command. For example:

  ```
  .measure PHASENOISE agn1 AVG phnoise from=100k to=10meg
  ```

- Integral evaluation

  ```
  .MEASURE PHASENOISE result INTEGRAL phnoise
  + [FROM = IFB1] [TO = IFB2]
  ```

  —integrates the phase noise value from the `IFB1` frequency to the `IFB2` frequency. For example:

  ```
  .MEASURE PHASENOISE rns1 INTEGRAL phnoise from=50k to 500k
  ```

- Derivative evaluation

  ```
  .MEASURE PHASENOISE result DERIVATIVE phnoise AT = IFB1
  ```

  —finds the derivative of phase noise at the `IFB1` frequency point. For example:

  ```
  .MEASURE PHASENOISE fdn1 DERIVATIVE phnoise at=10meg
  ```

Note: `.MEASURE PHASENOISE` cannot contain an expression that uses a phasenoise variable as an argument. You also cannot use `.MEASURE PHASENOISE` for error measurement and expression evaluation of the `.PHASENOISE` command.

See also, the .MEASURE PHASENOISE command in the *HSPICE Reference Manual: Commands and Control Options*.

Amplitude Modulation/Phase Modulation Separation

You can separate the Amplitude Modulation (AM) and Phase Modulation (PM) components of the total noise by calculating components in-phase (AM component) and in quadrature (PM component) with the carrier by using PAC and BPN PHASENOISE analysis. The output and measure syntax separates AM/PM noise.

Turn this feature on by setting the `.OPTION PHNOISEAMPM=1` (see .OPTION PHNOISEAMPM in the *HSPICE Reference Manual: Commands and Control Options*. See also, Important Note for AM/PM Users at the end of this section.

- If you use the NLP algorithm (`METHOD=0`) default, HSPICE advanced analog analyses calculates only the phase noise component.
- If you use either the PAC algorithm (`METHOD=1`) or the BPN algorithm (`METHOD=2`), HSPICE advanced analog analyses adds both the phase and amplitude noise components together to show the total noise at the output.

AM/PM .PRINT and .PROBE Statement Syntax

```
.PROBE PHASENOISE phnoise [la] [ltotal] [onoise]
.PRINT PHASENOISE phnoise [la] [ltotal] [onoise]
```

Keywords for AM/PM separations are:

- Phase Modulation term only: `phnoise`
- Amplitude Modulation term only: `la`
- Total phase noise term: `ltotal`
- Voltage noise term: `onoise`

For example:

```
.probe phasenoise phnoise la ltotal $ AM modulation and
                                      total phase noise
```

Add a Noise type suffix to each of these noise terms, `phnoise`, `la`, `ltotal`, `onoise`, to select specific noise-type components:

Table 2 Summary of Noise_term

Noise type	frequency-dependent	bias-dependent
Noise_term`_phnoise_stationary`	No	No
Noise_term`_phnoise_cyclostationary`	No	Yes
Noise_term`_phnoise_flicker`	Yes	No
Noise_term`_phnoise_cycloflicker`	Yes	Yes
Noise_term`_phnoise_fdep`	The union of `phnoise_Flicker` and `phnoise_cycloflicker` noise types	
Noise_term`_phnoise_findep`	The union of `phnoise_stationary` and `phnoise_cyclostationary` noise types	

Example

```
.PROBE PHASENOISE la_phnoise_cyclostationary
```

You can also show AM/PM separation for individual noise elements. (To enable this capability, the `listsources` option of `.PHASENOISE` must `= on`. This example outputs the phase modulation noise associated only with Cyclo-stationary sources (i.e., sources that are bias dependent, but not frequency dependent).

Noise Element output is of the form *`Noise_term(element_name)`*, where Noise_term can be `phnoise`, `la`, `ltotal`, `onoise`, and *`element_name`* is a valid netlist element name.

Example

```
.PROBE PHASENOISE la(x1)
```

Output File Format

- File `*.printpn#`: Writes output from the `.PRINT` statement when using HB to obtain the steady state solution.
- File `*.pn#`: Writes output from the `.PROBE` statement when using HB to obtain the steady state solution.

- File `*.printsnpn#`: Writes output from the `.PRINT` statement when using SN to obtain the steady state solution.
- File `*.snpn#`: Writes output from the `.PROBE` statement when using SN to obtain the steady state solution.

The `.PHASENOISE` command line parameters `Listfreq`, `ListCount`, `Listfloor`, and `Listsources` control and list Noise source contributions sequentially.The `listsources` argument must `=on` to generate a noise list block is for each output parameter specified in the `.PRINT`/`.PROBE` statement e.g., `phnoise`, `la`, `ltotal`, `onoise`.

.MEASURE Syntax and File Format

`.MEASURE PHASENOISE` extends output variables to the set: `am[noise] pm[noise]`

Measure File Format

- File `*.mpn#`: Writes output from the .MEASURE statement when using HB to obtain the steady state solution.
- File `*.msnpn#`: Writes output from the .MEASURE statement when using SN to obtain the steady state solution.

Interpreting Phase Noise Analysis Results

A typical phase noise plot consists of a line, which drops off as a function of frequency, at a slope of -20dbc/decade where white noise dominates, or -30dbc/decade where flicker noise dominates. At very low offset frequencies, the phase noise rolls off at according to a Lorentzian shape, such that it never exceeds 0 dbc/Hz even for very low offset frequencies. The 0 dbc/Hz value represents the power of the carrier oscillation, at 0 offset frequency. At very high offset frequencies, the slope can deviate from -20 dbc/decade due to the existence of a noise floor or a circuit feedback effect.

Numerical methods for phase noise analysis have limitations. The main limitation in the PAC phase noise algorithm is that it rolls off too quickly at low offset frequencies. In the low frequency region, you can trust NLP. The main limitation of the NLP algorithm is that it does not cover all high frequency effects, so you can trust PAC in the high frequency region.

The BPN algorithm attempts to combine the NLP and PAC results to generate a single result that is valid for all offset frequencies. It may fail if it cannot identify an overlap region where NLP and PAC results match. If the tool can not find an overlap region you should attempt to increase nharms on the .HBOSC command, as this increases the accuracy of both algorithms, especially PAC. PAC accuracy is more sensitive to nharms than NLP.

If you suspect the phase noise results to be inaccurate, check the following:

1. Is the .HBOSC steady state solution fully converged?

   Explanation: The NLP or PAC small-signal noise analysis requires a highly accurate steady state solution.

2. Did the phase noise analysis fully converge?

   Explanation: Phase noise analysis uses a GMRES iterative linear solver. If this iterative solver reaches its iteration limit before full convergence, the results are not reliable. Check the number of Krylov iterations that the phase noise analysis required. If it took the maximum number of iterations (as set by PHASENOISE_KRYLOV_ITR, default=1000), then the results did not fully converge and you should not trust them.
   You can use the options PHASENOISE_KRYLOV_DIM, PHASENOISE_KRYLOV_TOL, and PHASENOISE_KRYLOV_ITR to control the GMRES solver. You can increase PHASENOISE_KRYLOV_DIM to improve the convergence rate at the expense of memory, or increase PHASENOISE_KRYLOV_ITR to allow more iterations.

Important Note for AM/PM Users

There are discrepancies that may occur between this feature and the traditional PAC phase noise analysis in HSPICE advanced analog analyses. Total phase noise (i.e., `ltotal`) is the sum of two terms, the amplitude modulation (am) and phase modulation (phnoise). Traditionally, PAC phase noise reports the `phnoise` (phase modulation component) and `ltotal` (total phase noise) terms as identical, with the assumption that the am term (amplitude modulation component) was zero.

The PAC phase noise am/pm feature described in this section separately calculates the am and phnoise components. The am/pm feature affects all phase noise measurements that involve either PAC or BPN. In most cases the differences between `PHNOISEAMPM=1` and `=0` are small unless you expect a significant AM component. You may see a slight decrease in the new phase noise (phase-modulation) component when compared to the old calculation.

For example, the random jitter calculations are accurate only when they involve the PM component of phase noise. When basing calculation on `ltotal`, or you include AM noise, the process may introduce a small error.

NLP phase noise (`method=0`) only calculates the phnoise component; the am/pm option does not affect this method.

BPN phase noise (`method=2`) is affected in that the far side component is derived from the PAC phase noise.

Workaround: Ensure that `.OPTION PHNOISEAMPM=0` (the default). This assures that Periodic AC phase noise amplitude-modulation (AM) component is zero to maintain backward compatibility and traditional results for phnoise and jitter.

Accumulated Jitter Measurement for Closed Loop PLL Analysis

Enhancements to HSPICE advanced analog analyses include considerable support for a variety of jitter measurements. Many of these are important in a PLL flow, where you use the HBOSC or SNOSC analyses to compute a phase noise response for an oscillator or VCO, to derive the resulting random jitter from phase noise. In the PLL methodology, you use other HSPICE advanced analog analyses to compute phase noise contributions from the other PLL building blocks. You then perform a closed loop analysis by using phase-domain models for both signal and noise responses that takes into account the noise contributions from all such blocks. To complete this flow is the ability to compute “Accumulated Jitter” or “Timing Jitter” for the closed loop PLL. The accumulated jitter response is essentially an integral transformation of the closed-loop PLL response. The following sections show how you can measure accumulated jitter directly from the phase noise-output of an open loop oscillator analysis.

In the PLL flow, you interpret the closed loop phase noise from the results of a linear HSPICE advanced analog .AC/.NOISE analysis. The following sections describe a capability that allows direct measurement of accumulated jitter from the results of this closed loop noise analysis, without any special interpretation of the results.

This section covers the following topics:

- Jitter Measurements from Phase Noise

Jitter Measurements from Phase Noise

These sections discuss the following topics:

- Jitter Definitions
- Jitter Output Syntax
- .MEASURE Statements for Jitter
- Peak-to-Peak Jitter

Jitter Definitions

HSPICE advanced analog analyses provides several random jitter (RJ) measurements. This section defines, describes, and compares the various jitter measurements provided. You derive random jitter measurements from the results of an HSPICE advanced analog phase noise analysis. The following presents the relationships between phase noise and the random jitter measurements, and their means for calculation. The types of random jitter measurements include: Timing, Phase, Period, Tracking, Long-Term, and Cycle-to-Cycle Jitter.

Timing jitter is a measurement of oscillator uncertainty in the time domain. For clock applications, time domain measurements are preferable, since most specifications of concern involve time domain values.

Timing jitter is the standard deviation of the timing uncertainty, which is a function of the auto-correlation function in the power spectrum of the phase variations. Timing Jitter is the square root of the variance (standard deviation squared) of the timing uncertainty between two clock edges separated by an interval given by $\tau = N \cdot T_o$, where T_o is the ideal clock period. You can write it as a function of the auto-correlation function of the power spectrum of phase variations as:

Equation 24

$$\sigma^2_{TIE}(\tau) = \frac{2}{\omega_o^2}[R_\phi(0) - R_\phi(\tau)]$$

where TIE refers to the Time Interval Error. Call this measurement *Timing Jitter*, *Accumulated Jitter*, or *N-Cycle Jitter*, since it represents the jitter that may accumulate over an interval of many periods.

The Weiner-Khintchine Theorem [1] relates the auto correlation function to the power spectrum of phase variations as in the following equation:

Equation 25

$$R_\phi(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_\phi(\omega)e^{j\omega\tau}d\omega = 2\int_0^{\infty} L(f)\cos(2\pi f\tau)df$$

where $S_\phi(\omega)$ is the double-sided power spectrum of phase variations, and $L(f)$ is the single-sideband phase noise. The auto-correlation for $\tau = 0$ is given by:

Equation 26

$$R_\phi(0) \equiv \phi_{rms}^2 = 2\int_0^{\infty} L(f)df$$

which defines ϕ_{rms} in HSPICE advanced analog analyses known as RMS Phase Jitter.

Using the identity $2\sin^2\alpha = 1 - \cos 2\alpha$ we can then write:

Equation 27

$$\sigma_{TIE}^2(\tau) = \frac{8}{\omega_o^2}\int_0^{\infty} L(f)\sin^2(\pi f\tau)df$$

to enable currently supported HSPICE advanced analog analyses jitter measurements to be written as:

Equation 28

$$\phi_{rms} = \sigma_{ph}\cdot\omega_0 = \sqrt{2\int_0^{\infty} L(f)df} \qquad \text{“RMS Phase Jitter”}$$

$$\sigma_{TIE}(\tau) = \frac{2}{\omega_0}\sqrt{2\int_0^{\infty} L(f)sin^2(\pi f\tau)df} \qquad \text{“Timing (Time Interval Error) Jitter”}$$

From these definitions, several other key jitter measurements can be derived, including Period Jitter, Tracking Jitter, Long-Term Jitter, and Cycle-to-Cycle Jitter.

Period Jitter is equivalent to the value for Timing Jitter for a one period interval. We therefore have:

Equation 29

$$\sigma_{PER} = \sigma_{TIE}(T_0) = \frac{2}{\omega_0}\sqrt{2\int_0^{\infty} L(f) sin^2(\pi f T_0) df} \qquad \text{"Period Jitter"}$$

Tracking Jitter is equivalent to the value (in units of seconds) for RMS Phase Jitter, or:

Equation 30

$$\sigma_{tr} = \sigma_{ph} = \frac{\phi_{rms}}{\omega_0} = \frac{1}{\omega_0}\sqrt{2\int_0^{\infty} L(f) df} \quad \text{“Tracking Jitter”}$$

Long-Term Jitter is equivalent to $\sqrt{2}$ times the Tracking Jitter, i.e.:

Equation 31

$$\sigma_{\Delta T \to \infty} = \sigma_{TIE}(\tau \to \infty) = \sqrt{2}\frac{\phi_{rms}}{\omega_0} = \frac{2}{\omega_0}\sqrt{\int_0^{\infty} L(f)\, df} \quad \text{“Long-Term Jitter”}$$

Cycle-to-Cycle Jitter is based on the difference between adjacent Period Jitter measurements. It is given by:

Equation 32

$$\sigma_{CTC} = \sqrt{4\sigma^2_{PER} - \sigma^2_{TIE}\ (2T_0)} \quad \text{“Cycle-to-Cycle Jitter”}$$

In general, each of the above calculations must be performed carefully over limits of integration to accurately calculate jitter expressions based on the finite frequency limits provided for the phase noise analysis. Linear interpolation is used, but the phase noise generally follows more of a power law expansion.

Jitter Output Syntax

The timing jitter calculations are derived from the results of phase noise analysis. The phase noise output syntax supports the `JITTER` keyword as an output keyword in addition to the `PHNOISE` keyword.

```
.PRINT PHASENOISE PHNOISE JITTER
.PROBE PHASENOISE PHNOISE JITTER
```

If the `JITTER` keyword is present, the `.PHASENOISE` statement also outputs the raw jitter data to `*.jt0` and `*.printjt0` data files. The PHNOISE data is given in units of dBc/Hz, i.e., dB relative to the carrier, per Hz, across the output nodes specified by the PHASENOISE statement. The data plot is a function of offset frequency. If the JITTER keyword is present, .PHASENOISE outputs the TIE Timing Jitter (Accumulated Jitter) data to `*.jt0` and `*.printjt0` data files. These data are plotted as a function of time in units of seconds. The jitter calculations make use of some of the parameters given in the .PHASENOISE syntax.

The time samples for timing jitter output make use of the same number of points as the phase noise frequency sweep specification.

The output of timing jitter information uses a corresponding time sampling derived via:

Equation 33

$$\tau_1 = \frac{1}{T_0}, \tau_2 = \frac{2}{T_0}, ..., \tau_N = \frac{N}{T_0}$$

.MEASURE Statements for Jitter

The jitter-specific `.MEASURE` statements specify the jitter keywords as follows. (For discussion of the BER parameter, see below.)

```
.MEASURE PHASENOISE Jname PERJITTER phnoise
+ [UNITS=(sec|rad|UI)] [BER=val]
.MEASURE PHASENOISE Jname CTCJITTER phnoise
+ [UNITS=(sec|rad|UI)] [BER=val]
.MEASURE PHASENOISE Jname RMSJITTER phnoise
+ [FROM start_frequency] [TO end_frequency]
+ [UNITS=(sec|rad|UI)] [BER=val]
.MEASURE PHASENOISE Jname PHJITTER phnoise
+ [FROM start_frequency] [TO end_frequency]
+ [UNITS=(sec|rad|UI)] [BER=val]
.MEASURE PHASENOISE Jname TRJITTER phnoise
+ [FROM start_frequency] [TO end_frequency]
+ [UNITS=(sec|rad|UI)] [BER=val]
.MEASURE PHASENOISE Jname LTJITTER phnoise
+ [FROM start_frequency] [TO end_frequency]
+ [UNITS=(sec|rad|UI)] [BER=val]
```

RMSJITTER, PHJITTER, and TRJITTER are synonymous measurements, all based on the calculations described related to the RMS Phase Jitter value in units of seconds given by $\sigma_{ph} = \phi_{rms}/\omega_0$. These measurements allow control of the integration range using the FROM and TO parameters. The measurements for PERJITTER, and CTCJITTER use the full offset frequency sweep range given for the phase noise analysis to compute values (the FROM and TO parameters are ignored if entered).

As given currently in HSPICE advanced analog analyses, the frequency intervals can be modified for these jitter calculations (if desired, although not recommended), and UNITS can be selected between seconds, radians, and Unit Intervals. The following table specifies the calculation used for units=seconds for each jitter measurement.

MEASURE name	Calculation used (Units=sec)
RMSJITTER	$\sigma_{ph} = \phi_{rms}/\omega_0$
PHJITTER	$\sigma_{ph} = \phi_{rms}/\omega_0$
TRJITTER	$\sigma_{ph} = \phi_{rms}/\omega_0$
PERJITTER	σ_{PER}
LTJITTER	$\sigma_{\Delta T \to \infty} = \sqrt{2}\phi_{rms}/\omega_o$
CTCJITTER	σ_{CTC}

Example:

```
.meas phasenoise rj RMSJITTER phnoise from 1K to 100K
+ units = rad
```

Peak-to-Peak Jitter

As noted in .MEASURE Statements for Jitter, an additional BER (Bit Error Rate) parameter is supported. This parameter allows you to convert any jitter value from an RMS value into a Peak-to-Peak value. The RMS jitter values correspond to a 1-sigma standard deviation value for the Gaussian distribution of the jitter. Peak-to-peak values represent the full span of the Gaussian distribution. Since this span is theoretically unbounded for truly random distributions, the conversion to peak-to-peak values has to be interpreted as spanning some number of sigma values. You can arrive at this number (in other words, “sigma multiplier”) by specifying a corresponding Bit Error Rate.

The term “BER” corresponds to the unitless Bit Error Rate that allows for this conversion. The following table shows some example conversions from various BER values into a “sigma multiplier” value which corresponds to the number of sigma standard deviations in converting from RMS to peak-to-peak values:

Bit Error Rate	**Sigma Multiplier**
10^{-3}	6.180
10^{-4}	7.438
10^{-5}	8.530
10^{-6}	9.507
10^{-7}	10.399
10^{-8}	11.224
10^{-9}	11.996
10^{-10}	12.723
10^{-11}	13.412
10^{-12}	14.069
10^{-13}	14.698
10^{-14}	15.301
10^{-15}	15.883
10^{-16}	16.444

These conversions are done in accordance with the relationship:

Equation 34

$$\frac{1}{2}erfc\frac{\alpha}{2 \cdot \sqrt{2}} = \mathrm{BER}$$

where, $erfc$ is the complementary error function, and α is the Sigma Multiplier. Support for peak-to-peak conversions is included for a continuous range of

BER values from $10^{-16} \leq \alpha \leq 10^{-3}$(and some values extrapolated outside this range).

Specification of the BER parameter results in the output of the peak-to-peak jitter value, and not the RMS value. Labels for the measurements show appropriate “rms” and “p-p” labels. A BER parameter set to BER=0 is equivalent to having no parameter, and only results in the RMS calculation.

Errors/Warnings

Error handling and recovery is exercised to capture obvious errors in input specifications. The following error checking is performed:

- Calculations are be performed if oscillator or phase noise analysis fails.
- ERROR if L(f) > 1 over any part of the frequency sweep (non-dB form).
- ERROR if L(f) < 0 over any part of the frequency sweep (non-dB form).
- Error if any time or frequency samples are negative values.
- ERROR if BER < 0 for any Jitter measurement.
- WARNING if BER > 1 for any Jitter measurement.
- WARNING if f0 < 10 Hz. Message: “Jitter calculations may be ineffective for offset frequencies under 10 Hz.”

Clock Source with Random Jitter

For `.AC`-related `.NOISE` analysis, see Clock Source with Random Jitter in the *HSPICE User Guide: Basic Simulation and Analysis*.

Small-Signal Phase-Domain Noise Analysis (.ACPHASENOISE)

To see the influence that oscillator or VCO phase noise can have on a system where it is present, it is necessary to perform a phase-domain analysis where the circuit variables are phase, and the input noise stimuli are phase noise. This is the purpose of the .ACPHASENOISE analysis in HSPICE advanced analog analyses.

This type of analysis is critical, for example, in analyzing the effects of noise in a phase-locked loop (PLL). In a PLL design flow, the HBOSC or SNOSC analyses are used to compute a phase noise response for an oscillator or VCO. HSPICE advanced analog analyses can be used to compute phase noise contributions from the other PLL building blocks. A closed loop PLL analysis can then be performed by using phase-domain models for both signal and noise responses, where the noise contributions from all blocks are input as phase noise stimuli. Such an analysis can be performed to determine the PLL closed-loop phase noise, based on the contributions of each block, determined by an open loop analysis.

For more information on control options, see .ACPHASENOISE command in the *HSPICE Reference Manual: Commands and Control Options*.

This section covers the following topics:

- ACPHASENOISE Analysis .PRINT and .PROBE Syntax

ACPHASENOISE Analysis .PRINT and .PROBE Syntax

The unique aspect of the `.ACPHASENOISE` analysis is that it allows the small signal noise calculation results to be interpreted as phase noise values. The available `.print/.probe` measurements reflect this. The `.print/.probe` output syntax are the “`JITTER`” and “`PHNOISE`” keywords consistent with the HSPICE advanced analog `.phasenoise` analysis, namely:

```
.PRINT ACPHASENOISE PHNOISE JITTER
.PROBE ACPHASENOISE PHNOISE JITTER
```

As with the `.PHASENOISE` analysis, the `.ACPHASENOISE` analysis outputs raw data to `*.pn0` and `*.printpn0` files. The `PHNOISE` data is given in units of dBc/Hz, i.e., dB relative to the carrier, per Hz, across the output nodes specified by the `.ACPHASENOISE` statement. The data plot is a function of offset frequency. If the “`JITTER`” keyword is present, `.ACPHASENOISE` also outputs the accumulated TIE jitter data to `*.jt0` and `*.printjt0` data files. These data are plotted as a function of time in units of seconds. The Timing Jitter data itself has units of seconds. The timing jitter calculations make use of the parameters given in the `.ACPHASENOISE` syntax, such as “`freq`” and “`interval`”.

See also Using Noise Analysis Results as Input Noise Sources.

.MEASURE Support for ACPHASENOISE

Single valued jitter measurements are available from `.MEASURE` statements. Examples include period jitter, cycle-to-cycle jitter, and phase jitter measurements, respectively, as shown below:

```
.MEASURE ACPHASENOISE Jname PERJITTER phnoise
+ [UNITS=(sec|rad|UI)] [BER=val]

.MEASURE ACPHASENOISE Jname CTCJITTER phnoise
+ [UNITS=(sec|rad|UI)] [BER=val

.MEASURE ACPHASENOISE Jname PHJITTER phnoise
+ [FROM start_frequency [TO end_frequency]
+ [UNITS=(sec|rad|UI)] [BER=val]
```

Behavioral Noise Sources

In HSPICE advanced analog analyses, 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/(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 advanced analog analyses 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 * on H1*H2* represents the complex conjugate of H1 and H2.

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 the 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 returns v2.

This section covers the following topics:

- Using Noise Analysis Results as Input Noise Sources
- Power Supply Current and Voltage Noise Sources

Using Noise Analysis Results as Input Noise Sources

SN phase noise and phase noise analyses can output simulation results as ASCII data in `*.printsnpn0` files for SNOSC and SNNOISE. By extending the E and G voltage-controlled source syntax, the phase noise data in ASCII phase noise files can used as input for specifying behavioral noise sources

Usage Model

The syntax for the voltage controlled voltage (E) or current (G) source is as follows:

```
Exxx node1 node2 noisefile='filename' [mname='measname']
Gxxx node1 node2 noisefile='filename' [mname='measname']
```

Where,

`noisefile='`*`filename`*`'` is the name of the ASCII phase noise data file. The file name is typically designated as '*`design`*`.printsnpn0`', for a .SNOSC phase noise analysis or .SNNOISE analysis. But it also supports .PHASENOISE, .HBNOISE, .NOISE, and .ACPHASENOISE outputs.

`mname='`*`measname`*`'` is used to select the appropriate noise measurement name to be taken from the `*.printpn0` 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 analysis

The following syntaxes are supported in HSPICE advanced analog analyses:

- `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

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
```

References

[1] E. Ngoya, A. Suarez, R. Sommet, R. Quere, “Steady State Analysis of Free or Forced Oscillators by Harmonic Balance and Stability Investigation of Periodic and Quasi-Periodic Regimes,” *International Journal of Microwave and Millimeter-Wave Computer-Aided Engineering*, Volume 5, Number 3, pages 210-223 (1995)

[2] C.R. Chang, M.B. Steer, S. Martin, E. Reese, “Computer-Aided Analysis of Free-Running Microwave Oscillators,” *IEEE Trans. on Microwave Theory and Techniques*, Volume 39, No. 10, pages 1735-1745, October 1991.

[3] G.D. Vendelin, *Design of Amplifiers and Oscillators by the S-Parameter Method*, John Wiley & Sons, 1982

[4] A. Demir, A. Mehrotra, J. Roychowdhury, “Phase Noise in Oscillators: A Unifying Theory and Numerical Methods for Characterization” in Proc. IEEE DAC, pages 26-31, June 1998.

[5] A. Demir, A. Mehrotra, and J. Roychowdhury, “Phase Noise in Oscillators: A Unifying Theory and Numerical Methods for Characterization,” *IEEE Trans. Circuits System I*, Volume 47, pages 655–674, May 2000.

[6] A. van der Ziel, Noise in Solid State Devices and Circuits, John Wiley & Sons, 1986.

[7] A. Hajimiri, S. Limotyrakis, and T.H. Lee, “Jitter and phase noise in ring oscillators,” IEEE J. Solid-State Circuits, vol. 34, no. 6, pp. 790-804, June 1999.

[8] Jitter Analysis Techniques for High Data Rates, Application Note 1432, Agilent Technologies, Feb. 2003.

[9] Characterization of Clocks and Oscillators, NIST Technical Note 1337, National Institute of Standards and Technology.

[10] G.V. Klimovitch, “Near-carrier oscillator spectrum due to flicker and white noise,” Proc. ISCAS 2000 (Geneva), May 2000.

5

Large Signal Periodic AC, Transfer Function, and Noise Analyses

Describes how to use both harmonic balance-based and Shooting Newton-based AC, and transfer function analyses, as well as nonlinear, steady-state noise analysis.

The following topics are presented in this section:

- Multitone Harmonic Balance AC Analysis (.HBAC)
- Shooting Newton AC Analysis (.SNAC)
- Multitone Harmonic Balance Noise (.HBNOISE)
- Shooting Newton Noise Analysis (.SNNOISE)
- Periodic Time-Dependent Noise Analysis (.PTDNOISE)
- Multitone Harmonic Balance Transfer Function Analysis (.HBXF)
- Shooting Newton Transfer Function Analysis (.SNXF)

Multitone Harmonic Balance AC Analysis (.HBAC)

You use the `.HBAC` (Harmonic Balance AC) statement for analyzing linear behavior in large-signal periodic systems. The `.HBAC` statement uses a periodic AC (PAC) algorithm to perform linear analysis of autonomous (oscillator) or non-autonomous (driven) circuits, where the linear coefficients are modulated by a periodic, steady-state signal.

Multitone HBAC analysis extends single-tone HBAC to quasi-periodic systems with more than one periodic, steady-state tone. One application of multitone HBAC is to more efficiently determine mixer conversion gain under the influence of a strong interfering signal than is possible by running a swept three-tone HB simulation.

The following sections discuss these topics:

- Prerequisites and Limitations
- Input Syntax
- Output Syntax
- HBAC Output Data Files
- Using the .MEASURE Command with .HBAC
- Errors and Warnings
- Tutorial Example - Using Multi-Tone HB and HBAC Analyses for a Mixer

Prerequisites and Limitations

The following prerequisites and limitations apply to HBAC:

- Requires one and only one `.HBAC` statement. If you use multiple `.HBAC` statements, HSPICE advanced analog analyses uses only the last `.HBAC` statement.
- Requires one and only one `.HB` statement.
- Supports arbitrary number of tones.
- Requires placing the parameter sweep in the `.HB` statement.
- Requires at least one HB source.
- Requires at least one HBAC source.
- Supports unlimited number of HB and HBAC sources.
- The requested maximum harmonic in a `.PROBE` or `.PRINT` statement must be less than or equal to half the number of harmonics specified in harmonic balance (that is, max_harm <= num_hb_harms / 2).

Input Syntax

```
.HBAC frequency_sweep
```

Parameter	Description
frequency_sweep	Frequency sweep range for the input signal (also referred to as the input frequency band (IFB) or fin). You can specify LIN, DEC, OCT, POI, or SWEEPBLOCK. Specify the nsteps, start, and stop frequencies using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ SWEEPBLOCK=swblockname ■ DATA=*dataname*

HBAC Analysis Options

The following options directly relate to a HBAC analysis and override the corresponding PAC options if specified in the netlist:

- `.OPTION HBACTOL`, default = 1x10-8, Range = 1x10-14 to Infinity
- `.OPTION HBACKRYLOVDIM`, default = 300, Range = 1 to Infinity
- `.OPTION HBACKRYLOVITER | HBAC_KRYLOV_ITER`, default = 1000, Range = 1 to Infinity

If these parameters are not specified, then the following conditions apply:

- If `HBACTOL` > `HBTOL`, then `HBACTOL` = `HBTOL`
- If `HBACKRYLOVDIM` < `HBKRYLOVDIM`, then `HBACKRYLOVDIM` = `HBKRYLOVDIM`

Output Syntax

This section describes the syntax for the HBAC `.PRINT` and `.PROBE` statements. These statements are similar to those used for HB analysis.

.PRINT and .PROBE Statements

```
.PRINT HB TYPE(NODES | ELEM)[INDICES]
.PROBE HB TYPE(NODES | ELEM)[INDICES]
```

Parameter	Description
TYPE	Specifies a harmonic type node or element. TYPE can be one of the following: ▪ Voltage type – V = voltage magnitude and phase in degrees VR = real component VI = imaginary component VM = magnitude VP - Phase in degrees VPD - Phase in degrees VPR - Phase in radians VDB - dB units VDBM - dB relative to 1 mV ▪ Current type – I = current magnitude and phase in degrees IR = real component II = imaginary component IM = magnitude IP - Phase in degrees IPD - Phase in degrees IPR - Phase in radians IDB - dB units IDBM - dB relative to 1 mV ▪ Power type – P ▪ Frequency type – hertz[index], hertz[index1, index2, ...] You must specify the harmonic index for the hertz variable. The frequency of the specified harmonics is dumped.
NODES | ELEM	NODES or ELEM can be one of the following: ▪ Voltage type – a single node name (n1), or a pair of node names, (n1,n2) ▪ Current type – an element name (elemname) ▪ Power type – a resistor (resistorname) or port (portname) element name ▪ Frequency type – the harmonic index for the hertz variable. The frequency of the specified harmonics is dumped.
INDICES	Index to tones in the form [n1, n2, ..., nK, +/-1]. ▪ nj is the index of the j-th HB tone and the .HB statement contains K tones ▪ +/-1 is the index of the HBAC tone Wildcards are not supported if this parameter is used. You can transform HB data into the time domain and output by using the following syntax:.PRINT HBTRAN ov1 [ov2 ... ovN].PROBE HBTRAN ov1 [ov2 ... ovN]. See TYPE above for voltage and current type definitions.

HBAC Output Data Files

An HBAC analysis produces these output data files:

- Output from the `.PRINT` statement is written to a `.printhb#` file. This data is against the IFB points.
  - The header contains the large-signal fundamental and the range of small-signal frequencies.
  - The columns of data are labeled as F(Hz), followed by the output variable names. Each variable name has the associated mixing pair value appended.

    All *N* variable names and all *M* mixing pair values are printed for each swept small-signal frequency value (a total of N*M for each frequency value).
- Output from the `.PROBE` statement is written to a .hb# file. This data is against the IFB points.
- Reported performance log statistics are written to a .lis file:
  - Number of nodes
  - Number of FFT points
  - Number of equations
  - Memory in use
  - CPU time
  - Maximum Krylov iterations
  - Maximum Krylov dimension
  - Target GMRES residual
  - GMRES residual
  - Actual Krylov iterations taken
  - Frequency (swept input frequency values).

Using the .MEASURE Command with .HBAC

Since `.HBAC` requires an `.HB` analysis, the measure statements for this analysis are the same as for `.HB` analysis. For example:

```
.MEASURE HB result FIND out_var AT=val
```

Errors and Warnings

The following error and warning messages are used when HSPICE encounters a problem with a HBAC analysis.

Error Messages

HBAC frequency sweep includes negative frequencies. HBAC allows only frequencies that are greater than or equal to zero.

No HB statement is specified (error at parser). HBAC requires an HB statement to generate the steady-state solution.

Warning Messages

More than one HBAC statement (warning at parser). HSPICE advanced analog analyses uses only the last HBAC statement in the netlist.

No HBAC sources are specified (error at parser). HBAC requires at least one HBAC source.

GMRES Convergence Failure. When GMRES (Generalized Minimum Residual) reaches the maximum number of iterations and the residual is greater than the specified tolerance. The HBAC analysis generates a warning and then continue as if the data were valid. This warning reports the following information:

- Final GMRES Residual
- Target GMRES Residual
- Maximum Krylov Iterations
- Actual Krylov Iterations taken

Tutorial Example - Using Multi-Tone HB and HBAC Analyses for a Mixer

The example in this section shows how to use HSPICE advanced analog analyses to analyze a circuit driven by multiple input stimuli with different frequencies. Mixer circuits provide a typical example of this scenario: in this case, there might be two input signals (LO and RF), which are mixed to produce an IF output signal. In this case, HSPICE advanced analog analyses offers two options:

- Multi-tone HB analysis: specify the `LO` and `RF` base frequencies as two separate tones on the `.HB` command.
- Periodic AC analysis (HBAC): if one of the inputs is a small-signal, you can use a faster linear analysis to analyze its effect. For example, if a mixer's `LO` is a large signal, but `RF` is a small signal, a single-tone HB analysis using the `LO` frequency can be combined with HBAC in place of a 2-tone HB analysis.

To demonstrate both techniques, this example analyzes an ideal mixer built using behavioral elements. It is based on demonstration netlist `mix_tran.sp`, which is available in directory `$`*`installdir`*`/demo/hspice/rf_examples/`.

```
* Ideal mixer example: transient analysis
.OPTIONS POST
vlo lo 0 1.0 sin (1.0 0.5 1.0g 0 0 90)
rrf1 rf1 rf 1.0
g1 0 if cur='1.0*v(lo)*v(rf)' $ mixer element
c1 0 if q='1.0e-9*v(lo)*v(rf)' $ mixer element
rout if ifg 1.0
vctrl ifg 0 0.0
h1 out 0 vctrl 1.0 $ convert I to V
rh1 out 0 1.0
vrf rf1 0 sin (0 0.001 0.8GHz 0 0 114)
.tran 10p 10n
.opt sim_accuracy=100
.end
```

This example uses behavioral controlled current and charge sources to simulate a mixer. The `LO` signal is driven by a 0.5 Volt sinusoid at 1 GHz, and `RF` is driven by a 10mV signal at 800 MHz. The mixer output is the voltage at node out, v(out).

Two-tone HB Approach

To analyze this circuit using 2-tone HB, add:

- HB source for `LO`: add `HB 0.5 0 1 1` to the LO voltage source; this sets the amplitude to 0.5, no phase shift for the first harmonic of the first tone, which is 1 GHz.
- HB source for `RF`: add `HB 0.001 24 1 2` to the RF voltage source; this sets the amplitude to 0.001, 24 degrees phase shift for the first harmonic of the second tone (0.8 GHz).
- An `.HB` command specifying both tones: .hb tones=1g 0.8g nharms=6 3; only a small number of harmonics is required to resolve the signals.

The complete `mix_hb.sp` netlist for 2-tone HB analysis is then:

```
* Ideal mixer example: 2-tone HB analysis
.OPTIONS POST
vlo lo 0 1.0 sin (1.0 0.5 1.0g 0 0 90) HB 0.5 0 1 1
rrf1 rf1 rf 1.0
g1 0 if cur='1.0*v(lo)*v(rf)' $ mixer element
c1 0 if q='1.0e-9*v(lo)*v(rf)' $ mixer element
rout if ifg 1.0
vctrl ifg 0 0.0
h1 out 0 vctrl 1.0 $ convert I to V
rh1 out 0 1.0
vrf rf1 0 sin (0 0.001 0.8GHz 0 0 114) HB 0.001 24 1 2
.opt sim_accuracy=100
.hb tones=1g 0.8g nharms=6 3
.end
```

This example is available in directory `$<installdir>/demo/hspice/rf_examples/`.

HBAC Approach

To analyze this circuit using HBAC, start with the 2-tone HB analysis setup, and modify it as follows:

- Replace the `RF` HB signal with an HBAC signal: change `HB 0.001 24 1 2` to HBAC 0.001 24; this deactivates the source for HB and activates it for HBAC with the same magnitude and phase.
- Specify the frequency in the `.HBAC` command.
- Change the `.HB` command to single tone:

  ```
  .HB tones=1g nharms=6
  ```

  HBAC takes care of the second tone.
- Add a `.HBAC` command

```
.HBAC lin 1 0.8g 0.8g
```

This command runs an analysis at a single frequency point, 0.8 GHz. In general, HBAC analysis can sweep the advanced analog frequency over a range of values.

The following is the complete `mix_hbac.sp` netlist for HBAC analysis of this simple mixer. This netlist also contains commands for performing periodic noise analysis. It is available in directory `$`*`installdir`*`/demo/hspice/rf_examples/`.

```
* Ideal mixer example: HBAC analysis
.OPTIONS POST
vlo lo 0 1.0 sin (1.0 0.5 1.0g 0 0 90)
+ HB 0.5 0 1 1
rrf1 rf1 rf 1.0
g1 0 if cur='1.0*v(lo)*v(rf)' $ mixer element
c1 0 if q='1.0e-9*v(lo)*v(rf)' $ mixer element
rout if ifg 1.0
vctrl ifg 0 0.0
h1 out 0 vctrl 1.0 $ convert I to V
rh1 out 0 1.0
vrf rf1 0 sin (0 0.001 0.8GHz 0 0 114)
+ HBAC 0.001 24
.opt sim_accuracy=100
.hb tones=1g nharms=6
.hbac lin 1 0.8g 0.8g
* Noise analysis
.hbnoise v(out) rrf1 lin 40 0.1g 4g
.print hbnoise onoise nf
.probe hbnoise onoise nf
.end
```

Comparing Results

After running all three netlists above, you will have generated 3 output files:

- mix_tran.tr0
- mix_hb.hb0
- mix_hbac.hb0

You can compare the results of the three analyses in Custom WaveView.

1. To run the netlists and start Custom WaveView, type:

```
hspicerf mix_tran.sp
hspicerf mix_hb.sp
hspicerf mix_hbac.sp
wv &
```

2. Use **File** > **Import Waveform File** and select the `mix_tran.tr0`, `mix_hb.hb0`, and `mix_hbac.hb0` files from the Open: Waveform Files dialog box. A histogram displays.
3. Select the v(out) signal from the `mix_hb.hb0` file in the signal browser. Double-click on the signal name or drag and drop the signal in the waveform. You should see a histogram similar to the one from the `mix_hb.hb0` file.
4. Convert the HB and HBAC histograms to time domain. In the waveform, right-click in the name area of the panel containing the signal v(out), left-click on the waveform label for v(out) from the `mix_hb.hb0` file. From the Panel menu, choose Signal 'v(out)' > To Time-Domain. To accept the defaults for range and interval, click OK in the Convert to Time domain window.
5. Repeat Step 4 for the v(out) signal from the `mix_hbac.hb0` file.
6. Use **Waveview** > **New** to open a new waveform.
7. Select the v(out) signal from the mix_tran.tr0 file in the signal browser. Double-click on the signal name or drag and drop the signal in the waveform.
8. Compare the time domain waveforms from the `mix_hb.hb0` and `mix_hbac.hb0` files with the time domain waveform from the `mix_tran.tr0` file. In the file browser, click on IFT under derived waveforms. The signals 0|v(out) and 1|v(out) should appear in the signal browser. Select the 0|v(out) and 1|v(out) signals and drag and drop them in the waveform. All three time domain signals should be displayed in the same panel. The three signals are almost indistinguishable from each other.

You can also use HBAC to perform noise analysis on advanced analog circuits by using the `.HBNOISE` command, which is included in the `mix_hbac.sp` netlist.

- The `.HBNOISE` command invokes noise analysis, identifying an output node where the noise is measured, an input noise source (in this case, rrf1) which serves as a reference for noise figure computation, and a frequency sweep for the noise analysis.
- The `.PRINT` and `.PROBE` `hbnoise` commands instruct HSPICE advanced analog analyses to save the output noise and noise figure at each frequency in the `mix_hbac.printpn0` and `mix_hbac.pn0` output files.

This ideal mixer is noiseless, except for the resistors at the input and output.

The `mix_hbac.lis` file contains detailed data on the individual noise source contributions of the resistors. You can view `mix_hbac.printpn0` to see the output noise and noise figure at each frequency. In WaveView, you can view `mix_hbac.pn0` to plot the output noise and noise figure data as a function of frequency.

Shooting Newton AC Analysis (.SNAC)

You use the Shooting Newton AC (`.SNAC`) statement for analyzing linear behavior in large-signal periodic systems. The `.SNAC` statement uses a periodic AC (PAC) and Shooting Newton algorithm to perform linear analysis of nonautonomous (driven) circuits, where the linear coefficients are modulated by a periodic, steady-state signal.

The following section describes the periodic AC analysis based on a Shooting Newton algorithm. This functionality is similar to the Harmonic Balance (HBAC) for periodic AC analysis.

The following section discuss these topics:

- Prerequisites and Limitations
- Input Syntax
- Output Syntax
- SNAC Output Data Files
- Using the .MEASURE Command with .SNAC
- Errors and Warnings
- SNAC Example

Prerequisites and Limitations

The following prerequisites and limitations apply to SNAC:

- Requires one and only one .SNAC statement. If you use multiple .SNAC statements, HSPICE advanced analog analyses uses only the last .SNAC statement.
- Requires one and only one .SN statement.

- Requires placing the parameter sweep in the .SN statement.
- Requires at least one Periodic source.
- Limited to simulations that can be reduced to a single tone SN analysis.
- Supports unlimited number of sources.
- The requested maximum harmonic in a .PROBE or .PRINT statement must be less than or equal to half the number of harmonics specified in the SN statement (that is, max_harm ≤ nharms / 2).

Input Syntax

```
.SNAC frequency_sweep
```

Parameter	Description
frequency_sweep	Frequency sweep range for the input signal (also referred to as the input frequency band (IFB) or fin). You can specify LIN, DEC, OCT, POI, or SWEEPBLOCK. Specify the nsteps, start, and stop frequencies using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ *SWEEPBLOCK=swblockname* ■ DATA=*dataname*

Output Syntax

This section describes the syntax for the SNAC .PRINT and .PROBE statements. These statements are similar to those used for HB analysis.

.PRINT and .PROBE Statements

```
.PRINT SN TYPE(NODES | ELEM)[INDICES]
.PROBE SN TYPE(NODES | ELEM)[INDICES]
```

Parameter	Description
TYPE	Specifies a harmonic type node or element. TYPE can be one of the following: ■ Voltage type – V = voltage magnitude and phase in degrees VR = real component VI = imaginary component VM = magnitude VP - Phase in degrees VPD - Phase in degrees VPR - Phase in radians VDB - dB units VDBM - dB relative to 1 mV ■ Current type – I = current magnitude and phase in degrees IR = real component II = imaginary component IM = magnitude IP - Phase in degrees IPD - Phase in degrees IPR - Phase in radians IDB - dB units IDBM - dB relative to 1 mV ■ Power type – P ■ Frequency type – hertz[index], hertz[index1, index2, ...] You must specify the harmonic index for the hertz variable. The frequency of the specified harmonics is dumped.
NODES | ELEM	NODES or ELEM can be one of the following: ■ Voltage type – a single node name (n1), or a pair of node names, (n1,n2) ■ Current type – an element name (elemname) ■ Power type – a resistor (resistorname) or port (portname) element name ■ Frequency type – the harmonic index for the hertz variable. The frequency of the specified harmonics is dumped.
INDICES	Index to tones in the form [n1, +/-1]. ■ n1 is the index of the SN tone ■ +/-1 is the index of the SNAC tone Wildcards are not supported if this parameter is used. You can transform SN data into the time domain and output by using the following syntax:.PRINT SNTRAN ov1 [ov2 ... ovN]. PROBE SNTRAN ov1 [ov2 ... ovN]. See TYPE above for voltage and current type definitions.

SNAC Output Data Files

A SNAC analysis produces these output data files:

- Output from the `.PRINT` statement is written to a `.printsnac#` file.
- This data is against the IFB points.
- The header contains the large-signal fundamental and the range of small-signal frequencies.
- The columns of data are labeled as F(Hz), followed by the output variable names. Each variable name has the associated mixing pair value appended. All N variable names and all M mixing pair values are printed for each swept small-signal frequency value (a total of N*M for each frequency value).
- Output from the `.PROBE` statement is written to a `.snac#` file.

Reported performance log statistics are written to a `.lis` file:

- Number of nodes
- Number of FFT points
- Number of equations
- Memory in use
- CPU time
- Maximum Krylov iterations
- Maximum Krylov dimension
- Target GMRES residual
- GMRES residual
- Actual Krylov iterations taken
- Frequency (swept input frequency values)

Using the .MEASURE Command with .SNAC

Since `.SNAC` requires an `.SN` analysis, the measure statements for this analysis are the same as for `.SN` analysis. For example,

```
.MEASURE SN result FIND out_var AT=val
```

Errors and Warnings

The following error and warning messages are used when HSPICE encounters a problem with a SNAC analysis.

Error Messages

SNAC frequency sweep includes negative frequencies. SNAC allows only frequencies that are greater than or equal to zero.

No SN statement is specified (error at parser). SNAC requires an SN statement to generate the steady-state solution.

Warning Messages

More than one SNAC statement (warning at parser). HSPICE advanced analog analyses uses only the last SNAC statement in the netlist.

No SNAC sources are specified (error at parser). SNAC requires at least one SNAC source.

GMRES Convergence Failure. When GMRES (Generalized Minimum Residual) reaches the maximum number of iterations and the residual is greater than the specified tolerance. The SNAC analysis generates a warning and then continue as if the data were valid. This warning reports the following information:

- Final GMRES Residual
- Target GMRES Residual
- Maximum Krylov Iterations
- Actual Krylov Iterations taken

SNAC Example

The following example is shipped with the HSPICE distribution as `mix_snac.sp` and is available in directory: `$`*`installdir`*`/demo/hspice/rf_examples/`.

```
* Test SNAC: ideal I,Q mixer -rrd
.OPTIONS PROBE
.OPTIONS POST=2
$.OPTIONS snmaxiter=100
.OPTIONS SNACCURACY=5
vlo lo 0 1.0 cos(1.0 0.5 1g) $ Periodic, Large-Signal SN Input
rlo lo 0 1.0
rrf rf 0 1.0 $ Noise source
rrf1 rf1 rf 1.0 $ Noise source
g1 0 if cur='1.0*v(lo)*v(rf)' $ mixer element
c1 0 if q='1.0e-9*v(lo)*v(rf)' $ mixer element
rout if ifg 1.0
vctrl ifg 0 0.0
h1 out 0 vctrl 1.0
rh1 out 0 1.0
vrf rf1 0 snac .001 24.0 $ Small signal for SNAC with 1-tone
SN Input
.sn tones=1.0g nharms=3
.snac lin 1 0.8g 0.8g
.print sn v(rf1) v(lo) v(out)
.print snfd v(rf1) v(lo) v(out)
.print snac v(rf1) v(lo) v(out)
.measure snac vout1 find v(out)[1,-1] at=0.8g
.measure snac vout2 find v(out)[0,1] at=0.8g
.measure snac vout3 find v(out)[1,1] at=0.8g
.measure sn vlo1 find v(lo) at=0.5n
.measure sn vlo2 find v(lo) at=1n
.measure snfd vlo3 find v(lo)[1] at=1
.end
```

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 / HBAC and SN/SNAC 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 in the *HSPICE User Guide: Basic Simulation and Analysis*.

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 don’t need to specify DC explicitly (default=0).
[ac [mag [phase]]]	AC voltage or power source value.
[HBAC [mag [phase]]]	Advanced analog HBAC voltage or power source value.
[SNAC [mag [phase]]]	Advanced analog SNAC voltage or power source value.
[hb [mag [phase [harm [tone [modharm [modtone]]]]]]]	Advanced analog HB voltage, current, or power source value. Multiple HB specifications with different harm, tone, modharm, and modtone values are allowed. ■ *phase* is in degrees ■ *harm* and *tone* are indices corresponding to the tones specified in the `.HB` statement. Indexing starts at 1 (corresponding to the first harmonic of a tone). ■ *modtone* and *modharm* specify sources for multi-tone simulation. A source specifies a tone and a harmonic, and up to 1 offset tone and harmonic (modtone for tones and modharm for harmonics). The signal is then described as: V(or I) = mag*cos(2*pi* (harm*tone+modharm*modtone)*t + phase)

Parameter	Description
[transient waveform]	(Transient analysis) Any one of waveforms: AM, EXP, PULSE, PWL, SFFM, or SIN. Multiple transient descriptions are not allowed.
[power=[0 | 1 | W | dbm]]	(HSPICE advanced analog analyses) Power Switch ■ When 0 (default), element treated as a voltage or current source. ■ When 1 or W, element treated as a power source, realized as a voltage source with a series impedance. In this case, the source value is interpreted as RMS available power in units of Watts. ■ When dbm, element treated as a power source in series with the port impedance. Values are in dbms. You can use this parameter for Transient analysis if the power source is either DC or SIN.
[z0=*val*]	(LIN analysis) System impedance used when converting to a power source, inserted in series with the voltage source. Currently, this only supports real impedance. ■ When power=0, z0 defaults to 0. ■ When power=1, z0 defaults to 50 ohms. You can also enter zo=*val*.
[rdc=*val*]	(DC analysis) Series resistance (overrides `z0`).
[rac=*val*]	(AC analysis) Series resistance (overrides `z0`).
[RHBAC=*val*]	(HSPICE advanced analog HBAC analysis) Series resistance (overrides z0).
[rhb=*val*]	(HSPICE advanced analog HB analysis) Series resistance (overrides z0).
[rtran=*val*]	(Transient analysis) Series resistance (overrides z0).

Parameter	Description
[TRANFORHB=[0|1]]	■ 0 (default): The transient description is ignored if an HB value is given or a DC value is given. If no DC or HB value is given and TRANFORHB=0, then HB treats the source as a DC source, and the DC source value is the time=0 value. ■ 1: HB analysis uses the transient description if its value is VMRF, SIN, PULSE, PWL, or LFSR. If the type is a non-repeating PWL source, then the time=infinity value is used as a DC source 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:

```
Vin=1.5*cos(2*pi*1.e9*t - 90*pi/180) V
```

Vrf = 0.2*cos(2*pi*1.1e9*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

`Vm` has the following time-domain description:

Vm = 0.5 + cos(2*pi*1.e8*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

`Vt` is converted 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

`Pt` delivers 1 mW of power through a 50-ohm impedance.

Phase Noise and Buffer Chains

Phase noise is specific to oscillators. However, for buffer chains you can do periodic noise analysis. Phase noise that may be contributed by buffer, amplifier, divider, or multiplier circuits is often referred to as *residual phase noise*. There are three commands in HSPICE advanced analog analyses that can apply help you predict such noise: .HBNOISE, .SNNOISE, and .PTDNOISE. These analyses compute the noise spectral density at an output variable taking into consideration modulation effects.

.HBNOISE and .SNNOISE compute the average noise over one period, assuming your circuit is driven by a periodic input signal.

.PTDNOISE computes the noise at one specific time point within the period, or it can compute the noise as a function of time over one period. .PTDNOISE can also convert the noise to a “jitter” value, which would be an uncertainty in the timing as opposed to the uncertainty in the output voltage.

See the following sections for detailed information on these commands:

- Multitone Harmonic Balance Noise (.HBNOISE)
- Shooting Newton Noise Analysis (.SNNOISE)
- Periodic Time-Dependent Noise Analysis (.PTDNOISE)

Multitone Harmonic Balance Noise (.HBNOISE)

An HBNOISE (Harmonic Balance noise) analysis simulates the noise behavior in periodic systems and is designed for use with driven circuits. It employs a Periodic AC (PAC) algorithm to perform noise analysis of nonautonomous (driven) circuits under periodic, steady-state tone conditions. This can be extended to quasi-periodic systems having more than one periodic, steady-state tone. One application for a multitone HBNOISE analysis is determining mixer noise figures under the influence of a strong interfering signal.

The PAC method simulates noise assuming that the stationary noise sources and/or the transfer function from the noise source to a specific output are periodically modulated.

- The modulated noise source (thermal, shot, or flicker) is modeled as a cyclostationary noise source.
- A PAC algorithm solves the modulated transfer function.
- You can also use the HBNOISE PAC method with correlated noise sources, including the MOSFET level 9 and level 11 models, and the behavioral noise source in the G-element (Voltage Dependent Current Source).

You use the `.HBNOISE` statement to perform a Periodic Noise Analysis.

The following sections discuss these topics:

- Supported Features
- Input Syntax
- Output Syntax
- Output Data Files
- Measuring HBNOISE Analyses with .MEASURE
- Errors and Warnings
- HBNOISE Example

Supported Features

HBNOISE supports the following features:

- All existing HSPICE advanced analog noise models.
- Uses more than one single-tone, harmonic balance to generate the steady-state solution.
- Unlimited number of HB sources (using the same tone, possibly multiple harmonics).
- Includes stationary, cyclostationary, frequency-dependent, and correlated noise effects.
- Swept parameter analysis.
- Results are independent of the number of HBAC sources in the netlist.

Prerequisites and Limitations

The following prerequisites and limitations apply to HBNOISE:

- Requires one `.HB` statement (which determines the steady-state solution).
- Requires at least one HB source or one TRANFORHB source.
- Requires placing the parameter sweep in the `.HB` statement.
- The requested maximum harmonic in `.HBNOISE` must be less than or equal to half the number of harmonics used in harmonic balance (that is, *max_harm* <= *num_hb_harms*/2).

Input Syntax

```
.HBNOISE [output] [insrc] [parameter_sweep]
+ [n1, n2, ..., nk,+/-1]
+ [listfreq=(frequencies|none|all)] [listcount=val]
+ [listfloor=val] [listsources=on|off]
```

Parameter	Description
output	Output node, pair of nodes, or 2-terminal element. HSPICE advanced analog analyses references equivalent noise output to this node (or pair of nodes). Specify a pair of nodes as V(n+,n-). If you specify only one node, V(n+), then HSPICE advanced analog analyses assumes that the second node is ground. You can also specify a 2-terminal element name that refers to an existing element in the netlist.
insrc	An input source. If this is a resistor, HSPICE advanced analog analyses uses it as a reference noise source to determine the noise figure. If the resistance value is 0, the result is an infinite noise figure.
parameter_sweep	Frequency sweep range for the input signal. Also referred to as the input frequency band (IFB) or *fin*). You can specify LIN, DEC, OCT, POI, SWEEPBLOCK, DATA, MONTE, or OPTIMIZE sweeps. Specify the nsteps, start, and stop frequencies using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ *SWEEPBLOCK=swblockname*
n1,n2,...,nk, +/-1	Index term defining the output frequency band (OFB or *fout*) at which the noise is evaluated. Generally, fout=ABS(n1*f1+n2*f2+...+nk*fk+/-fin) where: ■ f1,f2,...,fk are the first through k-th steady-state tones determined from the harmonic balance solution ■ fin is the IFB defined by *parameter_sweep*. The default index term is [1,1,...1,-1]. For a single tone analysis, the default mode is consistent with simulating a low-side, down conversion mixer where the advanced analog signal is specified by the IFB and the noise is measured at a down-converted frequency that the OFB specifies. In general, you can use the [n1,n2,...,nk,+/-1] index term to specify an arbitrary offset. The noise figure measurement is also dependent on this index term.

Parameter	Description

listfreq	Prints the element noise value to the .lis file. You can specify at which frequencies the element noise value is printed. The frequencies must match the sweep_frequency values defined in the *parameter_sweep*, otherwise they are ignored.
	
	In the element noise output, the elements that contribute the largest noise are printed first. The frequency values can be specified with the NONE or ALL keyword, which either prints no frequencies or every frequency defined in *parameter_sweep*. Frequency values must be enclosed in parentheses. For example:`listfreq=(none)`
	`listfreq=(all)`
	`listfreq=(1.0G)`
	`listfreq=(1.0G, 2.0G)` The default value is NONE.

listcount	Prints the element noise value to the .lis file, which is sorted from the largest to smallest value. You do not need to print every noise element; instead, you can define `listcount` to print the number of element noise frequencies. For example, `listcount=5` means that only the top 5 noise contributors are printed. The default value is 1.
listfloor	Prints the element noise value to the `.lis` file and defines a minimum meaningful noise value (in V/Hz$^{1/2}$ units). Only those elements with noise values larger than `listfloor` are printed. The default value is 1.0e-14 V/Hz$^{1/2}$.
listsources	Prints the element noise value to the `.lis` file when the element has multiple noise sources, such as a FET, which contains the thermal, shot, and 1/f noise sources. You can specify either ON or OFF: ON Prints the contribution from each noise source and OFF does not. The default value is OFF.

Output Syntax

The HSPICE advanced analog HB and SN noise analyses can output the output noise (onoise), noise figures (NF, SSNF and DSNF) and, the input referred noise (inoise). This section describes the syntax for the HBNOISE `.PRINT` and `.PROBE` statements.

.PRINT and .PROBE Statements

```
.PRINT HBNOISE [ONOISE] [NF] [SSNF] [DSNF] [INOISE]
```

```
.PROBE HBNOISE [ONOISE] [NF] [SSNF] [DSNF] [INOISE]
```

Parameter	Description
ONOISE	Outputs the voltage noise at the output frequency band (OFB) across the output nodes in the .HBNOISE statement. The data is plotted as a function of the input frequency band (IFB) points. Units are in V/Hz$^{1/2}$. Simulation ignores ONOISE when applied to autonomous circuits.
NF SSNF	NF and SSNF both output a single-side band noise figure as a function of the IFB points: NF = SSNF = 10 Log(SSF) Single side-band noise factor, SSF = {(Total Noise at output, at OFB, originating from all frequencies) - (Load Noise originating from OFB)} / (Input Source Noise originating from IFB).
DSNF	DSNF outputs a double side-band noise figure as a function of the IFB points. DSNF = 10 Log(DSF) Double side-band noise factor, DSF = {(Total Noise at output, at the OFB, originating from all frequencies) - (Load Noise originating from the OFB)} / (Input Source Noise originating from the IFB and from the image of IFB).
INOISE	Outputs input referred noise which can be printed, probed, or measured.

Output Data Files

An HBNOISE analysis produces these output data files:

- Output from the `.PRINT` statement is written to a .printpn# file.
- Output from the `.PROBE` statement is written to a .pn# file.

  Both the *.printpn# and *.pn# files output data against the input frequency band points.
- Standard output information is written to a .lis file:
  - simulation time
  - HBNOISE linear solver method
  - HBNOISE simulation time
  - total simulation time

See also Using Noise Analysis Results as Input Noise Source.

Measuring HBNOISE Analyses with .MEASURE

Note: A `.MEASURE HBNOISE` statement cannot contain an expression that uses a `HBNOISE` variable as an argument. Also, you cannot use a `.MEASURE HBNOISE` statement for error measurement and expression evaluation of `HBNOISE`.

The `.MEASURE HBNOISE` syntax supports several types of measurements:

- Find-when

  ```
  .MEASURE HBNOISE result FIND out_var1
  + AT = Input_Frequency_Band value
  ```

  The previous measurement yields the result of a variable value at a specific IFB point.

  ```
  .MEASURE HBNOISE result FIND out_var1
  + WHEN out_var2 = out_var3
  ```

  The previous measurement yields the result at the input frequency point when *out_var2* == *out_var3*.

  ```
  .MEASURE HBNOISE result WHEN out_var2 = out_var3
  ```

  The previous measurement yields the input frequency point when *out_var2* == *out_var3*.

- Average, RMS, min, max, and peak-to-peak

  ```
  .MEASURE HBNOISE result [RMS] out_var [FROM = IFB1]
  + [TO = IFB2]
  ```

- Integral evaluation

  ```
  .MEASURE HBNOISE result INTEGRAL out_var
  + [FROM = IFB1] [TO = IFB2]
  ```

  This measurement integrates the *out_var* value from the IFB1 frequency to the IFB2 frequency.

- Derivative evaluation

  ```
  .MEASURE HBNOISE result DERIVATIVE out_var AT = IFB1
  ```

This measurement finds the derivative of *out_var* at the IFB1 frequency point.

Note: `.MEASURE HBNOISE` cannot contain an expression that uses an *hbnoise* variable as an argument. You also cannot use `.MEASURE HBNOISE` for error measurement and expression evaluation of `HBNOISE`.

- Input referred noise

```
.MEASURE [HBNOISE|SNNOISE] result FIND inoise
+ AT = IFB_value
```

This measurement yields the result of the input referred noise at a specific input frequency band point.

```
.MEASURE [HBNOISE|SNNOISE] result FIND inoise
+ WHEN out_var2 = out_var3
```

This measurement yields the result at the input frequency point when out_var2 == out_var3.

```
.MEASURE HBNOISE result func inoise [FROM = IFB1]
+ [TO = IFB2]
```

Where `func` is one of the following measurement types:

- AVG (average): Calculates the area under the inoise curve, divided by the periods of interest.
- MAX (maximum): Reports the maximum value of inoise over the specified interval.
- MIN (minimum): Reports the minimum value of inoise over the specified interval.
- PP (peak-to-peak): Reports the maximum value, minus the minimum value of inoise over the specified interval.
- RMS (root mean squared): Calculates the square root of the area under the inoise curve, divided by the period of interest.

```
.MEASURE HBNOISE result INTEGRAL inoise
+ [FROM =IFB1] [TO = IFB2]
```

This measurement integrates the inoise value from the IFB1 frequency to the IFB2 frequency.

```
.MEASURE HBNOISE result DERIVATIVE inoise AT = IFB1
```

This measurement finds the derivative of inoise at the IFB1 frequency point.

The HSPICE advanced analog analyses optimization flow can read the measured data from a `.MEASURE HBNOISE` analysis. This flow can be combined in the HSPICE advanced analog analyses optimization routine with a `.MEASURE HBTR` analysis (see Using .MEASURE with .HB Analyses) and a `.MEASURE PHASENOISE` analysis (see Measuring Phase Noise with .MEASURE PHASENOISE).

Errors and Warnings

HBNOISE Errors

See the list of HBAC Errors and Warnings.

HBNOISE Example

This example performs an HB analysis, then runs an HBNOISE analysis over a range of frequencies, from 9.0e8 to 9.2e8 Hz. Simulation outputs the output noise at V(out) and the single side-band noise figure versus IFB, from 1e8 to 1.2e8 Hz, to the *.pn0 file. The netlist for this example is shown immediately following.

```
$$*-Ideal mixer + noise source
$ prints total noise at the output (1.57156p V/sqrt-Hz),
$ single-sideband noise figure, (3.01 dB)
$ double-sideband noise figure. (0 dB)
.OPTION PROBE
.OPTION POST=2
vlo lo 0 0.0 hb 1.0 0 1 1$ Periodic, HB Input
Ilo lo 0 0
rsrc rfin rf1 1.0$ Noise source
c1 0 if q='1.0e-9*v(lo)*v(rfin)' $ mixer element
g1 0 if cur='1.0*v(lo)*v(rfin)'  $ mixer element
rout if 0 1.0
vrf rf1 0 $ hbac 2.0 0.0
.hb tones=1.0g nharms=4 $ sweep mval 1 2 1
.HBNOISE rout rsrc lin 11 0.90g 0.92g
.print HBNOISE onoise ssnf dsnf
.probe HBNOISE onoise ssnf dsnf
.end
```

Shooting Newton Noise Analysis (.SNNOISE)

A SNNOISE (Shooting Newton noise) analysis simulates the noise behavior in periodic systems. It uses a Periodic AC (PAC) algorithm to perform noise analysis of non-autonomous (driven) circuits under periodic, steady-state tone conditions. SNNOISE is similar to the HBNOISE analysis.

The PAC method simulates noise assuming that the stationary noise sources and/or the transfer function from the noise source to a specific output are periodically modulated.

- The modulated noise source (thermal, shot, or flicker) is modeled as a cyclostationary noise source.
- A PAC algorithm solves the modulated transfer function.
- You can also use the SNNOISE PAC method with correlated noise sources, including the MOSFET Level 9 and Level 11 models, and the behavioral noise source in the G-element (Voltage Dependent Current Source).

You use the `.SNNOISE` statement to perform a Periodic Noise Analysis.

The following sections discuss these topics:

- Supported Features
- Input Syntax
- Output Syntax
- Output Data Files
- Measuring SNNOISE Analyses with .MEASURE
- SNNOISE Analysis Example

Supported Features

SNNOISE supports the following features:

- All existing HSPICE advanced analog noise models.
- Uses Shooting Newton to generate the steady-state solution.
- Unlimited number of sources.
- Includes stationary, cyclostationary, frequency-dependent, and correlated noise effects.

- Swept parameter analysis.
- Results are independent of the number of SNAC sources in the netlist.

Prerequisites and Limitations

The following prerequisites and limitations apply to SNNOISE:

- Requires one `.SN` statement (which determines the steady-state solution).
- Requires at least one Periodic source. Does not recognize HB sources.
- Requires placing the parameter `sweep` in the `.SN` statement.

Input Syntax

```
.SNNOISE [output] [insrc] [parameter_sweep]
+ [n1+/-1]
+ [listfreq=(frequencies|none|all)] [listcount=val]
+ [listfloor=val] [listsources=on|off]
```

Parameter	Description
output	Can be an output node, pair of nodes, or a 2-terminal element. HSPICE advanced analog analysis references the equivalent noise output to this node (or pair of nodes). Specify a pair of nodes as V(n+,n-). If you specify only one node V(n+), then HSPICE advanced analog analysis assumes that the second node is ground. You can also specify a 2-terminal element name that refers to an existing element in the netlist. If the 2-terminal element is a voltage source, then HSPICE advanced analog analysis outputs the noise current through the voltage source in A/sqrt(Hz). For all other 2-terminal devices, HSPICE outputs the noise voltage across the device in V/sqrt(Hz).
insrc	An input source. If this is a resistor, HSPICE advanced analog analyses uses it as a reference noise source to determine the noise figure. If the resistance value is 0, the result is an infinite noise figure.
parameter_sweep	Frequency sweep range for the input signal. Also referred to as the input frequency band (IFB) or *fin*). You can specify LIN, DEC, OCT, POI, SWEEPBLOCK, DATA, MONTE, or OPTIMIZE sweeps. Specify the nsteps, start, and stop frequencies using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ *SWEEPBLOCK=swblockname*

Parameter	Description
n1,+/-1	Index term defining the output frequency band (OFB or *fout*) at which the noise is evaluated. Generally, fout=ABS(n1*f1+/-fin) Where: f1 is the fundamental harmonic (tone) determined in the Shooting Newton analysis n1 is the associated harmonic multiplier n1,n2,...,nk are the associated harmonic multipliers; n1 can be any non-negative integer $\leq$ nharm defined in the .SN statement; +/-1 is fixed, either +1 or -1 fin is the IFB defined by *parameter_sweep*. The default index term is [1,-1]. For a single tone analysis, the default mode is consistent with simulating a low-side, down conversion mixer where the advanced analog signal is specified by the IFB and the noise is measured at a down-converted frequency that the OFB specifies. In general, you can use the [n1,+/-1] index term to specify an arbitrary offset. The noise figure measurement is also dependent on this index term. See Specifying Variant Indices and Measuring SNNOISE Analyses with .MEASURE.

listfreq	Prints the element noise value to the .lis file. You can specify at which frequencies the element noise value is printed. The frequencies must match the sweep_frequency values defined in the *parameter_sweep*, otherwise they are ignored.
	
	In the element noise output, the elements that contribute the largest noise are printed first. The frequency values can be specified with the NONE or ALL keyword, which either prints no frequencies or every frequency defined in *parameter_sweep*. Frequency values must be enclosed in parentheses. For example:`listfreq=(none)`
	`listfreq=(all)`
	`listfreq=(1.0G)`
	`listfreq=(1.0G, 2.0G)`
	The default value is NONE.

listcount	Prints the element noise value to the .lis file, which is sorted from the largest to smallest value. You do not need to print every noise element; instead, you can define `listcount` to print the number of element noise frequencies. For example, `listcount=5` means that only the top 5 noise contributors are printed. The default value is 1.
listfloor	Prints the element noise value to the `.lis` file and defines a minimum meaningful noise value (in V/Hz$^{1/2}$ units). Only those elements with noise values larger than `listfloor` are printed. The default value is 1.0e-14 V/Hz$^{1/2}$.
listsources	Prints the element noise value to the `.lis` file when the element has multiple noise sources, such as a FET, which contains the thermal, shot, and 1/f noise sources. You can specify either ON or OFF: ON Prints the contribution from each noise source and OFF does not. The default value is OFF.

Specifying Variant Indices

.SNNOISE is the appropriate HSPICE advanced analog analysis for the computation of noise at the output of a sample and hold circuit. When using .SNNOISE, you need to specify the indices as [0,1] instead of the default [1,-1]. When you specify the indices as [0,1], you get results that are “what you see is what you get” with respect to the frequency sweep specified in the .SNNOISE command. There are two more important things to consider when using .SNNOISE. You must use enough harmonics to resolve the clock edge and you will want a high density of SN time points. It is recommended that the number of time points be between 2 and 20 times the number of harmonics used. To increase the density of the time points during the .SN analysis, you can use the option DELMAX to specify a maximum time step.

For example, you can use the following settings:

```
.OPTION SNACCURACY=50
.OPTION DELMAX=5p

.SN TONES=5e6 nharms=2000 trinit=800n
.SNNOISE v(vo) vsin [0,1] dec 100 1e6 1e9
```

This example uses 100 points per decade for the frequency sweep instead of 1000 points per decade. This is to speed up the simulation. It does not affect the accuracy of the results.

Output Syntax

This section describes the syntax for the SNNOISE `.PRINT` and `.PROBE` statements.

.PRINT and .PROBE Statements

```
.PRINT SNNOISE [ONOISE] [NF] [SSNF] [DSNF] [INOISE]
.PROBE SNNOISE [ONOISE] [NF] [SSNF] [DSNF] [INOISE]
```

Parameter	Description
ONOISE	Outputs the voltage noise at the output frequency band (OFB) across the output nodes in the .SNNOISE statement. The data is plotted as a function of the input frequency band (IFB) points. Units are in $V/Hz^{1/2}$. Simulation ignores ONOISE when applied to autonomous circuits.

Parameter	Description
NF *SSNF*	NF and SSNF both output a single-side band noise figure as a function of the IFB points: `NF = SSNF = 10 Log(SSF)` Single side-band noise factor, SSF = {(Total Noise at output, at OFB, originating from all frequencies) - (Load Noise originating from OFB)} / (Input Source Noise originating from IFB).
DSNF	DSNF outputs a double side-band noise figure as a function of the IFB points. `DSNF = 10 Log(DSF)` Double side-band noise factor, DSF = {(Total Noise at output, at the OFB, originating from all frequencies) - (Load Noise originating from the OFB)} / (Input Source Noise originating from the IFB and from the image of IFB).
INOISE	Outputs input referred noise which can be printed, probed, or measured.

Output Data Files

An SNNOISE analysis produces these output data files:

- Output from the `.PRINT` statement is written to a .printsnpn# file.
- Output from the `.PROBE` statement is written to a `.snpn#` file.

  Both the `*.printsnpn#` and `*.pn#` files output data against the input frequency band points.
- Standard output information is written to a `.lis` file:
  - simulation time
  - SNNOISE linear solver method
  - SNNOISE simulation time
  - total simulation time

See also Using Noise Analysis Results as Input Noise Sources.

Measuring SNNOISE Analyses with .MEASURE

Note: A `.MEASURE SNNOISE` statement cannot contain an expression that uses a `SNNOISE` variable as an argument. Also, you cannot use a `.MEASURE SNNOISE` statement for error measurement and expression evaluation of `SNNOISE`.

The `.MEASURE SNNOISE` syntax supports four types of measurements:

- Find-when

  ```
  .MEASURE SNNOISE result FIND out_var1
  + At = Input_Frequency_Band value
  ```

  The previous measurement yields the result of a variable value at a specific IFB point.

  ```
  .MEASURE SNNOISE result FIND out_var1
  + WHEN out_var2 = out_var3
  ```

  The previous measurement yields the result at the input frequency point when *out_var2* == *out_var3*.

  ```
  .MEASURE SNNOISE result WHEN out_var2 = out_var3
  ```

  The previous measurement yields the input frequency point when *out_var2* == *out_var3*.

- Average, RMS, min, max, and peak-to-peak

  ```
  .MEASURE SNNOISE result [RMS] out_var [FROM = IFB1]
  + [TO = IFB2]
  ```

- Integral evaluation

  ```
  .MEASURE SNNOISE result INTEGRAL out_var
  + [FROM = IFB1] [TO = IFB2]
  ```

  This measurement integrates the *out_var* value from the IFB1 frequency to the IFB2 frequency.

- Derivative evaluation

  ```
  .MEASURE SNNOISE result DERIVATIVE out_var AT = IFB1
  ```

  This measurement finds the derivative of *out_var* at the IFB1 frequency point.

  Note: `.MEASURE SNNOISE` cannot contain an expression that uses an *hbnoise* variable as an argument. You also cannot use `.MEASURE SNNOISE` for error measurement and expression evaluation of `SNNOISE`.

- Input referred noise:

```
.MEASURE SNNOISE result FIND inoise
+ AT = IFB_value
```

This measurement yields the result of the input referred noise at a specific input frequency band point.

```
.MEASURE SNNOISE result FIND inoise
+ WHEN out_var2 = out_var3
```

This measurement yields the result at the input frequency point when out_var2 == out_var3.

```
.MEASURE SNNOISE result func inoise [FROM = IFB1]
+ [TO = IFB2]
```

Where `func` is one of the following measurement types:

- AVG (average): Calculates the area under the inoise curve, divided by the periods of interest.
- MAX (maximum): Reports the maximum value of inoise over the specified interval.
- MIN (minimum): Reports the minimum value of inoise over the specified interval.
- PP (peak-to-peak): Reports the maximum value, minus the minimum value of inoise over the specified interval.
- RMS (root mean squared): Calculates the square root of the area under the inoise curve, divided by the period of interest.

```
.MEASURE SNNOISE result INTEGRAL inoise
+ [FROM =IFB1] [TO = IFB2]
```

This measurement integrates the inoise value from the IFB1 frequency to the IFB2 frequency.

```
.MEASURE SNNOISE result DERIVATIVE inoise AT = IFB1
```

This measurement finds the derivative of inoise at the IFB1 frequency point.

SNNOISE Analysis Example

This example performs an SN analysis, then runs an SNNOISE analysis over a range of frequencies, from 9.0e8 to 9.2e8 Hz. Simulation outputs the output noise at V(out) and the single side-band noise figure versus IFB, from 9.0e8 to

9.2e8 Hz, to the *.pn0 file. The netlist for this example is shown immediately following.

```
*
$$*-Ideal mixer + noise source
$ prints total noise PSD at the output (2.47e-20 V^2) when q=0
$ single-sideband noise figure, (3.01 dB)
$ double-sideband noise figure. (0 dB)
.OPTION PROBE
.OPTION POST=2
vlo lo 0 0.0 cos (0 1.0 1.0g 0 0 0)
Ilo lo 0 0
rsrc rfin rf1 1.0$ Noise source
g1 0 if cur='1.0*v(lo)*v(rfin)' $ mixer element
c1 0 if q='1.0e-9*v(lo)*v(rfin)' $ mixer element
rout if 0 1.0
vrf rf1 0 $ hbac 2.0 0.0
.option delmax=0.002n
.SN tones=1G   nharms=4 trstab=10n
.SNNOISE rout rsrc lin 11 0.90g 0.92g
.probe SNNOISE onoise ssnf dsnf
.print SNNOISE onoise ssnf dsnf
.end
```

Periodic Time-Dependent Noise Analysis (.PTDNOISE)

While HBNOISE and SNNOISE calculate a time-averaged power spectral density, there are applications where a characterization of the time-dependence of the noise is required. These applications include computation of jitter associated with a noisy signal crossing a threshold and computation of the noise associated with discretization of an analog signal, which computes the noise in a periodically driven circuit at a point in time. Periodic Time-Dependent noise analysis (PTDNOISE) calculates the noise spectrum and the total noise at a point in time. Jitter in a digital threshold circuit can then be determined from the total noise and the digital signal slew rate.

Circuits driven by large periodic signals produce cyclostationary noise, that is, the noise characteristics are periodic in time. Cyclostationary noise can be characterized in several ways, with the particular application determining which is appropriate.[9] The time-average power spectral density (PSD) ignores frequency correlations in the noise, but is adequate when the fundamental frequency of the cyclostationary noise is much larger than the bandwidth of interest. The time-average PSD is calculated in the HBNOISE/SNNOISE analyses. [10]

The harmonic power spectral density (HPSD) or equivalently, the auto-correlation function, R(t1,t2), contains the correlation information between noise sidebands that is necessary to build behavioral cyclostationary noise sources and to separate the amplitude modulation (AM) and phase modulation (PM) noise components. (See Amplitude Modulation/Phase Modulation Separation for more information.

The time-dependent power spectral density (TDPSD) can be integrated over frequency to yield the time-dependent noise (TDN). TDN can then be used to determine jitter associated with a noisy signal crossing a threshold. PTDNOISE analysis allows the calculation of TDPSD, TDN, and jitter. In addition, you can calculate both the time-domain power spectral density (TDSN) and the integrated noise (time-dependent noise, TDN) at multiple time points.

By measuring the jitter associated with a noisy signal crossing a threshold, jitter is modeled by displacing the time in a noise free signal v(t) with a stochastic process j(t).

Equation 35

$$V_{jitter}(t) = v(t + j(t))$$

We can also determine the voltage at this node including the time-dependent noise n(t):

Equation 36

$$Vn(t) = v(t) + n(t)$$

by equating these two representations, expanding in a Taylor series, and dropping higher order terms, as follows:

Equation 37

$$V(t) + n(t) = v(t + j(t)) = v(t) + dv(t)/dt \cdot j(t) + \ldots$$

Equation 38

$$N(t) = dv(t)/dt \cdot j(t)$$

In terms of variances, jitter is then defined as:

Equation 39

$$Var(j(t)) = n^2(t)/(dv(t)/dt)^2$$

The following sections discuss these topics:

- PTDNOISE Input Syntax
- PTDNOISE Output Syntax and File Format
- Error Handling and Warnings
- Usage Example

PTDNOISE Input Syntax

```
.PTDNOISE output TIME=[val|meas|sweep]
+ [TDELTA=time_delta]
+ frequency_sweep
+ [listfreq=(frequencies|none|all)]
+ [listcount=val] [listfloor=val]
+ [listsources=on|off]
```

Parameter	Description
output	Is an output node, pair of nodes, or 2-terminal elements. HSPICE advanced analog analyses references the equivalent noise output to this node (or pair of nodes). Specify a pair of nodes as V(n+,n-); only one node as V(n+, n-). If you specify only one node, V(n+), then HSPICE advanced analog analyses assumes the second node is ground. You can also specify a 2-terminal element name that refers to an existing element in the netlist.
TIME	Time point at which time domain noise is evaluated. Specify either a time point explicitly, such as: TIME=value, where value is either numerical or a parameter name or a .MEASURE name associated with a time domain .MEASURE command located in the netlist. PTDNOISE uses the time point generated from the .MEASURE command to evaluate the noise characteristics. This is useful if you want to evaluate noise or jitter when a signal reaches some threshold value.
TDELTA	A time value used to determine the slew rate of the time-domain output signal. Specified as TDELTA=value. The signal slew rate is then determined by the output signal at TIME +/- TDELTA and dividing this difference by 2 x TDELTA. This slew rate is then used in the calculation of the strobed jitter. If this term is omitted a default value of 0.01 x the .SN period is assumed.
frequency_sweep	Frequency sweep range for the output noise spectrum. The upper and lower limits also specify the integral range in calculating the integrated noise value. Specify LIN,DEC, OCT, POI, SWEEPBLOCK, DATA sweeps. Specify the nsteps, start, and stop frequencies using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC*nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ *SWEEPBLOCK=swblockname* ■ DATA dataname

Parameter	Description
listfreq	Prints the element noise value to the `.lis` file. This information is only printed if a noise spectrum is requested in a PRINT or PROBE statement. (See PTDNOISE Output Syntax and File Format.) You can specify which frequencies the element noise is printed. The frequencies must match the sweep_frequency values defined in the *frequency_sweep*, otherwise they are ignored. In the element noise output, the elements that contribute the largest noise are printed first. The frequency values can be specified with the NONE or ALL keyword, which either prints no frequencies or every frequency defined in *frequency_sweep*. Frequency values must be enclosed in parentheses. For example: ■ listfreq=(none) ■ listfreq=(all) ■ listfreq=(1.0) ■ listfreq=(1.0G, 2.0G) The default value is NONE.
listcount	Prints the element noise value to the `.lis` file, which is sorted from the largest to smallest value. You do not need to print every noise element; instead, you can define `listcount` to print the number of element noise frequencies. For example, `listcount=5` means that only the top 5 noise contributors are printed. The default value is 1.
listfloor	Prints the element noise value to the `.lis` file and defines a minimum meaningful noise value (in V/Hz$^{1/2}$ units). Only those elements with noise values larger than `listfloor` are printed. The default value is 1.0e-14 V/Hz$^{1/2}$.
listsources	Prints the element noise value to the `.lis` file when the element has multiple noise sources, such as a MOSFET, which contains the thermal, shot, and 1/f noise sources. You can specify either ON or OFF: ON prints the contribution from each noise source and OFF does not. The default value is OFF.

PTDNOISE Output Syntax and File Format

PTDNOISE output syntax allows for the output of a single parameter: onoise, where, onoise is the noise voltage spectral density at each frequency point specified by the frequency_sweep keyword for the time points specified by the TIME keyword. The units are $\frac{V}{\sqrt{Hz}}$.

```
.PROBE PTDNOISE onoise
```

```
.PRINT PTDNOISE onoise
```

Parameter	Units	Description
onoise	$\frac{V}{\sqrt{Hz}}$	Noise voltage spectral density at each frequency point specified by *frequency_sweep* at the time point specified by *time_value*

Output File Format

The following PTDNOISE output files are generated depending on the user input:

File	Description
*.printptn#	Writes output from the .PRINT statement when using HB to obtain the steady state solution
*.ptn#	Writes output from the .PROBE statement when using HB to obtain the steady state solution
*.printsnptn#	Reports output from the .PRINT statement when using SN to obtain the steady state solution.
*.snptn#	Writes output from the .PROBE statement when using SN to obtain the steady state solution.
*.lis	Standard output file *.lis contains the following information: ■ Performance Statistics Log ■ Number of Nodes ■ Number of FFT Points ■ Number of Equations ■ Memory in use ■ Maximum Krylov iterations ■ Maximum Krylov Dimension ■ Target GMRES Residual ■ Gmres Residual ■ Actual Krylov Iterations taken ■ Frequency (swept input frequency values) Noise source contributions are listed sequentially and are controlled by the PTDNOISE command line parameters: listtime, listfreq, listcount, listfloor, and listsources.

.MEASURE Syntax and File Format

The syntax for .MEASURE PTDNOISE is:

```
.MEASURE PTDNOISE result [integnoise|jitter|slewrate]
```

.MEASURE PTDNOISE allows for the measurement of these parameters: integnoise, time-point, tdelta-value, slewrate, and strobed jitter.

Parameter	Units	Description
integnoise	V	Voltage noise integrated over a frequency range specified by frequency_range at the time point specified by TIME=*val*.
slewrate	v/sec	Output signal slewrate at the time point specified by TIME=*val*.
jitter	sec	Calculated from the noise voltage (integrated over the frequency range specified by frequency_range), divided by the slew rate at the same node(s), at the time point specified by TIME=*val*.

Note: .MEASURE PTDNOISE is ignored when a TIME=*sweep* is specified in the netlist and a warning message is issued.

Measure File Format

File	Description
*.msnptn#	Contains output from the .MEASURE statement when using .SN to obtain the steady state solution.

Error Handling and Warnings

Error messages are generated under the following circumstances:

- PTDNOISE frequency sweep includes negative frequencies. PTDNOISE allows only frequencies that are greater than or equal to zero.
- PTDNOISE time sweep includes negative times. PTDNOISE allows only time points that are greater than or equal to zero.
- No SN statement is specified (error at parser). PTDNOISE requires an SN statement to generate the steady-state solution.
- Incorrect match to .MEASURE statement.

A warning is issued for a PTDNOISE convergence failure. When the gmres solver reaches the maximum number of iterations and the residual is greater than the specified tolerance, PTDNOISE generates a warning and then continue as if the data were valid. The Warning reports the following information:

- Final GMRES Residual
- Target GMRES Residual
- Maximum Krylov Iterations
- Actual Krylov Iterations taken

Usage Example

The following test case illustrates the PTDNOISE analysis for a simple inverter.

```
* Simple RC + Inverter - rcInvPTDNoise.sp
* rrd Jan 03, 2007
* Simulates PSD(t,f) of a simple inverter
* sweep time points
.param f0 = 5.0e8
.sn tones=f0 nharms=4 trinit=10n
.PTDNOISE v(out1) TIME=lin 3 0 2n TDELTA=.1n dec 5 1e5 1e10
+ listfreq=(1e6,1e8)
+ listcount=1
+ listsources=ON

.MEASURE PTDNOISE strobejit STROBEJITTER onoise FROM = 1e4 TO =
1e10
$.measure SN t1 trig AT=0 targ v(out1) val=1.5 fall=1

.opt post
.probe ptdnoise onoise
.print ptdnoise onoise
.probe sn v(out1)

vd    vdd  0   3.0
.global vdd
vgate in0  0   COS(1.5 1.4 'f0'  0 0 0)
rin   in0 in1 50
rout  out1  0    .1g

xo1 in1 out1 inv

.subckt inv in out
m1 out in 0 0 n l=350e-9 w=4.5e-6
m2 out in vdd vdd p l=350e-9 w=4.5e-6
.ends
```

```
.MODEL N NMOS
+Level= 49 Tnom=27.0 version =3.1 TLEVC= 1
*
***

.MODEL P PMOS
+Level= 49 Tnom=27.0 version =3.1 TLEVC= 1

.end
```

Troubleshooting .PTDNOISE Simulation

When performing jitter measurements on a clock buffer using PTDNOISE analysis, you may get unusual results. If you define noise frequencies that are multiples of your input clock, you get large values in your jitter measurement. When you sweep through frequencies that are multiples of the clock, PTD noise up-converts `frequency=0` flicker noise. (These contributions can be large, since flicker noise typically has a $freq^{-n}$ dependence where $n \cong 1$.) For this reason you should avoid sweeping through multiples of the clock frequency.

Note also, the flicker noise is limited at very small frequencies so you do not see infinite noise.

Multitone Harmonic Balance Transfer Function Analysis (.HBXF)

The `.HBXF` command calculates the transfer function from a given source in the circuit to a designated output. Frequency conversion is calculated from the input frequencies to a single output frequency that is specified with the command. The relationship between the `.HBXF` command and the input/output is expressed in the following equation:

Equation 40

$$Y_m(j\omega_0) = \sum_{\omega \varepsilon W} HBXF_{m,n}(j\omega_0, j(\omega + \Delta\omega)) \cdot X_n(j(\omega + \Delta\omega))$$

where:

- $HBXF_{m,n}(j\omega_0, j(\omega + \Delta\omega))$ is the transfer function from input port n to the output port m
- W is the set of all possible harmonics
- $\omega + \Delta\omega$ is the input frequency
- $\Delta\omega$ is the offset frequency
- m is the output node number
- n is the input node number
- ω_0 is the output frequency
- Y is the output (voltage or current)
- X is the input (voltage or current)

The following sections discuss these topics:

- Supported Features
- Input Syntax
- Output Syntax
- Output Data Files
- Using the .MEASURE Command with .HBXF
- Example
- HBXF Test Listing

Supported Features

The `.HBXF` command supports the following features:

- All existing HSPICE advanced analog models and elements
- Sweep parameter analysis
- Unlimited number of HB sources

Prerequisites and Limitations

The following prerequisites and limitations apply to the `.HBXF` command:

- Only one `.HBXF` statement is required. If you use multiple `.HBXF` statements, HSPICE advanced analog analyses uses only the last `.HBXF` statement.
- At least one `.HB` statement is required, which determines the steady-state solution.
- Parameter sweeps must be placed in `.HB` statements.

Input Syntax

```
.HBXF out_var freq_sweep
```

Parameter	Description
out_var	Specify `i(2`*_port_elem*`)` or `V(`*n1*`[,`*n2*`])`
freq_sweep	Frequency sweep range for the input signal (also referred to as the input frequency band (IFB or fin)). A sweep of type LIN, DEC, OCT, POI, or SWEEPBLOCK. Specify the nsteps, start, and stop frequencies using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ *SWEEPBLOCK=swblockname* **Note:** Specify the frequency sweep range for the output signal. HSPICE advanced analog analyses determines the offset frequency in the input sidebands; for example, f1 = abs(fout - k*f0) s.t. f1<=f0/2 The f0 is the steady-state fundamental tone, and f1 is the input frequency.

Output Syntax

This section describes the syntax for the HBXF `.PRINT` and `.PROBE` statements.

.PRINT and .PROBE Statements

```
.PRINT HBXF TYPE (NODES | ELEM)
```

```
.PROBE HBXF TYPE (NODES | ELEM)
```

Parameter	Description
TYPE	TYPE can be one of the following: ▪ TFV = existing source ▪ TFI = placeholder value for the current source attached to the given node. The transfer function is computed on the output variables and input current or voltage.
NODES | ELEM	NODES or ELEM can be one of the following: ▪ Voltage type – a single node name (n1), or a pair of node names, (n1,n2) ▪ Current type – an element name (elemname) ▪ Power type – a resistor (resistorname) or port (portname) element name.

Output Data Files

An HBXF calculation produces these output data files:

- Output from the `.PRINT` statement is written to a `.printxf#` file.
  - The output is in ohms, siemens, or undesignated units, and the header in the output file is Z(..). Y(..) or GAIN(..).
- Output from the `.PROBE` statement is written to an

  `.xf#` file.
- Reported performance log statistics are written to a .lis file:
  - HBXF CPU time
  - HBXF peak memory usage

Using the .MEASURE Command with .HBXF

Since `.HBXF` requires an `.HB` analysis, the measure statements for this analysis are the same as for `.HB` analysis. For example,

```
.MEASURE HB result FIND out_var AT=val
```

Example

Based on the HB analysis, the following example computes the trans-impedance from `isrc` to `v(1)`.

```
.hb tones=1e9 nharms=4
.hbxf  v(1) lin 10 1e8 1.2e8
.print hbxf tfv(isrc)  tfi(n3)
```

HBXF Test Listing

```
* Test HBXF: nonlinear order-2 poly equation
.OPTIONS  PROBE
.OPTIONS POST=2
vlo lo 0 cos(0 1.0 1g 0 0) tranforhb=1
rlo lo 0 50
vrf1 rf1 0 0
rrf1 rf1 0 50
E1 out 0 POLY(2) lo 0 rf1 0  0 1 1 1 10 1
rout out 0 50
.hb tones=1g nharms=5
.hbxf v(out) lin 2 100meg 200meg
.print hb v(out) v(rf1) v(lo)
.print hbxf tfv(vrf1) tfv(vlo)
.end
```

Shooting Newton Transfer Function Analysis (.SNXF)

The `.SNXF` command calculates transfer functions from an arbitrary number of small signal sources to a designated output in a circuit under periodic steady state conditions. Frequency conversion is calculated from multiple input frequencies to a single output at a single frequency that is specified on the command line.

Prerequisites and Limitations

The following prerequisites and limitations apply to the .SNXF command:

- Only one .SNXF statement is required. If you use multiple .SNXF statements, HSPICE advanced analog analyses uses only the last one.
- At least one .SN statement is required, which determines the steady-state solution.
- Parameter sweeps must be placed in .SN statements.

The following sections discuss these topics:

- Input Syntax
- Output Syntax

- Output Data Files
- Using the .MEASURE Command with .SNXF
- Example
- SNXF Test Listing

Input Syntax

```
.SNXF out_var freq_sweep
```

Parameter Description

Parameter	Description
out_var	Specify `i(2_port_elem)` or `V(n1[,n2])`
freq_sweep	Frequency sweep range for the input signal (also referred to as the input frequency band (IFB or fin)). A sweep of type LIN, DEC, OCT, POI, or SWEEPBLOCK. Specify the nsteps, start, and stop frequencies using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ *SWEEPBLOCK=swblockname* Specify the frequency sweep range for the output signal. HSPICE advanced analog analyses determines the offset frequency in the input sidebands Fin, where Fin = abs(n*F0 +/- Fout). F0 is the steady-state fundamental tone, and Fout is the output frequency. SNXF then generates the transfer functions from all of the input sidebands (the Fin values) to the output frequency Fout.

Output Syntax

This section describes the syntax for the SNXF `.PRINT` and `.PROBE` statements.

.PRINT and .PROBE Statements

```
.PRINT SNXF TYPE(NODES | ELEM)
.PROBE SNXF TYPE(NODES | ELEM)
```

Parameter Description

TYPE can be one of the following:

- TFV = existing source
- TFI = placeholder value for the current source attached to the given node.

The transfer function is computed on the output variables and input current or voltage `.NODES | ELEM NODES` or `ELEM` can be one of the following:

- Voltage type – a single node name (n1), or a pair of node names, (n1,n2)
- Current type – an element name (elemname)
- Power type – a resistor (resistorname) or port (portname) element name

Output Data Files

An SNXF calculation produces these output data files:

- Output from the `.PRINT` statement is written to a `.printsnxf#` file. The output is in ohms, siemens, or undesignated units, and the header in the output file is Z(..). Y(..) or GAIN(..).
- Output from the .PROBE statement is written to an `.snxf#` file.

Reported performance log statistics are written to a `.lis` file:

- SNXF CPU time
- SNXF peak memory usage

Using the .MEASURE Command with .SNXF

Since `.SNXF` requires an `.SN` analysis, the measure statements for this analysis are the same as for `.SN` analysis. For example,

```
.MEASURE SN result FIND out_var AT=val
```

Example

Based on the SN analysis, the following example computes the transimpedance from isrc to v(1).

```
.SN tones=1e9 nharms=4
.SNXF v(1) lin 10 1e8 1.2e8
print SNXF TFV(isrc) TFI(n3)
```

SNXF Test Listing

```
* Test SNXF: nonlinear order-2 poly equation
.OPTIONS  PROBE
.OPTIONS POST=2
vlo lo 0 cos(0 1.0 1g 0 0)
rlo lo 0 50
vrf1 rf1 0 0
rrf1 rf1 0 50
E1 out 0 POLY(2) lo 0 rf1 0  0 1 1 1 10 1
rout out 0 50
.opt delmax=.01n
.sn tones=1g nharms=5
.snxf v(out) lin 2 100meg 200meg
.print sn v(out) v(rf1) v(lo)
.print snxf tfv(vrf1) tfv(vlo)
.end
```

References

[1] S. Maas, *Nonlinear Microwave Circuits*, Chapter 3, IEEE Press, 1997.

[2] R. Gilmore and M.B. Steer, “Nonlinear Circuit Analysis Using the Method of Harmonic Balance - A Review of the Art, Part I, Introductory Concepts.” *International Journal of Microwave and Millimeter-wave Computer-Aided Engineering*, Volume 1, No. 1, pages 22-37, 1991.

[3] R. Gilmore and M.B. Steer, “Nonlinear Circuit Analysis Using the Method of Harmonic Balance - A Review of the Art. Part II. Advanced Concepts.” *International Journal of Microwave and Millimeter-wave Computer-Aided Engineering*, Volume 1, No. 2, pages 159-180, 1991.

[4] V. Rizzoli, F. Mastri, F. Sgallari, G. Spaletta, “Harmonic-Balance Simulation of Strongly Nonlinear Very Large-Size Microwave Circuits by Inexact Newton Methods,” *MTT-S Digest*, pages 1357-1360, 1996.

[5] S. Skaggs, *Efficient Harmonic Balance Modeling of Large Microwave Circuits*, Ph.D. thesis, North Carolina State University, 1999.

[6] R.S. Carson, *High-Frequency Amplifiers*, 2nd Edition, John Wiley & Sons, 1982

[7] S.Y. Liao, *Microwave Circuit Analysis and Amplifier Design*, Prentice-Hall, 1987.

[8] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, 1995.

[9] J. Roychowdhury, D. Long, and P. Feldmann, “Cyclostationary Noise Analysis of Large RF Circuits with Multitone Excitations,” *IEEE Journal of Solid-State Circuits*, volume 33, pages 324–336, March 1998.

[10] A. Demir, A. Sangiovanni-Vincentelli, “Analysis and Simulation of Noise in Nonlinear Electronic Circuits and Systems”, Kluwer Academic, 1998.

6

S-parameter Analyses

Describes how to do frequency translation and large-signal S-parameter extraction, as well as noise parameter calculation.

These topics are covered in the following sections:

- Frequency Translation S-Parameter (HBLIN) Extraction
- Large-Signal S-parameter (HBLSP) Analysis

This chapter discusses various techniques supported in HSPICE advanced analog analyses for extracting circuit scattering parameters. Since advanced analog circuits can operate under large-signal and small-signal conditions, there are several types of scattering parameters that are useful to measure.

Linear small-signal scattering parameters represent the advanced analog frequency-domain transfer characteristics for a circuit that is operating at its DC bias condition, but the stimulus and response signals are sufficiently small that they do not influence the operating point. This type of analysis is performed using the `.LIN` analysis. For information on doing small-signal S-parameter analysis (.LIN), see Linear Network Parameter Analysis in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

In the case of advanced analog mixers and receiver front-ends, some of the input and output frequencies of interest involve a frequency translation. This translation is intentional and caused by nonlinear mixing in the circuit due to devices being driven by large-signal periodic waveforms. This type of scattering parameter analysis therefore must begin by solving the large-signal periodic response, and then finding the small-signal behavior about this large-signal operating point. This capability is provided by the .HBLIN analysis, which has setup and analysis control options similar to .LIN, but is capable of extracting S-parameters about a large-signal periodic steady-state operating point.

In the case of circuits such as power amplifiers, the extraction of scattering parameters is also important, but the circuit stimulus and response signals may themselves be large-signal periodic waveforms. And, it can be important to analyze how these S-parameter vary as a function of input power levels. This capability is provided by the .HBLSP Large-Signal S-parameter analysis, which uses large-signal stimulus signals for the S-parameter extractions.

Frequency Translation S-Parameter (HBLIN) Extraction

Frequency translation scattering parameter (S-parameter) extraction is used to describe N-port circuits that exhibit frequency translation effects, such as mixers. The analysis is similar to the existing LIN analysis, except that the circuit is first linearized about a periodically varying operating point instead of a simple DC operating point. After the linearization, the S-parameters between circuit ports that convert signals from one frequency band to another are calculated.

You use the `.HBLIN` statement to extract frequency translation S-parameters and noise figures.

Frequency translation S-parameter describes the capability of a periodically linear time varying systems to shift signals in frequency. The S-parameters for a frequency translation system are similar to the S-parameters of a linear-time-varying system, it is defined as:

Equation 41

$$b = S \cdot a$$

$$S_{\mathrm{i,j;m,n}}(\omega) = \frac{b_{\mathrm{i,m}}(\omega)}{a_{\mathrm{j,n}}(\omega)}$$

$$a_{\mathrm{k} \neq j,\, p \neq n}(\omega) = 0$$

The incident waves, $a_{i,n}(\omega)$, and reflected waves, $b_{i,n}(\omega)$, are defined by using these equations:

Equation 42

$$a_{i,n}(\omega) = \frac{V_i(\omega + n\omega_0) + Z_{0i}I_i(\omega + n\omega_0)}{2\sqrt{Z_{0i}}}$$

$$b_{i,n}(\omega) = \frac{V_i(\omega + n\omega_0) - Z_{0i}I_i(\omega + n\omega_0)}{2\sqrt{Z_{0i}}}$$

where:

- ω_0 is the fundamental frequency (tone).
- n is a signed integer.
- i is the port number.
- $a_{i,n}(\omega)$ is the input wave at the frequency $\omega + n\omega_0$ on the i^th^ port.
- $b_{i,n}(\omega)$ is the reflected wave at the frequency $\omega + n\omega_0$ on the i^th^ port.
- $V_i(\omega + n\omega_0)$ is the Fourier coefficient at the frequency $\omega + n\omega_0$ of the voltage at port i.
- $I_i(\omega + \omega n_0)$ is the Fourier coefficient at the frequency $\omega + n\omega_0$ of the current at port i.
- Z_{0i} is the reference impedance at port i.
- V and I definitions are Fourier coefficients rather than phasors.

For a multi-tone analysis, it can be expressed as:

Equation 43

$$b = S \cdot a$$

$$S_{i,j;m_1 \ldots m_N,n_1,n_2 \ldots n_N}(\omega) = \frac{b_{i,m_1,m_2 \ldots m_N}(\omega)}{a_{j,n_1,n_2 \ldots n_N}(\omega)}$$

$$a_{k,p_1,p_2 \ldots p_N | k \neq j, \nabla p_q \neq n_q}(\omega) = 0$$

Equation 44

$$a_{i,n_1,n_2\ldots n_N}(\omega) = \frac{V_i\left(\omega + \sum_{j=1}^{N} n_j\omega_j\right) + Z_{0i}I_i\left(\omega + \sum_{j=1}^{N} n_j\omega_j\right)}{2\sqrt{Z_{oi}}}$$

$$b_{i,n_1,n_2\ldots n_N}(\omega) = \frac{V_i\left(\omega + \sum_{j=1}^{N} n_j\omega_j\right) - Z_{0i}I_i\left(\omega + \sum_{j=1}^{N} n_j\omega_j\right)}{2\sqrt{Z_{oi}}}$$

where:

- ω_j is the ith tone.

The frequency translate S-parameters are calculated by applying different $n_j(j = 1 \sim N)$ to different ports.

Limitations

The HBLIN analysis has these known limitations:

- Noise parameters are not calculated for mixed-mode operation.
- Only the S-parameters corresponding to the set of frequencies specified at each port are extracted.
- Multiple small-signal tones are not supported.
- The port (P) element impedance cannot be specified as complex.

HB Analysis

An HB analysis is required prior to an HBLIN analysis. To extract the frequency translation S-parameters, a sweep of the small-signal tone is necessary. You can identify the small-signal tone sweep in the `.HBLIN` command or in the `.HB` command together with a `SS_TONE` specification.

For additional information regarding HB analysis, see Harmonic Balance Analysis on page 27.

Port Element

You must use a port (P) element as the termination at each port of the system. To indicate the frequency band that the S-parameters are extracted from, it is necessary to specify a harmonic index for each P-element.

Port Element Syntax

Without `SS_TONE`

```
Pxxx p n n_ref PORT=portnumber
+ [HBLIN = [H1, H2, ... HN, +/-1]] ...
```

With `SS_TONE`

```
Pxxx p n n_ref [PORT=portnumber]
+ [HBLIN = [H1, H2, ... +/-1 ... HN]] ...
```

Parameter	Description
n_ref	Reference node used when a mixed-mode port is specified.
PORT	The port number. Numbered sequentially beginning with 1 with no shared port numbers.
HBLIN	Integer vector that specifies the harmonic index corresponding to the tones defined in the .HB command. The +/-1 term corresponds to the small-signal tone specified by SS_TONE in the .HB command. If there is no SS_TONE in the .HB command, the +/-1 term must be at the last entry of HBLIN vector.

HBLIN Analysis

You use the `.HBLIN` statement to extract frequency translation S-parameters and noise figures.

Input Syntax

Without `SS_TONE`

```
.HBLIN frequency_sweep
+ [NOISECALC = [1|0|yes|no]] [FILENAME=file_name]
+ [DATAFORMAT = [ri|ma|db]]
+ [MIXEDMODE2PORT = [dd|cc|cd|dc|sd|sc|cs|ds]]
```

With `SS_TONE`

```
.HBLIN [NOISECALC = [1|0|yes|no]] [FILENAME=file_name]
```

```
+ [DATAFORMAT = [ri|ma|db]]
+ [MIXEDMODE2PORT = [dd|cc|cd|dc|sd|sc|cs|ds]]
```

Parameter	Description
frequency_sweep	Frequency sweep range for the input signal (also referred to as the input frequency band (IFB) or fin). You can specify LIN, DEC, OCT, POI, or SWEEPBLOCK. Specify the nsteps, start, and stop frequencies using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ SWEEPBLOCK=swblockname ■ DATA=*dataname*
NOISECALC	Enables calculating the noise figure. The default is no (0).
FILENAME	Specifies the output file name for the extracted S-parameters or the object name after the -o command-line option. The default is the netlist file name.
DATAFORMAT	Specifies the format of the output data file. ■ dataformat=RI, real-imaginary. ■ dataformat=MA, magnitude-phase. This is the default format for Touchstone files. ■ dataformat=DB, DB(magnitude)-phase.
MIXEDMODE2PORT	Describes the mixed-mode data map of output mixed mode S-parameter matrix. The availability and default value for this keyword depends on the first two port (P element) configuration as follows: ■ case 1: p1=p2=single-ended (standard-mode P element) available: ss default: ss ■ case 2: p1=p2=balanced (mixed-mode P element) available: dd, cd, dc, cc default: dd ■ case 3: p1=balanced p2=single-ended available: ds, cs default: ds ■ case 4: p1=single p2=balanced available: sd, sc default: sd

Example 1

Single-tone analysis with frequency translation. In this example, the 2-port S-parameters from RF (1G-del_f) to IF (del_f) are extracted. The LO signal is specified by normal voltage source Vlo. The frequency on port 1 is in the RF band, 1G-del_f, and the frequency on port 2 is in the IF band, del_f. The IF band is swept from 0- to 100-MHz. The results are output to file ex1.s2p.

```
p1 RFin gnd port=1 HBLIN=(1,-1)
p2 IFout gnd port=2 HBLIN=(0,1)
Vlo LOin gnd DC 0 HB 2.5 0 1 1
.HB tones=1G harms=5
.HBLIN lin 5 0 100meg noisecalc=no filename=ex1
+ dataformat=ma
```

Example 2

Another single-tone analysis with frequency translation example. In this example, the 3-port S-parameters are extracted. Port 3 provides the periodic large signal. The frequency on port 1 is del_f, the frequency on port 2 is 1G*2-del_f, and the frequency on port 3 is 1G*1+del_f. The small-signal frequency is swept from 0 to 100MHz. HBNOISE calculation is required. The results are output to file ex2.s3p.

```
p1 1 0 port=1 HBLIN=(0, 1)
p2 2 0 port=2 HBLIN=(2, -1)
p3 3 0 port=3 hb 0.5 0 1 1 HBLIN=(1, 1)
.HB tones=1G harms=5
.HBLIN lin 5 0 100meg noisecalc=yes filename=ex2
```

Output Syntax

This section describes the syntax for the `HBLIN .PRINT` and `.PROBE` statements.

.PRINT and .PROBE Statements

```
.PRINT HBLIN Smn | Smn(TYPE) | S(m,n) | S(m,n)(TYPE)
.PROBE HBLIN Smn | Smn(TYPE) | S(m, n) | S(m, n)(TYPE)
.PRINT HBLIN SXYmn | SXYmn(TYPE) | SXY(m,n) | SXY(m,n)(TYPE)
.PROBE HBLIN SXYmn | SXYmn(TYPE) | SXY(m, n) | SXY(m, n)(TYPE)
.PRINT HBLIN NF SSNF DSNF
.PROBE HBLIN NF SSNF SSNF
```

Parameter	Description

Smn | Smn(TYPE) |
S(m,n) | S(m,n)(TYPE)
SXYmn | SXYmn(TYPE) |
SXY(m,n) | SXY(m,n)(TYPE)
	Complex 2-port parameters. Where:
	■ m = 1 or 2
	■ n = 1 or 2
	■ X and Y are used for mixed-mode S-parameter output.
	■ The values for X and Y can be D (differential), C (common), or S (single-end).
	TYPE = R, I, M, P, PD, D, DB, or DBM
	■ R = real
	■ I = imaginary
	■ M = magnitude
	■ P = PD = phase in degrees
	■ D = DB = decibels
	■ DBM = decibels per 1.0e-3

NF SSNF	NF and SSNF both output a single-side band noise figure as a function of the IFB points: NF = SSNF = 10 Log(SSF) Single side-band noise factor, SSF = {(Total Noise at output, at OFB, originating from all frequencies) - (Load Noise originating from OFB)} / (Input Source Noise originating from IFB).
DSNF	DSNF outputs a double side-band noise figure as a function of the IFB points. DSNF = 10 Log(DSF) Double side-band noise factor, DSF = {(Total Noise at output, at the OFB, originating from all frequencies) - (Load Noise originating from the OFB)} / (Input Source Noise originating from the IFB and from the image of IFB).

Output Data Files

An HBLIN analysis produces these output data files:

- The S-parameters from the `.PRINT` statement are written to a `.printhl#` file.
- The extracted S-parameters from the `.PROBE` statement are written to a `.hl#` file.

Large-Signal S-parameter (HBLSP) Analysis

Use the `.HPSLP` command to invoke periodically driven nonlinear circuit analyses for power-dependent S-parameters. An HBLSP analysis provides three kinds of analyses for periodically-driven nonlinear circuits, such as those that employ power amplifiers and filters:

- Two-port power-dependant (large-signal) S-parameter extraction
- Two-port small-signal S-parameter extraction
- Two-port small-signal noise parameter calculation

Unlike small-signal S-parameters, which are based on linear analysis, power-dependent S-parameters are based on harmonic balance simulation. Its solution accounts for nonlinear effects such as compression and variation in power levels.

The definition for power-dependent S-parameters is similar to that for small-signal parameters. Power-dependent S-parameters are defined as the ratio of reflected and incident waves by using this equation:

b = S * a ; S[i, j]=b[i,n]/a[j,n] when a[k,n](k!=j)=0

The incident waves, a[i, n], and reflected waves, b[i, n], are defined by using these equations:

a[i, n] = (V[i](n*W_0) + Z_o[i] * I[i](n*W_0)) / (2 * sqrt(Z_o[i]))

b[i, n] = (V[i](n*W_0) - Z_o[i] * I[i](n*W_0)) / (2 * sqrt(Z_o[i]))

Where:

- W_0 is the fundamental frequency (tone).
- n is a signed integer.
- i is the port number.
- a[i, n] is the input wave at the frequency n*W_0 on the i^{th} port.
- b[i, n] is the reflected wave at the frequency n*W_0 on the i^{th} port.
- V[i](n*W_0) is the Fourier coefficient at the frequency n*W_0 of the voltage at port i.

- I[i]($n*W_0$) is the Fourier coefficient at the frequency $n*W_0$ of the current at port i.
- Z_o[i] is the reference impedance at port i.

An HBLSP analysis only extracts the S-parameters on the first harmonic (that is, n=1).

.HBLSP Syntax

```
.HBLSP NHARMS=nh [POWERUNIT=dbm|watt]
+ [SSPCALC=1|0|YES|NO] [NOISECALC=1|0|YES|NO]
+ [FILENAME=file_name] [DATAFORMAT=ri|ma|db]
+ FREQSWEEP freq_sweep POWERSWEEP power_sweep
```

Argument	Description
NHARMS	Number of harmonics in the HB analysis triggered by the `.HBLSP` command.
POWERUNIT	Power unit. Default is watt.
SSPCALC	Extract small-signal S-parameters. Default is 0 (NO).
NOISECALC	Perform small-signal 2-port noise analysis. Default is 0 (NO).
FILENAME	Output data `.p2d` filename. Default is the netlist name or the object name after the -o command-line option.
DATAFORMAT	Format of the output data file. Default is ma (magnitude, angle).
FREQSWEEP	Frequency sweep specification. A sweep of type LIN, DEC, OCT, POI, or SWEEPBLOCK. Specify the *nsteps*, *start*, and *stop* times using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ *SWEEPBLOCK=swblockname* This keyword must appear before the POWERSWEEP keyword.

Argument	Description
POWERSWEEP	Power sweep specification. A sweep of type LIN, DEC, OCT,POI, or SWEEPBLOCK. Specify the nsteps, start, and stop times using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps power_values* ■ *SWEEPBLOCK=swblockname* This keyword must follow the FREQSWEEP keyword.

Limitations

The HBLSP analysis has these known limitations:

- Power-dependent S-parameter extraction is a 2-port analysis only. Multiport power-dependent S-parameters are not currently supported.
- The intermodulation data block (`IMTDATA`) in the `.p2d` file is not supported.
- The internal impedance of the P (port) Element can only be a real value. Complex impedance values are not supported.

Input Syntax

```
.HBLSP NHARMS=nh [POWERUNIT=[dbm|watt]]
+ [SSPCALC=[1|0|YES|NO]] [NOISECALC=[1|0|YES|NO]]
+ [FILENAME=file_name] [DATAFORMAT=[ri|ma|db]]
+ FREQSWEEP freq_sweep POWERSWEEP power_sweep
```

Parameter	Description
NHARMS	Number of harmonics in the HB analysis triggered by the `.HBLSP` statement.
POWERUNIT	Power unit. Default is watt.
SSPCALC	Extract small-signal S-parameters. Default is 0 (NO).
NOISECALC	Perform small-signal 2-port noise analysis. Default is 0 (NO).
FILENAME	Output data .p2d filename. Default is the netlist name or the object name after the -o command-line option.

Parameter	Description
DATAFORMAT	Format of the output data file. Default is ma (magnitude, angle).
FREQSWEEP	Frequency sweep specification. A sweep of type LIN, DEC, OCT, POI, or SWEEPBLOCK. Specify the *nsteps*, *start*, and *stop* times using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps freq_values* ■ *SWEEPBLOCK=swblockname* This keyword must appear before the POWERSWEEP keyword.
POWERSWEEP	Power sweep specification. A sweep of type LIN, DEC, OCT,POI, or SWEEPBLOCK. Specify the nsteps, start, and stop frequencies using the following syntax for each type of sweep: ■ LIN *nsteps start stop* ■ DEC *nsteps start stop* ■ OCT *nsteps start stop* ■ POI *nsteps power_values* ■ *SWEEPBLOCK=swblockname* This keyword must follow the FREQSWEEP keyword.

Note: The `FREQSWEEP` and `POWERSWEEP` keywords must appear at the end of an `.HBLSP` statement.

Examples

Example 1 does 2-port single-tone, power-dependent S-parameter extraction, without frequency translation:

- Frequency sweep: The fundamental tone is swept from 0 to 1G
- Power sweep: The power input at port 1 is swept from 6 to 10 Watts.
- Five harmonics are required for the HB analysis. Large-signal S-parameters are extracted on the first harmonic.
- Five harmonics are required in the HBLSP triggered HB analysis.
- The DC value in p1 statement is used to set DC bias, which is used to perform small-signal analyses.
- Small-signal S-parameters are required extracted.
- Small-signal two-port noise analysis is required.
- The data will be output to the `ex1.p2d` file.

Example 16 2-Port, Single Tone

```
   p1 1 0 port=1 dc=1v
   p2 2 0 port=2
   .hblsp nharms=5 powerunit = watt
   + sspcalc=1 noisecalc=1 filename=ex1
   + freqsweep lin 5 0 1G powersweep lin 5 6 10
```

Example 2 generates large scale S-parameters as a function of input for a differential equalizer.

Example 17 4-Port Network

```
* hblsp example
.opt post
p1 n1 0 port=1 ac=1
p2 n2 0 port=2
*** put your DUT
R1 n1 n2 10***
.hblsp nharms=5
+ freqsweep lin 4 1k 10k
+ powersweep lin 2 5 10
.end
```

Output Syntax

This section describes the syntax for the `HBLSP .PRINT` and `.PROBE` statements. These statements only support S and noise parameter outputs. Node voltage, branch current, and all other parameters are not supported in `HBLSP .PRINT` and `.PROBE` statements.

.PRINT and .PROBE Statements

```
.PRINT HBLSP Smn | Smn(TYPE) | S(m, n) | S(m, n)(TYPE)
+ ...small signal 2-port noise params...
.PROBE HBLSP Smn | Smn(TYPE) | S(m, n) | S(m, n)(TYPE)
```

```
+ ...small signal 2-port noise params...
```

Parameter	Description

Smn | Smn(TYPE) | S(m,n) | S(m,n)(TYPE)	Complex 2-port parameters. Where:
	■ m = 1 or 2
	■ n = 1 or 2
	■ TYPE = R, I, M, P, PD, D, DB, or DBM
	R = real
	I = imaginary
	M = magnitude
	P = PD = phase in degrees
	D = DB = decibels
	DBM = decibels per 1.0e-3

... small signal 2-port noise parameters ...	G_AS | NF | RN | YOPT | GAMMA_OPT | NFMIN | VN2 | ZCOR | GN | RHON | YCOR | ZOPT | IN2 For a description of these parameters, see Linear Network Parameter Analysis in the *HSPICE User Guide: Signal Integrity Modeling and Analysis*.

Output Data Files

An HBLSP analysis produces these output data files:

- The large-signal S-parameters from the `.PRINT` statement are written to a `.printls#` file.
- The small-signal S-parameters from the `.PRINT` statement are written to a `.printss#` file.
- The large-signal S-parameters from the `.PROBE` statement are written to a `.ls#` file.
- The small-signal S-parameters from the `.PROBE` statement are written to a .ss# file.
- The extracted large- and small-signal S and noise parameters are written to a `.p2d` file.

The large-signal and small-signal S-parameters from the `.PROBE` statement are viewable in Custom WaveView.

7 Envelope Analysis

Describes how to use envelope simulation.

These topics are covered in the following sections:

- Envelope Simulation
- Envelope Analysis Commands
- Nonautonomous Form
- Oscillator Analysis Form
- Fast Fourier Transform Form
- Output Syntax

Envelope Simulation

Envelope simulation combines features of time- and frequency-domain analysis. Harmonic Balance (HB) solves for a static set of phasors for all the circuit state variables, as shown in this equation:

Equation 45

$$v(t) = a_0 + \sum_{i=1}^{N} [a_i \cos\omega_i t + b_i \sin\omega_i t]$$

In contrast, envelope analysis finds a dynamic, time-dependent set of phasors, as this equation shows:

Equation 46

$$v(t) = a_0(\hat{t}) + \sum_{i=1}^{N} [a_i(\hat{t})\cos\omega_i t + b_i(\hat{t})\sin\omega_i t]$$

Thus, in envelope simulation, each signal is described by the evolving spectrum. Envelope analysis is generally used on circuits excited by signals with significantly different timescales. An HB simulation is performed at each point in time of the slower-moving ($\hat{t}$) timescale. In this way, for example, a 2-tone HB simulation can be converted into a series of related 1-tone simulations where the transient analysis proceeds on the ($\hat{t}$) timescale, and 1-tone HB simulations are performed with the higher frequency tone as the fundamental frequency.

In HSPICE advanced analog analyses, any voltage or current source identified as a HB source either in a V or I element statement, or by an `.OPTION TRANFORHB` command, is used for HB simulations at each point in $\hat{t}$ time. All other sources are associated with the transient timescale. Also, the input waveforms can be represented in the frequency domain as RF carriers modulated by an envelope by identifying a VMRF signal source in a V or I element statement. The amplitude and phase values of the sampled envelope are used as the input signal for HB analysis.

Some typical applications for envelope simulation are amplifier spectral regrowth, adjacent channel power ration (ACPR), and oscillator startup and shutdown analyses.

Envelope Analysis Commands

This section describes those commands specific to envelope analysis. These commands are:

- Standard envelope simulation (`.ENV`)
- Oscillator simulation, both startup and shutdown (`.ENVOSC`)
- Envelope Fast Fourier Transform (`.ENVFFT`)

Nonautonomous Form

```
.ENV TONES=f1 [f2...fn] NHARMS=h1 [h2...hn]
```

```
+ ENV_STEP=tstep ENV_STOP=tstop
```

Parameter	Description
TONES	Carrier frequencies, in hertz.
NHARMS	Number of harmonics.
ENV_STEP	Envelope step size, in seconds.
ENV_STOP	Envelope stop time, in seconds.

Description

You use the `.ENV` command to do standard envelope simulation. The simulation proceeds just as it does in standard transient simulation, starting at `time=0` and continuing until `time=env_stop`. An HB analysis is performed at each step in time. You can use Backward-Euler (BE), trapezoidal (TRAP), or level-2 Gear (GEAR) integration.

Recommended option settings are:

- For BE integration, set `.OPTION SIM_ORDER=1`.
- For TRAP, set `.OPTION SIM_ORDER=2` (default) `METHOD=TRAP` (default).
- For GEAR, set `.OPTION SIM_ORDER=2` (default) `METHOD=GEAR`.

Example

```
.env tones=1e9 nharms=6 env_step=10n env_stop=1u
```

Oscillator Analysis Form

```
.ENVOSC TONE=f1 NHARMS=h1 ENV_STEP=tstep ENV_STOP=tstop
+ PROBENODE=n1,n2,vosc <FSPTS=num, min, max>
```

Parameter	Description
TONE	Carrier frequencies, in hertz.
NHARMS	Number of harmonics.
ENV_STEP	Envelope step size, in seconds.
ENV_STOP	Envelope stop time, in seconds.
PROBENODE	Defines the nodes used for oscillator conditions and the initial probe voltage value.

Parameter	Description
FSPTS	Specifies the frequency search points used in the initial small-signal frequency search. Usage depends on oscillator type.

Description

You use the `.ENVOSC` command to do envelope simulation for oscillator startup or shutdown.

Oscillator startup or shutdown analysis with this command must be helped along by converting a bias source from a DC description to a PWL description that either:

- Starts at a low value that supports oscillation and ramps up to a final value (startup simulation)
- Starts at the DC value and ramps down to zero (shutdown simulation).

In addition to solving for the state variables at each envelope time point, the `.ENVOSC` command also solves for the frequency. This command is intended to be applied to high-Q oscillators that take a long time to reach steady-state. For these circuits, standard transient analysis is too costly. Low-Q oscillators, such as typical ring oscillators, are more efficiently simulated with standard transient analysis.

Example

```
.envosc tone=250Meg nharms=10 env_step=20n env_stop=10u
+ probenode=v5,0,1.25
```

Fast Fourier Transform Form

```
.ENVFFT output_var [NP=val] [FORMAT=keyword]
+[<WINDOW=keyword] [ALFA=val]
```

Parameter	Description
output_var	Any valid output variable.
NP	The number of points to use in the FFT analysis. NP must be a power of 2. If not a power of 2, then it is automatically adjusted to the closest higher number that is a power of 2. The default is 1024.

FORMAT	Specifies the output format:
	
	NORM= normalized magnitude
	UNORM=unnormalized magnitude (default)

Parameter	Description

WINDOW	Specifies the window type to use:
	
	RECT=simple rectangular truncation window (default)
	BART=Bartlett (triangular) window
	HANN=Hanning window
	HAMM=Hamming window
	BLACK=Blackman window
	HARRIS=Blackman-Harris window
	GAUSS=Gaussian window
	KAISER=Kaiser-Bessel window

ALFA	Controls the highest side-lobe level and bandwidth for GAUSS and KAISER windows. The default is 3.0.

Description

You use the `.ENVFFT` command to perform Fast fourier Transform (FFT) on envelope output. This command is similar to the `.FFT` command. The only difference is that transformation is performed on real data with the `.FFT` command, and with the `.ENVFFT` command, the data being transformed is complex. You usually want to do this for a specific harmonic of a voltage, current, or power signal.

Example

```
.envfft v(out)[1]
```

Output Syntax

The results from envelope simulation can be made available through the `.PRINT`, `.PROBE`, and `.MEASURE` commands. This section describes the basic syntax you can use for this purpose.

.PRINT or .PROBE

You can print or probe envelope simulation results by using the following commands:

```
.PRINT ENV ov1 <ov2... >
.PROBE ENV ov1 <ov2... >
```

Where `ov1...` are the output variables to print or probe.

.MEASURE

In HSPICE advanced analog analyses, the independent variable for envelope simulation is the first tone. Otherwise and except for the analysis type, the `.MEASURE` statement syntax is the same as the syntax for HB; for example,

```
.MEASURE ENV result ...
```

Envelope Output Data File Format

The results of envelope simulations are written to *.ev# data files by the `.PROBE` statement. The format of an *.ev# data file is equivalent to an *.hb# data file with the addition of one fundamental parameter sweep that represents the slowly-varying time-envelope variation $\hat{t}$ of the Fourier coefficients and frequencies. You can recognize this swept parameter” in the *.ev# file by the keyword `env_time`.

Each row in the tabulated data of an *.ev# file includes values for identifying frequency information, the complex data for the output variables, and information on the envelope time sweep. For example, the header for a data file dump for output variables v(in) and v(out) that follow a 2-tone envelope analysis, have entries for:

```
hertz  v(in)  v(out)  n0  f0  n1  f1  sweep  env_time  $&%#
```

Which result in data blocks with floating point values following:

```
env_time[0]
f[0]  a[0]{v(in)}  b[0] {v(in)}  a[0] {v(out)}  b[0] {v(out)}  n0
f0  n1  f1
f[1]  a[1]{v(in)}  b[1] {v(in)}  a[1] {v(out)}  b[1] {v(out)}  n0
f0  n1  f1
...
f[N]  a[N]{v(in)}  b[N] {v(in)}  a[N] {v(out)}  b[N] {v(out)}  n0
f0  n1  f1

env_time[1]
f[0]  a[0]{v(in)}  b[0] {v(in)}  a[0] {v(out)}  b[0] {v(out)}  n0
f0  n1  f1
f[1]  a[1]{v(in)}  b[1] {v(in)}  a[1] {v(out)}  b[1] {v(out)}  n0
f0  n1  f1
...
f[N]  a[N]{v(in)}  b[N] {v(in)}  a[N] {v(out)}  b[N] {v(out)}  n0
f0  n1  f1

...

env_time[M-1]
f[0]  a[0]{v(in)}  b[0] {v(in)}  a[0] {v(out)}  b[0] {v(out)}  n0
f0  n1  f1
f[1]  a[1]{v(in)}  b[1] {v(in)}  a[1] {v(out)}  b[1] {v(out)}  n0
f0  n1  f1
...
f[N]  a[N]{v(in)}  b[N] {v(in)}  a[N] {v(out)}  b[N] {v(out)}  n0
f0  n1  f1
```

Where there are M data blocks corresponding to M envelope time points, with each block containing N+1 rows for the frequency data. The units for the `env_time` sweep are seconds.

8

Post-Layout Analysis

Describes the post-layout analysis flow, including post-layout back-annotation, DSPF and SPEF files, linear acceleration, check statements, and power analysis.

These topics are covered in the following sections:

- Post-Layout Back-Annotation
- Linear Acceleration Control Options Summary

Post-Layout Back-Annotation

A traditional, straightforward, “brute-force” flow runs an RC extraction tool that produces a detailed standard parasitic format (DSPF) file. DSPF is the standard format for transferring RC parasitic information. This traditional flow then feeds this DSPF file into the circuit simulation tool for post-layout simulation.

A key problem is that the DSPF file is flat. Accurately simulating a complete design, such as an SRAM or an on-chip cache, is a waste of workstation memory, disc space usage, and simulation runtime. Because this DSPF file is flat, control and analysis are limited.

- How do you set different options for different blocks for better trade-off between speed and accuracy?
- How do you perform a power analysis on a flat netlist to check the power consumption?

- This traditional flow flattens all nodes after extraction so it is more difficult to compare the delay before and after extraction.
- This traditional flow can also stress the limits of an extraction tool so reliability also becomes an issue.

HSPICE advanced analog analyses provides a flow that solves all of these problems.

- Star-RCXT generates a hierarchical Layout Versus Schematic (LVS) ideal netlist, and flat information about RC parasitics in a DSPF or (standard parasitic exchange format (SPEF) file.
- HSPICE advanced analog analyses uses the hybrid flat-hierarchical approach to back-annotate the RC parasitics, from the DSPF or SPEF file, into the hierarchical LVS ideal netlist.

Using the hierarchical LVS ideal netlist cuts simulation runtime and CPU memory usage. Because HSPICE advanced analog analyses uses the hierarchical LVS ideal netlist as the top-level netlist, you can fully control the netlist. For example:

- You can set different modes to different blocks for better accuracy and speed trade-off.
- You can run power analysis, based on the hierarchical LVS ideal netlist, to determine the power consumption of each block. If you use the hierarchical LVS ideal netlist, you can reuse all post-processing statements from the pre-layout simulation for the post-layout simulation. This saves time, and the capacity of the verification tool is not stressed so reliability is higher.

HSPICE advanced analog analyses supports only the XREF:COMPLETE flow and the XREF:NO flow from Star-RCXT. Refer to the *Star-RCXT User Guide* for more information about the XREF flow.

To generate a hierarchical LVS ideal netlist with Star-RCXT, include the following options in the Star-RCXT command file.

```
*** for XREF:NO flow ***
NETLIST_IDEAL_SPICE_FILE: ideal_spice_netlist.sp
NETLIST_IDEAL_SPICE_TYPE: layout
NETLIST_IDEAL_SPICE_HIER:YES

*** for XREF:COMPLETE flow ***
NETLIST_IDEAL_SPICE_FILE: ideal_spice_netlist.sp
NETLIST_IDEAL_SPICE_TYPE: schematic
NETLIST_IDEAL_SPICE_HIER:YES
```

Note: Before version 2002.2, Star-RCXT used `NETLIST_IDEAL_SPICE_SKIP_CELLS` to generate the hierarchical ideal SPICE netlist. HSPICE advanced analog analyses can still simulate post-layout designs using the brute-force flow, but the post-layout flow is preferable in HSPICE advanced analog analyses.

HSPICE advanced analog analyses supports the following post-layout flows to address post-layout simulation scenarios.

- Standard Post-Layout Flow
- Selective Post-Layout Flow
- Additional Post-Layout Options

Standard Post-Layout Flow

Use this flow mainly for analog or mixed signal design, and high-coverage verification runs when you need to back-annotate RC parasitics into the hierarchical LVS ideal netlist. In this flow, HSPICE advanced analog analyses expands all nets from the DSPF or SPEF file. To expand only selected nets, use see Selective Post-Layout Flow.

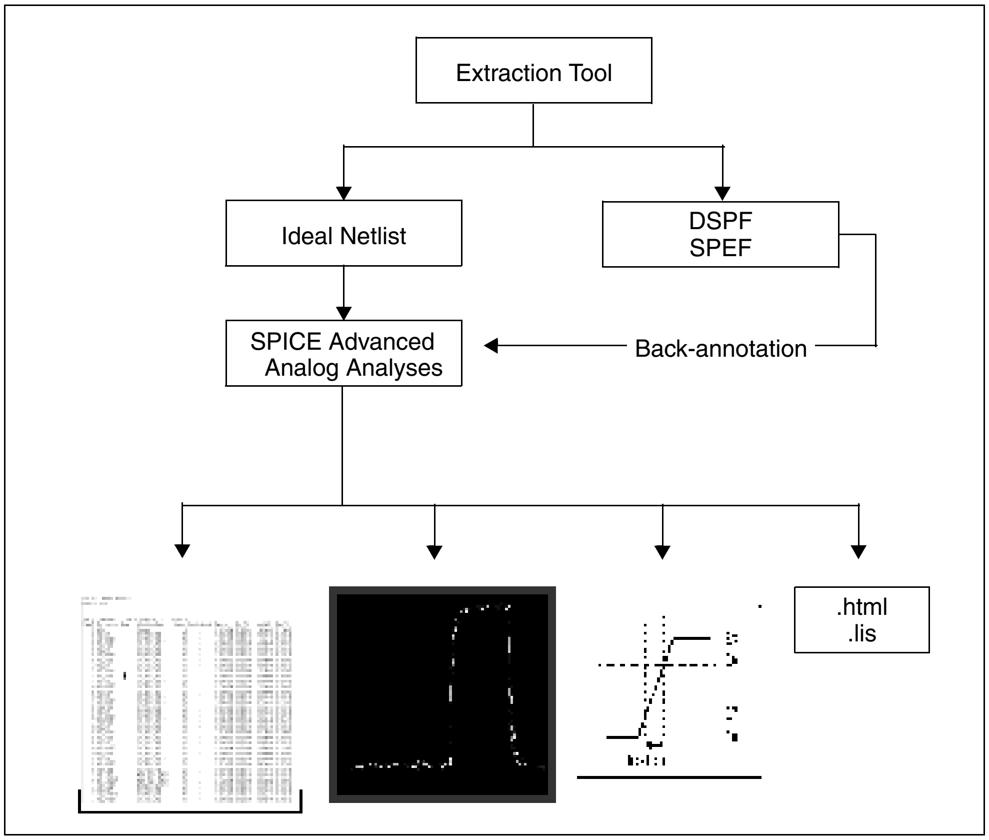


Figure 29 Standard Post-Layout Flow

Standard Post-Layout Flow Control Options

The standard post-layout flow options are `SIM_DSPF` and `SIM_SPEF`. Include one of these options in your netlist. For example,

```
.OPTION SIM_DSPF="[scope] dspf_filename"
.OPTION SIM_SPEF="spec_filename"
```

In the `SIM_DSPF` syntax, `scope` can be a subcircuit definition or an instance. If you do not specify `scope`, it defaults to the top-level definition. HSPICE advanced analog analyses requires both a DSPF file and an ideal netlist. Only flat DSPF files are supported; hierarchy statements, such as `.SUBCKT` and .x1, are ignored.

Very large circuits generate very large DSPF files; this is when using either the `SIM_DSPF` or the `SIM_DSPF_ACTIVE` option can really improve performance.

You can specify a DSPF file in the `SIM_SPEF` option, or a SPEF file in the `SIM_DSPF` option. The `scope` function is not supported in the SPEF format.

For descriptions and usage examples, see .OPTION SIM_DSPF and .OPTION SIM_SPEF in the *HSPICE Reference Manual: Commands and Control Options*.

Example

```
$  models
.MODEL p pmos
.MODEL n nmos
.INCLUDE add4.dspf
.OPTION SIM_DSPF="add4.dspf"
.VEC "dspf_adder.vec"
.TRAN 1n 5u
vdd vdd 0 3.3
.OPTION POST
.END
```

SIM_DSPF With SIM_LA Option

The `SIM_DSPF` option accelerates the simulation by more than 100%. By using the `SIM_LA` option at the same time, you can further reduce the total CPU time:

```
$  models
.MODEL p pmos
.MODEL n nmos
.INCLUDE add4.dspf
.OPTION SIM_DSPF="add4.dspf"
.OPTION SIM_LA=PACT
.VEC "dspf_adder.vec"
.TRAN 1n 5u
vdd vdd 0 3.3
.OPTION POST
.END
```

To expand only active nodes, such as those that move, include the `SIM_DSPF_ACTIVE` option in your netlist. For example:

```
.OPTION SIM_DSPF_ACTIVE="active_net_filename"
```

This option is most effective when used with a large design—for example, over 5K transistors. Smaller designs lose some of the performance gain, due to internal overhead processing.

For syntax and description of `SIM_DSPF_ACTIVE` option, see .OPTION SIM_DSPF_ACTIVE in the *HSPICE Reference Manual: Commands and Control Options*.

When you have included the appropriate control option, run HSPICE advanced analog analyses, using the ideal netlist.

The structure of a DSPF file is:

```
*|DSPF 1.0
*|DESIGN “demo”
*|Date “October 6, 1998”
...
.SUBCKT < name > < pins >
* Net Section
C1 ...
R1 ...
...
* Instance Section
...
.ENDS
```

Selective Post-Layout Flow

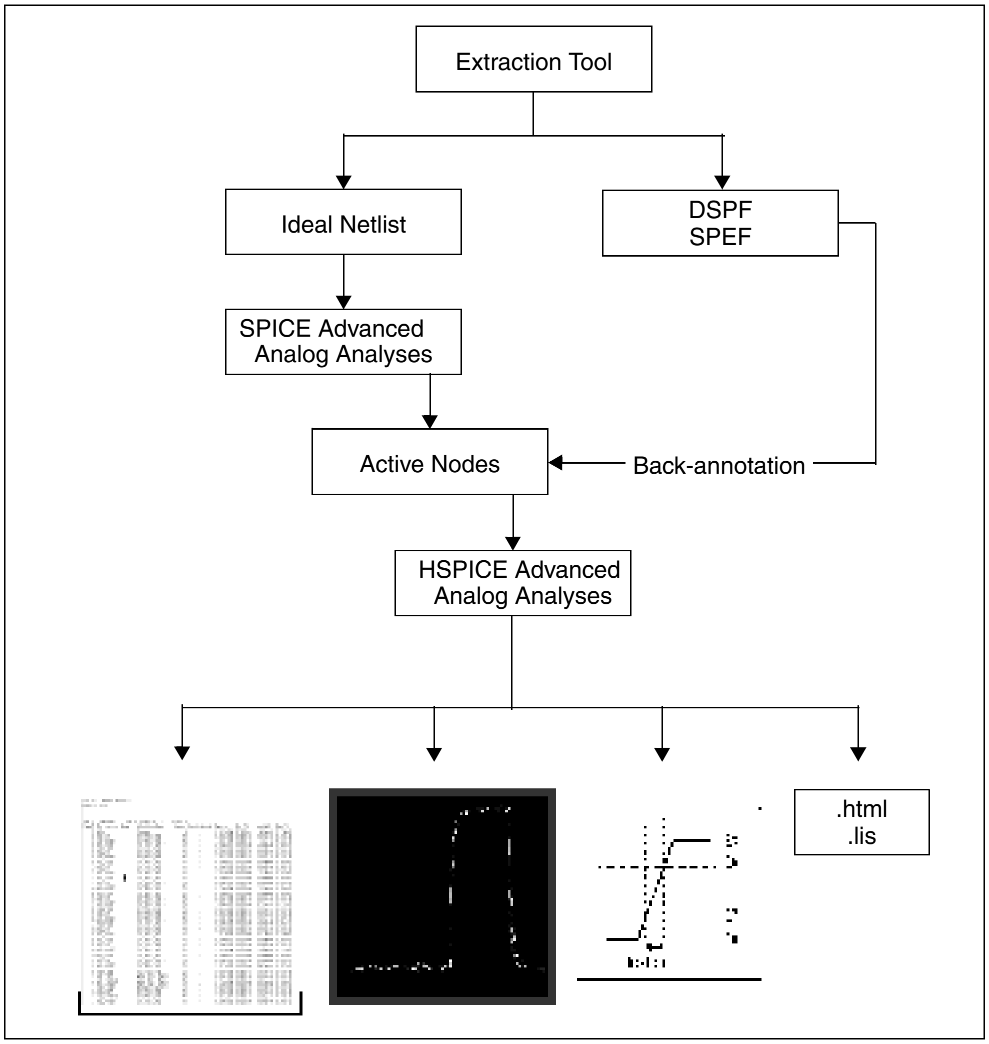


Figure 30 Selective Post-Layout Flow

You can use the selective post-layout flow to simulate a post-layout design for a memory or digital circuit, and for a corner-point verification run. Instead of back-annotating all RC parasitics into the ideal netlist, the selective post-layout flow automatically detects and back-annotates only active parasitics, into the hierarchical LVS ideal netlist. For a high-latency design, the selective post-layout flow is an order of magnitude faster than the standard post-layout flow.

Note: The selective post-layout flow applies only to advanced analog transient analyses and cannot be used with other analyses such as DC, AC, or HB.

Selective Post-Layout Flow Control Options

To invoke the selective post-layout flow, include one of the options listed in Table 3 in your netlist.

Table 3 Selective Post-Layout Flow Options

Syntax	Description
SIM_DSPF_ACTIVE -or- SIM_SPEF_ACTIVE	HSPICE advanced analog analyses performs a preliminary verification run to determine the activity of the nodes and generates two ASCII files: active_node.rc and active_node.rcxt. These files save all active node information in both Star-RC format and Star-RCXT format. By default, a node is considered active if the voltage varies by more than 0.1V. To change this value, use the SIM_DSPF_VTOL or SIM_SPEF_VTOL option. For descriptions and usage examples, see OPTION SIM_DSPF_ACTIVE and .OPTION SIM_SPEF_ACTIVE in the *HSPICE Reference Manual: Commands and Control Options*.
SIM_DSPF_VTOL -or- SIM_SPEF_VTOL	HSPICE advanced analog analyses performs a second simulation run by using the *active_node* file, the DSPF or SPEF file, and the hierarchical LVS ideal netlist to back-annotate only active portions of the circuit. If a net is latent, then HSPICE advanced analog analyses does not expand the net. This saves simulation runtime and memory. ■ *value* is the tolerance of the voltage change. ■ *scopen* can be a subcircuit definition (which has an @ prefix), or a subcircuit instance. By default, HSPICE advanced analog analyses performs only one iteration of the second simulation run. Use the SIM_DSPF_MAX_ITER or SIM_SPEF_MAX_ITER option to change it. For descriptions and usage examples, see .OPTION SIM_DSPF_VTOL and .OPTION SIM_SPEF_VTOL in the *HSPICE Reference Manual: Commands and Control Options*.

Table 3 Selective Post-Layout Flow Options (Continued)

Syntax	Description
SIM_DSPF_MAX_ITER -or- SIM_SPEF_MAX_ITER	value is the maximum number of iterations for the second simulation run. Some of the latent nets might turn active after the first iteration of the second run. In this case: ■ Resimulate the netlist to ensure the accuracy of the post-layout simulation. ■ Use SIM_DSPF_MAX_ITER or SIM_SPEF_MAX_ITER to set the maximum number of iterations for the second run. If the active_node remains the same after the second simulation run, HSPICE advanced analog analyses ignores these options. For descriptions and usage examples, see .OPTION SIM_DSPF_MAX_ITER and .OPTION SIM_SPEF_MAX_ITER in the *HSPICE Reference Manual: Commands and Control Options*.

Additional Post-Layout Options

Other post-layout options are listed in Table 4.

Table 4 Additional Post-Layout Options

Syntax	Description
SIM_DSPF_RAIL -or- SIM_SPEF_RAIL	By default, HSPICE advanced analog analyses does not back-annotate parasitics of the power-net. To back-annotate power-net parasitics, include one of these options in the netlist. Default=OFF. ON expands nets in a power rail as it expands all nets.
SIM_DSPF_SCALER SIM_SPEF_SCALER -or- SIM_DSPF_SCALEC SIM_SPEF_SCALEC	Scales the resistance or capacitance values. ■ scaleR is the scale factor for resistance ■ scaleC is the scale factor for capacitance.
SIM_DSPF_LUMPCAPS -or- SIM_SPEF_LUMPCAPS	If HSPICE advanced analog analyses cannot back-annotate an instance in a net because one or more instances are missing in the hierarchical LVS ideal netlist, then by default HSPICE advanced analog analyses does not evaluate the net. Instead of ignoring all parasitic information for this net, HSPICE advanced analog analyses includes these options to connect a lumped capacitor with a value equal to the net capacitance to this net. Default = ON adds lumped capacitance; ignores other net contents.

Table 4 Additional Post-Layout Options (Continued)

Syntax	Description
SIM_DSPF_INSERROR -or- SIM_SPEF_INSERROR	HSPICE advanced analog analyses supports options to skip the unmatched instance, and continue the evaluation of the next instance. The default is OFF. ON skips unmatched instances and continues the evaluation.
SIM_SPEF_PARVALUE	This option affects only values in a SPEF file that have triplet format: *float:float:float*, which this option interprets as *best:average:worst*. In such cases: ■ If SIM_SPEF_PARVALUE=1, HSPICE advanced analog analyses uses best. ■ If SIM_SPEF_PARVALUE=2 (default), HSPICE advanced analog analyses uses average. ■ If SIM_SPEF_PARVALUE=3, HSPICE advanced analog analyses uses worst.

Unsupported SPEF Options

HSPICE advanced analog analyses does not yet support the following IEEE-481 SPEF options:

- Hierarchical SPEF definition (multiple SPEF files connected with a hierarchical definition):
- `*DEFINE` and `*PDEFINE`
- `*R_NET` and `*R_PNET` definition
- `*D_PNET` definition.

Selective Extraction Flow

Use the selective extraction flow if disk space is limited. Especially use this option when simulating a full-chip post-layout design, where block latency is high. HSPICE advanced analog analyses feedbacks the active net information to Star-RCXT to extract only the active parasitic.

The major advantage of this flow is a smaller DSPF or SPEF file, which saves disk space.

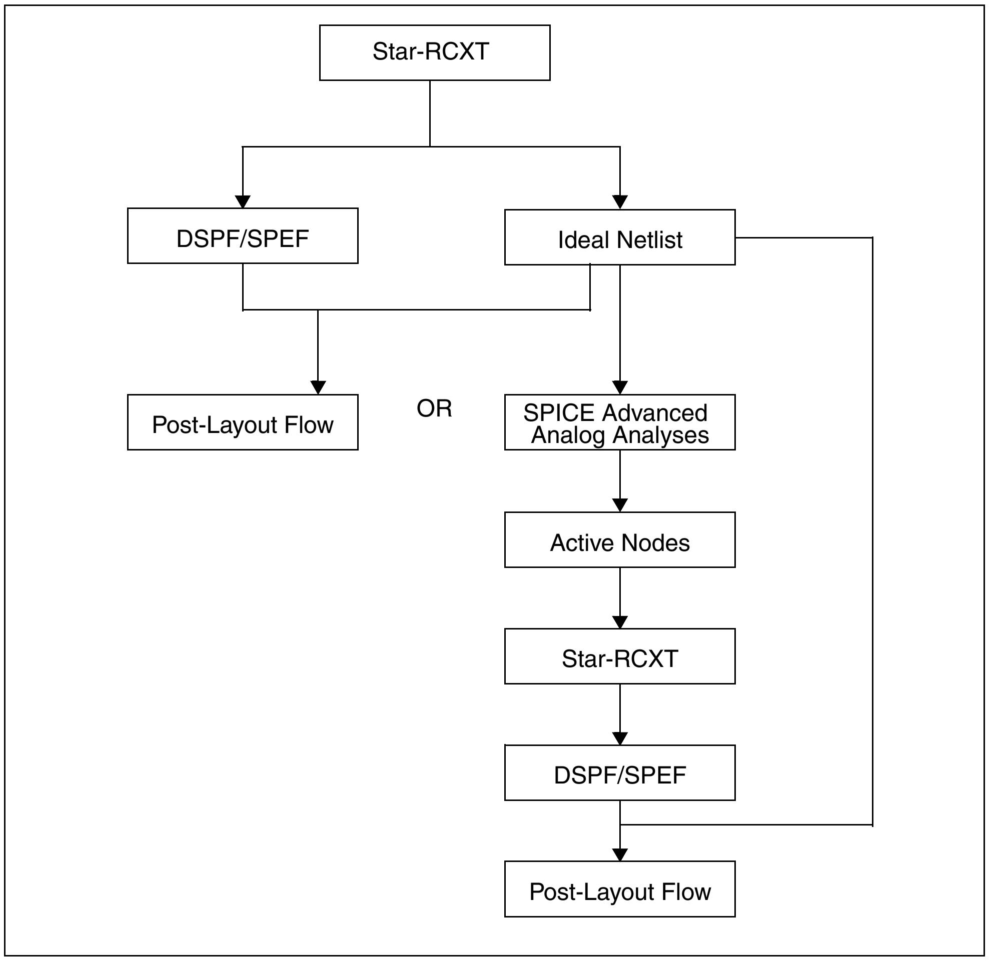


Figure 31   Selective Extraction Flow

Note: HSPICE advanced analog analyses generates an active node file in both Star-RC and Star-RCXT format. It then expands the active node file to the Star-RCXT command file to extract only active parasitics.

Overview of DSPF Files

In general, an SPF (Standard Parasitic Format) file describes interconnect delay and loading, due to parasitic resistance and capacitance. DSPF (Detailed Standard Parasitic Format) is a specific type of SPF file that describes the actual parasitic resistance and capacitance components of a net. DSPF is a standard output format commonly used in many parasitic extraction tools, including Star-RCXT. The HSPICE advanced analog circuit simulator can read DSPF files.

DSPF File Structure

The DSPF standard is published by Open Verilog International (OVI). For information about how to obtain the complete DSPF specification, or any other documents from OVI, see:

http://www.ovi.org/document.html

The OVI DSPF specification requires the following file structure in a DSPF file. Parameters in {braces} are optional:

```
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}
```

Table 5   DSPF Parameters

Parameter	Definition
*|DSPF	Specifies that the file is in DSPF format.
{version}	Version number of the DSPF specification (optional).
*|	Words that start with *| are keywords.
||	Or (use the option either preceding or following ||). For example, *|P || *I means you can use either the *|P option or the *|I option.
design_name	Name of your circuit design (optional).
date	Date and time when a parasitic extraction tool (such as Star-RCXT) generated the DSPF file (optional).
vendor	Name of the vendor (such as Synopsys) whose tools you used to generate the DSPF file (optional).
program_name	Name of the program (such as Star-RCXT) that generated the DSPF file (optional).
program_version	Version number of the program that generated the DSPF file (optional).
divider	Character that divides levels of hierarchy in a circuit path (optional). If you do not define this parameter, the default hierarchy divider is a slash (/). For example, X1/X2 indicates that X2 is a subcircuit of the X1 circuit.
delimiter	Character used to separate the name of an instance and a pin in a concatenated instance pin name, or a net name and a sub-node number in a concatenated sub-node name. If you do not define this parameter, the default delimiter is a colon (:).

Table 5 DSPF Parameters (Continued)

Parameter	Definition
path	Hierarchical path to a net, instance, or pin, within a circuit.
net_name	Name of a net in a circuit or subcircuit.
instance_name	Name of an instance of a subcircuit.
pin_name	Name of a pin on an instance of a subcircuit.
pinCap	Capacitance of a pin.
pin_type	■ I (input) ■ O (output) ■ B (bidirectional) ■ X (don’t care) ■ S (switch) ■ J (jumper)
resistance	Resistance on a pin in ohms for input (I), output (O), or bidirectional (B) pins. You can use resistance-capacitance (RC) pairs to model pin characteristics by using a higher-order equivalent RC ladder circuit than a single capacitor model. For example: C0 {R1 C1 R2 C2...}. Attaching RC pairs increases the order of the equivalent circuit from the first (C0) order. For X, S, and J pin types, simulation ignores this generalized capacitance value, but you should insert a 0 value as a place-holder for format integrity. The resistance value can be a real number or an exponent (optionally followed by a real number). You can enter an O (ohms) after the value.
capacitance	Capacitance on a pin in farads for input (I), output (O), or bidirectional (B) pins. Use as part of a resistance-capacitance (RC) pair. Optionally enter an F (farads) after the value.
unit	■ K (kilo) ■ M (milli) ■ U (micro) ■ N (nano) ■ P (pico) ■ F (femto)
x_coordinate	Location of a pin relative to the x (horizontal) axis.
y_coordinate	Location of a pin relative to the y (vertical) axis.
capacitor_ statements	SPICE-type statements that define capacitors in the subcircuit.
resistor_ statements	SPICE-type statements that define resistors in the subcircuit.

Table 5 DSPF Parameters (Continued)

Parameter	Definition
subcircuit_call_ statements	Statements that call the subcircuit from higher-level circuits.
.END	Marks the end of the file (optional).

DSPF File Example

```
*|DSPF 1.0
*|DESIGN "my_circuit"
*|DATE June 15, 2002 14:12:43
*|VENDOR "Synopsys"
*|PROGRAM "Star-RC"
*|VERSION "Star-RCXT 2002.2"
*|DIVIDER /
*|DELIMITER :
.SUBCKT BUFFER OUT IN
* Description of Nets
*GROUND_NET VSS
*|NET IN 1.221451PF
*|P(IN 1 0.0 0 10)
*|I(DF1:A DF1 A I 0.0PF 10.0 10.0)
*|I(DF1:B DF1 B I 0.0PF 10 0 20.0)
*|S(IN:1 5.0 10.0)(IN:2 5.0 20.0)
   C1 IN VSS 0.117763PF
   C2 IN:1 VSS 0.276325PF
   C3 IN:2 VSS 0.286325PF
   C4 DF1:A VSS 0.270519PF
   C5 DF1:B VSS 0.270519PF
   R20 IN N:1 1.70333E00
   R21 IN:1 DF1:A 1.29167E-01
   R22 IN:1 IN:2 1.29167E-01
   R23 IN:2 DF1:B 1.70333E-01
*|NET BF 0.287069PF
*|I(DF1:C DF1 C O 0.0PF 12.0 15.0)
*|I(INV1:IN INV1 IN I 0.0PF 30.0 15.0)
   C6 DF1:C VSS 0.208719PF
   C7 INV1:IN VSS 0.783500PF
   R24 DF1:C INV1:IN 1.80833E-01
*|NET OUT 0.148478PF
*|S(OUT:1 45.0 15.0)
*|P(OUT O 0.0PF 50.0 5.0)
*|I(INV1:OUT INV1 OUT O 0.0PF 40.0 15.0)
   C8 INV1:OUT VSS 0.147069PF
   C9 OUT:1 VSS 0.632813PF
   C10 OUT VSS 0.776250PF
   R25 INV1:OUT OUT:1 3.11000E00
   R26 OUT:1 OUT 3.03333E00

* Description of Instances
XDF1 DF1:A DF1:B DF1:C DFF
XINV1 INV1:IN INV1:OUT INV
.ENDS
.END
```

Overview of SPEF Files

The Standard Parasitics Exchange Format (SPEF) file structure is described in IEEE standard *IEEE-1481*. For information about how to obtain the complete SPEC (*IEEE-1481*) specification, or any other documents from IEEE, see:

http://www.ieee.org/products/onlinepubs/stand/standards.html

SPEF File Structure

The IEEE-1481 specification requires the following file structure in a 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
```

Table 6 SPEF Parameters

Parameter	Definition
*SPEF	Specifies that the file is in SPEF format.
{version}	Version number of the SPEF specification, such as “IEEE 1481-1998”.
*	Words that start with an asterisk (*) are keywords.
|	Or. For example, NS|PS means choose either nanoseconds or picoseconds as the time units.
design_name	Name of your circuit design.
date	Date and time when a parasitic extraction tool (such as Star-RCXT) generated the SPEF file.
vendor	Name of the vendor (such as Synopsys) whose tools you used to generate the SPEF file (optional).
program_name	Name of the program (such as Star-RCXT) that generated the SPEF file.
program_version	Version number of the program that generated the SPEF file.

Table 6 SPEF Parameters (Continued)

Parameter	Definition
flow_type	One or more of the following flow types: ■ EXTERNAL_LOADS: The SPEF file defines all external loads (if any). If you do not specify this flow type, then some or all external loads are not defined in this SPEF file. If HSPICE advanced analog analyses cannot find external load data outside the SPEF file, it reports an error. ■ EXTERNAL_SLEWS: The SPEF file defines all external slews (if any). If you do not specify this flow type, then some or all external slews are not defined in this SPEF file. If HSPICE advanced analog analyses cannot find external slew data outside the SPEF file, it reports an error. ■ FULL_CONNECTIVITY: A SPEF file defines all net connectivity. If you do not specify this flow type, then some or all net connectivity is not defined in this SPEF file. If HSPICE advanced analog analyses cannot find connectivity data outside the SPEF file, it issues an error. This flow does not look for presence or absence of power and ground nets, or any other nets that do not correspond to the logical netlist. If a SPEC file includes FULL_CONNECTIVITY and MISSING_NETS, HSPICE advanced analog analyses reports an error. ■ MISSING_NETS: If any logical nets are not defined in the netlist, HSPICE advanced analog analyses merges missing parasitic data from another source. If it does not find another source, HSPICE advanced analog analyses rereads the netlist and estimates the missing parasitics. This flow does not look for presence or absence of power and ground nets, or any other nets that do not correspond to the logical netlist. If you use FULL_CONNECTIVITY and MISSING_NETS in the same SPEF file, HSPICE advanced analog analyses reports an error. ■ NETLIST_TYPE_VERILOG, NETLIST_TYPE_VHDL87, NETLIST_TYPE_VHDL93, or NETLIST_TYPE_EDIF: Specifies the type of naming conventions used in the SPEF file. If you specify more than one format in one SPEF file, HSPICE advanced analog analyses reports an error. ■ ROUTING_CONFIDENCE *positive_integer*: Specifies a default routing confidence value for all nets in the SPEF file. ■ ROUTING_CONFIDENCE_ENTRY *positive_integer* character_string: Specifies one or more characters that represent additional routing confidence values, which you can assign to nets in the SPEF file.
flow_type (continued)	■ NAME_SCOPE LOCAL|FLAT: Specifies whether paths in the SPEF file are LOCAL (relative to the current SPEF file) or FLAT (relative to the top level of your circuit design). ■ SLEW_THRESHOLDS low high: Specifies low and high default input slew thresholds for your circuit design as a percentage of the voltage level for the input pin. ■ PIN_CAP NONE|INPUT_OUTPUT|INPUT_ONLY: Specifies the type of pin capacitance to include when calculating the total capacitance for all nets in the SPEF file, either no capacitance, all input and output capacitances, or only input capacitances.
divider	Character used to divide levels of hierarchy in a circuit path name. Must be one of the following characters: . / : | For example, X1/X2 means that X2 is a subcircuit of the X1 circuit.

Table 6 SPEF Parameters (Continued)

Parameter	Definition
delimiter	Character used to separate the name of an instance and a pin in a concatenated instance pin name. Must be one of these characters: . / : |
bus_prefix bus_suffix	Delimiter characters that precede and follow a bus bit or an arrayed instance number. If these characters are not matching pairs, HSPICE advanced analog analyses reports an error. Valid bus delimiter prefix and suffix character pairs are brackets “[ ]”, braces “{ }”, parentheses “( )”, or angle brackets “< >”>
time_unit	A positive number. For example, 10 PS means use time units of 10 picoseconds. 5 NS means use time units of 5 nanoseconds.
capacitance_unit	A positive number. For example, 10 PF means capacitance units of 10 picofarads. 5 FF means use capacitance units of 5 femtoseconds.
resistance_unit	Positive number. For example, 10 OHM sets resistance units to 10 ohms. 5 KOHM sets resistance units to 5 kilo ohms.
inductance_unit	A positive number. For example, 10 HENRY means use inductance units of 10 henries. 5 MH means use inductance units of 5 millihenries. 2 UH means use inductance units of 2 micro-henries.
name_index	Name used throughout a SPEF file. To reduce file space, you can map other names to this name.
name_id|bit|path|name| physical_ref	A name identifier, bit, path, name, or physical reference to map to the name_index.
logical_power_net	Logical path (or logical path index) to a power net.
physical_power_net	Physical path (or physical path index) to a power net. You can specify multiple *logical_power_net physical_power_net* pairs.
ground_net	Name of a net to use as a ground net. You can specify multiple ground net names.
logical_port	Logical name of an input, output, or bidirectional port.
coordinate	Geometric location of a logical or physical port.
par_value	Either a single float value, or a triplet in float:float:float form.
rising_slew	Rising slew of the waveform for the port. T_UNIT defines the time unit for the waveform.
falling_slew	Rising slew of the waveform for the port. T_UNIT defines the time unit for the waveform.
low_threshold	Low voltage threshold as a percentage of the port’s input voltage. Can bed one float value or a triplet in float:float:float form.

Table 6 SPEF Parameters (Continued)

Parameter	Definition
high_threshold	High voltage threshold as a percentage of the input voltage for the port. Either a single *float* value or a triplet in *float:float:float* form.
cell_type	Type of cell that drives the port. If you do not know the cell type, use the reserved word UNKNOWN_DRIVER as the cell type.
physical_port	Physical name of an input, output, or bidirectional port.
logical_instance	Logical name of a subcircuit in your *design_name* circuit design. You can specify more than one logical_instance. Whenever you specify a logical instance name, you must set NAME_SCOPE to FLAT. If you connect a logical net to a physical port, HSPICE advanced analog analyses reports an error.
physical_instance	Physical name of a subcircuit in your *design_name* circuit design. You can specify more than one physical_instance. Whenever you specify a physical instance name, you must set NAME_SCOPE to FLAT. If you connect a physical net to a logical port, HSPICE advanced analog analyses reports an error.
routing_confidence	One of the following positive integers: ■ 10: Statistical wire load model. ■ 20: Physical wire load model. ■ 30: Physical partitions with locations, no cell placement. ■ 40: Estimated cell placement with Steiner tree-based route. ■ 50: Estimated cell placement with global route. ■ 60: Final cell placement with Steiner route. ■ 70: Final cell placement with global route. ■ 80: Final cell placement, final route, 2d extraction. ■ 90: Final cell placement, final route, 2.5d extraction. ■ 100: Final cell placement, final route, 3d extraction.
logical_pin	Logical name of a pin.
physical_node	Physical name of a node.
net_name	Name of a net in a circuit or subcircuit.
cap_id	Unique identifier for capacitance between two specific nodes.
res_id	Unique identifier for resistance between two specific nodes.
induc_id	Unique identifier for inductance between two specific nodes.
node1	First of two nodes, between which you are specifying a capacitance, resistance, or inductance value.

Table 6 SPEF Parameters (Continued)

Parameter	**Definition**
node2	Second of two nodes, between which you are specifying a capacitance, resistance, or inductance value. For a capacitance value, if you do not specify a second node name, HSPICE advanced analog analyses assumes that the second node is ground.
capacitance	Specifies the capacitance value assigned to a *cap_id* identifier. *capacitance_unit* defines the units of capacitance. For example, if you set capacitance to 5 and capacitance_unit to 10 PF, then the actual capacitance value is 50 picoFarads.
resistance	Specifies the resistance value assigned to a *res_id* identifier. *resistance_unit* defines the units of resistance. For example, if you set *resistance* to 5 and *resistance_unit* to 5 KOHM, then the actual resistance value is 25 kilo ohms.
inductance	Specifies the resistance value assigned to an *induc_id* identifier. *inductance_unit* defines the units of inductance. For example, if you set *inductance* to 6 and *inductance_unit* to 2 UH, then the actual inductance value is 12 microhenries.

SPEF File Example

```
*SPEF "IEEE 1481-1998"
*DESIGN   "My_design"
*DATE   "11:26:34 Friday June 28, 2002"
*VENDOR   "Synopsys, Inc."
*PROGRAM   "Star-RCXT"
*VERSION   "2002.2."
*DESIGN_FLOW   "EXTERNAL_LOADS" "EXTERNAL_SLEWS" "MISSING_NETS"
*DIVIDER   /
*DELIMITER   :
*BUS_DELIMITER   [ ]
*T_UNIT   1 NS
*C_UNIT   1 PF
*R_UNIT   1 OHM
*L_UNIT   1 HENRY

*POWER_NETS   VDD
*GND_NETS   VSS

*PORTS
CONTROL O *L 30 *S 0 0
FARLOAD O *L 30 *S 0 0
INVX1FNTC_IN I *L 30 *S 5 5
NEARLOAD O *L 30 *S 0 0
TREE O *L 30 *S 0 0
```

If you use triplet format, the above section would look like this:

```
*PORTS
CONTROL O *L 30:30:30 *S 0:0:0 0:0:0
FARLOAD O *L 30:30:30 *S 0:0:0 0:0:0
INVX1FNTC_IN I *L 30:30:30 *S 5:5:5 5:5:5
NEARLOAD O *L 30:30:30 *S 0:0:0 0:0:0
TREE O *L 30:30:30 *S 0:0:0 0:0:0
```

This triplet formatting principle applies to the rest of this example.

```
*D_NET INVX1FNTC_IN 0.033
*CONN
*P INVX1FNTC_IN I
*I FL_1281:A *L 0.033
*END
*D_NET INVX1FNTC 2.033341

*CONN
*I FL_1281:X O *L 0.0
*I I1184:A I *L 0.343
*I FL_1000:A I *L 0.343
*I NL_1000:A I *L 0.343
*I TR_1000:A I *L 0.343

*CAP
216 FL_1000:A 0.346393
217 I1184:A 0.344053
218 INVX1FNTC_IN 0
219 INVX1FNTC_IN:10 0.154198
220 INVX1FNTC_IN:11 0.117827
221 INVX1FNTC_IN:12 0.463063
222 INVX1FNTC_IN:13 0.0384381
223 INVX1FNTC_IN:14 0.00246845
224 INVX1FNTC_IN:15 0.00350198
225 INVX1FNTC_IN:16 0.00226712
226 INVX1FNTC_IN:17 0.0426184
227 INVX1FNTC_IN:18 0.0209701
228 INVX1FNTC_IN:2 0.0699292
229 INVX1FNTC_IN:20 0.019987
230 INVX1FNTC_IN:21 0.0110279
231 INVX1FNTC_IN:24 0.0192603
232 INVX1FNTC_IN:25 0.0141824
233 INVX1FNTC_IN:3 0.0520437
234 INVX1FNTC_IN:4 0.0527105
235 INVX1FNTC_IN:5 0.1184749
236 INVX1FNTC_IN:6 0.0468458
237 INVX1FNTC_IN:7 0.0391578
238 INVX1FNTC_IN:8 0.0113856
239 INVX1FNTC_IN:9 0.0142528
240 NL_1000:A 0.344804
```

```
241 TR_000:A 0.34506

*RES
152 INVX1FNTC_IN INVX1FNTC_IN:18 8.39117
153 INVX1FNTC_IN INVX1FNTC_IN:5 25.1397
154 INVX1FNTC_IN:11 INVX1FNTC_IN:20 4.59517
155 INVX1FNTC_IN:12 INVX1FNTC_IN:13 3.688
156 INVX1FNTC_IN:13 INVX1FNTC_IN:17 25.102
157 INVX1FNTC_IN:14 INVX1FNTC_IN:16 0.0856444
158 INVX1FNTC_IN:14 NL_1000:A 0.804
159 INVX1FNTC_IN:15 INVX1FNTC_IN:16 1.73764
160 INVX1FNTC_IN:15 INVX1FNTC_IN:24 0.307175
161 INVX1FNTC_IN:17 INVX1FNTC_IN:25 5.65517
162 INVX1FNTC_IN:18 FL_1000:A 1/36317
163 INVX1FNTC_IN:2 INVX1FNTC_IN:4 6.95371
164 INVX1FNTC_IN:2 INVX1FNTC_IN:5 50.9942
165 INVX1FNTC_IN: INVX1FNTC_IN:21 4.71035
166 INVX1FNTC_IN: I1184:A 0.403175
167 INVX1FNTC_IN: TR_1000:A 0.923175
168 INVX1FNTC_IN: INVX1FNTC_IN:12 31.7256
169 INVX1FNTC_IN: INVX1FNTC_IN:4 11.9254
170 INVX1FNTC_IN: INVX1FNTC_IN:7 25.3618
171 INVX1FNTC_IN: INVX1FNTC_IN:6 23.3057
172 INVX1FNTC_IN: INVX1FNTC_IN:24 8.64717
173 INVX1FNTC_IN: INVX1FNTC_IN:8 7.46529
174 INVX1FNTC_IN: INVX1FNTC_IN:10 2.04729
175 INVX1FNTC_IN: INVX1FNTC_IN:10 10.8533
176 INVX1FNTC_IN: INVX1FNTC_IN:11 1.05164

*END

*D_NET NE_794 1.98538

*CONN
*I NL_1039:X O *L 0 *D INVX
*I NL_2039:A I *L 0.343
*I NL_1040:A I *L 0.343

*CAP
3387 NE_794 0
3388 NE_794:1 0.0792492
3389 NE_794:10 0.0789158
3390 NE_794:11 0.0789991
3391 NE_794:12 0.0789991
3392 NE_794:13 0.0792992
3393 NE_794:14 0.00093352
3394 NE_794:15 0.00063346
3395 NE_794:16 0.0792992
```

```
3396 NE_794:17 0.80116
3397 NE_794:18 0.80116
3398 NE_794:19 0.00125452
3399 NE_794:2 0.0789158
3400 NE_794:20 0.00336991
3401 NE_794:21 0.00668512
3402 NE_794:23 0.00294932
3403 NE_794:25 0.00259882
3404 NE_794:26 0.00184653
3405 NE_794:3 0.0789158
3406 NE_794:4 0.0796826
3407 NE_794:5 0.0796826
3408 NE_794:6 0.0789991
3409 NE_794:7 0.0789991
3410 NE_794:8 0.0793992
3411 NE_794:9 0.0789158
3412 NL_1039:X 0.00871972
3413 NL_1040:A 0.344453
3414 NL_2039:A 0.343427

*RES
2879 NE_794:1 NE_794:13 66.1953
2880 NE_794:1 NE_794:2 0.311289
2881 NE_794:11 NE_794:12 0.311289
2882 NE_794:13 NE_794:14 0.353289
2883 NE_794:14 NE_794:19 0.365644
2884 NE_794:15 NE_794:16 0.227289
2885 NE_794:15 NE_794:20 0.239644
2886 NE_794:17 NE_794:18 0.14
2887 NE_794:19 NE_794:21 0.0511746
2888 NE_794:2 NE_794:9 65.9153
2889 NE_794:20 NE_794:23 1.15117
2890 NE_794:21 NL_1039:X 3.01917
2891 NE_794:25 NE_794:26 0.166349
2892 NE_794:26 NL_1040:A 0.651175
2893 NE_794:3 NE_794:10 65.9153
2894 NE_794:3 NE_794:4 0.311289
2895 NE_794:4 NE_794:17 66.5437
2896 NE_794:5 NE_794:18 66.5437
2897 NE_794:5 NE_794:6 0.311289
2898 NE_794:6 NE_794:11 65.98853
2899 NE_794:7 NE_794:12 65.9853
2900 NE_794:7 NE_794:8 0.311289
2901 NE_794:8 NE_794:16 66.3213
2902 NE_794:9 NE_794:10 0.311289
2903 NL_1039:X NE_794:25 1.00317
2904 NL_2039:A NE_794:23 0.171175
```

```
*END
```

Linear Acceleration

Linear acceleration, by using the SIM_LA option, accelerates the simulation of circuits that include large linear RC networks. To achieve this acceleration, HSPICE advanced analog analyses reduces all matrices that represent RC networks. The result is a smaller matrix that maintains the original port behavior, yet achieves significant savings in memory and computation. Thus, the SIM_LA option is ideal for circuits with large numbers of resistors and capacitors, such as clock trees, power lines, or substrate networks.

In general, the RC elements are separated into their own network. The nodes shared by both main circuit elements (including .PRINT, .PROBE, and .MEASURE statements), and RC elements. are the port nodes of the RC network,. All other RC nodes are internal nodes. The currents flowing into the port nodes are a frequency-dependent function of the voltages at those nodes.

The multiport admittance of a network represents this relationship.

- The SIM_LA option formulates matrices to represent multiport admittance.
- Then, to eliminate as many internal nodes as possible, it reduces the size of these matrices, while preserving the admittance, otherwise known as port node behavior.
- The amount of reduction depends on the *f*0 upper frequency, the threshold frequency where SIM_LA preserves the admittance. This is shown graphically in Figure 32.

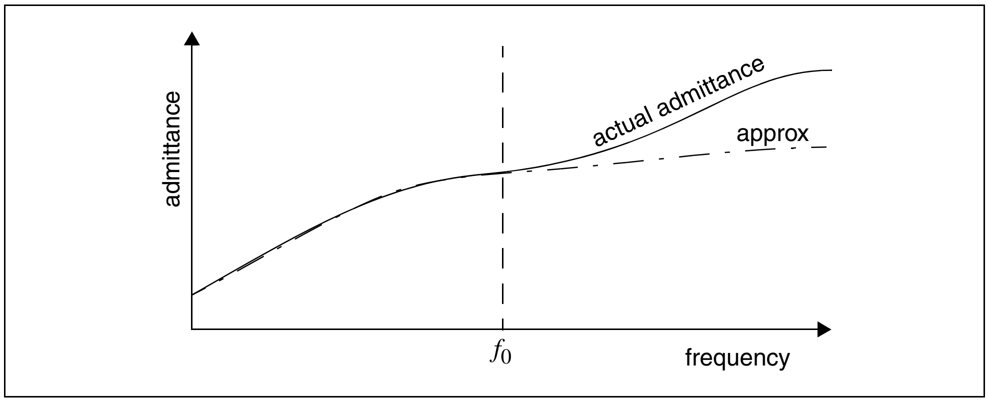


Figure 32 Multiport Admittance vs. Frequency

The `SIM_LA` option is very effective for post-layout simulation, because of the volume of parasitics. For frequencies below *f*0, the *approx* signal matches that of the original admittance. Above f_0, the two waveforms diverge, but presumably the higher frequencies are not of interest. The lower the *f*0 frequency, the greater the amount of reduction.

For the syntax and description of this control option, see .OPTION SIM_LA in the *HSPICE Reference Manual: Commands and Control Options*.

You can choose one of two algorithms, explained in the following sections:

- PACT Algorithm
- PI Algorithm

PACT Algorithm

The `PACT` (Pole Analysis via Congruence Transforms) algorithm reduces the RC networks in a well-conditioned manner, while preserving network stability.

- The transform preserves the first two moments of admittance at DC (slope and offset), so that DC behavior is correct (see Figure 33).
- 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 the most accurate of the two algorithms, and is the default.

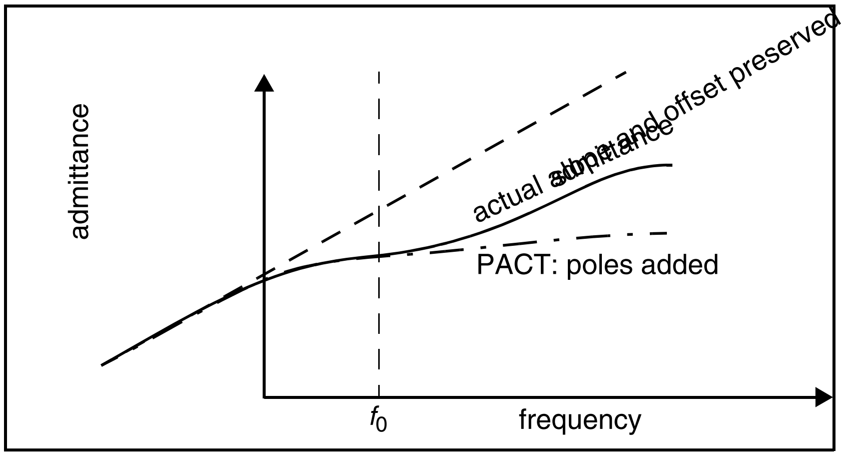


Figure 33 PACT Algorithm

PI Algorithm

This algorithm creates a *pi* model of the RC network.

- For a two-port, the *pi* model reduced network consists of:
  - a resistor connecting the two ports, and
  - a capacitor connecting each port to ground

    The result resembles the Greek letter pi.
- For a general multiport, `SIM_LA` preserves the DC admittance between the ports, and the total capacitance that connects the ports to ground. All floating capacitances are lumped to ground.

Linear Acceleration Control Options Summary

In addition to `.OPTION 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 7 on page 233 contains a summary of these control options. For the syntax and descriptions of these options, see HSPICE Netlist Simulation Control Options in the *HSPICE Reference Manual: Commands and Control Options*.

Table 7 PACT Options

Syntax	Description
.OPTION SIM_LA=PACT | PI	Activates linear matrix reduction and selects between two methods.
.OPTION SIM_LA_FREQ=<value>	Upper frequency where you need accuracy preserved. *value* is the upper frequency for which the PACT algorithm preserves accuracy. If *value* is 0, PACT drops all capacitors, because only DC is of interest. The maximum frequency required for accurate reduction depends on both the technology of the circuit and the time scale of interest. In general, the faster the circuit, the higher the maximum frequency. The default is 1GHz.
.OPTION SIM_LA_MAXR=<value>	Maximum resistance for linear matrix reduction. *value* is the maximum resistance preserved in the reduction. SIM_LA assumes that any resistor greater than *value* has an infinite resistance, and drops the resistor after reduction finishes. The default is 1e15 ohms.
.OPTION SIM_LA_MINC=<value>	Minimum capacitance for linear matrix reduction. *value* is the minimum capacitance preserved in the reduction. After reduction completes, SIM_LA lumps any capacitor smaller than *value* to ground. The default is 1e-16 farads.
.OPTION SIM_LA_MINMODE= ON|OFF	Reduces the number of nodes instead of the number of elements.
.OPTION SIM_LA_TIME=<value>	Minimum time for which accuracy must be preserved. *value* is the minimum switching time for which the PACT algorithm preserves accuracy. HSPICE advanced analog analyses does not accurately represent waveforms that occur more rapidly than this time. SIM_LA_TIME is simply the dual of SIM_LA_FREQ. The default is equivalent to setting LA_FREQ=1 GHz. The default is 1ns.
.OPTION SIM_LA_TOL=<value>	Error tolerance for the PACT algorithm. *value* is the error tolerance for the PACT algorithm, is between 0.0 and 1.0. The default is 0.05.

Example

In this example, the circuit has a typical risetime of 1ns. Set the maximum frequency to 1 GHz, or set the minimum switching time to 1ns.

```
.OPTION SIM_LA_FREQ = 1GHz
-or-
.OPTION SIM_LA_TIME = 1ns
```

However, if spikes occur in 0.1ns, HSPICE will not accurately simulate them. To capture the behavior of the spikes, use:

```
.OPTION SIM_LA_FREQ = 10GHz
-or-
.OPTION SIM_LA_TIME = 0.1ns
```

Note: Higher frequencies (smaller times) increase accuracy, but only up to the minimum time step used in HSPICE.

Part 3: Noise Analyses

Note: For `.AC`-related `.NOISE` analysis, see AC Small-Signal and Noise Analysis in the *HSPICE User Guide: Basic Simulation and Analysis*.

This Part contains the following chapters/topics:

- Chapter 9, Transient Noise Analysis
- Chapter 10, Simulation of Random Noise

9

Transient Noise Analysis

Describes the HSPICE solutions to perform transient noise analysis and compute noise statistics and their variation over time for circuits driven with non-periodic waveforms.

Transient noise analysis shows the effect of noise on the signal magnitude. It is also useful to see how noise affects the timing of the signal. From the transient noise analysis results, you can measure jitter. The two jitter measurements are time interval error (TIE) and autocorrelation function. TIE measures the time-shift behavior relative to a reference signal. The autocorrelation function tracks the relative time-shift behavior of the signal.

This chapter describes two approaches:

- Monte Carlo (default), where HSPICE uses random signal sources to predict the statistical characteristics of the circuit performance due to device noise. (See Monte Carlo Noise Analysis on page 241.)
- Stochastic Differential Equation (SDE), for advanced users, which makes a direct prediction of the actual statistics of the output waveforms. (See Stochastic Differential Equation (SDE) Analysis on page 247.)

HSPICE includes several different algorithms for understanding circuit behavior based on noise generated internally by electronic devices and thermal noise. PHASENOISE analysis computes the effects noisy elements have on the output spectrum of oscillators. HBNOISE and SNNOISE analyses compute the small-signal variations that noise can create under large-signal steady-state operating conditions (see Multitone Harmonic Balance Noise (.HBNOISE) or Shooting Newton Noise Analysis (.SNNOISE). Periodic time-domain noise (PTDNOISE) analysis computes the noise statistics of a periodic signal, and how those statistics vary with time over the period of the steady-state signal (see Periodic Time-Dependent Noise Analysis (.PTDNOISE). What makes Transient Noise Analysis unique compared to these other approaches is that its noise analysis is a complement to the fully nonlinear dynamic time-domain `.TRAN` simulation of HSPICE.

For additional information and graphical illustration of the HSPICE transient noise solution, see the webinar link: **Jitter Analysis Using HSPICE Transient Noise Techniques** at http://www.hspice.com and https://www.synopsys.com/news/pubs/snug/sanjose11/ma6_tutorial_HSPICE_trannoise.pdf (requires Synopsys SolvNet user ID and password).

Note: For `.AC`-related `.NOISE` analysis, see AC Small-Signal and Noise Analysis in the *HSPICE User Guide: Basic Simulation and Analysis*.

The following sections discuss in these topics:

- Overview of HSPICE Transient Noise Analysis
- Monte Carlo Noise Analysis
- Stochastic Differential Equation (SDE) Analysis
- Jitter Measurements from .TRANNOISE Results
- Error Handling, Error Recovery, Status Reporting
- References

Overview of HSPICE Transient Noise Analysis

A variety of noise measurements are desirable from circuit simulation. The traditional SPICE `.noise` analysis provides a measurement of the RMS noise voltage at an output node as a function of frequency. This RMS value is similar to a measurement of the standard deviation of an equivalent Gaussian distribution of noise present at the output node of interest due to the contributing random noise sources within the circuit. The `.noise` analysis is a small-signal analysis That gives an output noise (onoise) value over the `.ac` frequency range.

More advanced examples of noise measurements include Phase Noise and Timing Jitter. Timing Jitter, in particular, is a measurement of a clock or oscillator's random noise over a time interval. Timing Jitter represents the standard deviation (or variance) of the timing uncertainty (i.e. the random drift of the clock edges) as a function of time. It is therefore a time-domain noise measurement that HSPICE typically evaluates at each cycle or half-cycle. For some clocks, oscillators, and PLLs, you can separate this measurement into time-independent and time-dependent contributions, and written as

Equation 47

$$\sigma^2(\tau) = \frac{2}{\omega_0^2}[R_\phi(0) - R_\phi(\tau)]$$

where $\sigma^2(\tau)$ represents the time-dependent variance, τ is the time advance, and $R_\phi(\tau)$ is the autocorrelation function that relates to the power spectrum of phase variations as

Equation 48

$$R_\phi(\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_\phi(\omega)e^{j\omega\tau}d\omega$$

These special relationships allow the computation of timing jitter from the results of `.phasenoise` analysis, since we can assume $L(f) \cong S_\phi(f)$, and therefore derive the time-varying noise from the frequency-domain phase noise simulation solution.

Other measurements of time-dependent or time-varying noise are also desirable for circuits other than clocks and oscillators, and for situations other than steady-state operation. The purpose of such measurements is usually similar: derive useful information on the time-varying statistical behavior of the circuit due to its internal noise sources.

Transient Noise Analysis is a typical `.TRAN` simulation, but with all random noise sources within the circuit activated as contributing signal sources. The Monte Carlo Noise analysis is a transient noise simulation approach that uses uncorrelated random signal sources for device noise in such a way that you can seed all noise signals uniquely. In addition, you can repeatedly, from run-to-run, predict the statistical characteristics of circuit performance due to device noise. You can typically examine the resulting outputs by using histogram plots to measure the statistical behavior.

Modeling Frequency-Dependent Noise Sources

Transient noise techniques require that you model noise sources in the time domain. These techniques model device noise sources in terms of standard white Gaussian noise processes. You write the standard white Gaussian noise process as $\zeta(t)$.

You calculate white noise source models in terms of intensity functions. For example, if Equation 49 gives the (frequency dependent) noise from a time varying conductance

Equation 49

$$\overline{i_n^2} = 4kTG(t)\Delta f$$

...then you can model in terms of intensity as the random time-domain current given by

Equation 50

$$j_n = \sqrt{2kTG(t)}\zeta(t)$$

In the case of flicker noise, it is necessary to create the 1/f power spectrum of flicker noise sources by filtering a $\zeta(t)$ process. You can accomplished this with the following rational function transfer relationship:

Equation 51

$$H(s) = \prod_{i=1}^{N}\frac{(s+\omega_{zi})}{(s+\omega_{pi})}$$

Since you can model all flicker sources over the same frequency range (i.e., bandwidth), you can use this same transfer function for all sources. Tests show that the above fit is very accurate using one pole/zero per octave. This modeling technique shows reasonable fits using one pole/zero for every two octaves, or even with one pole/zero per decade. This fitting algorithm is therefore frequency range-specific, which is why you use the parameters FMAX and FMIN for specifying frequency ranges for Transient Noise Analysis similar to the modeling of the frequency-dependent S-element.

The Trannoise analysis includes a time-domain noise source for all noise contributions within all devices. So, for example, if the transistor model in use includes induced gate noise, so does `.TRANNOISE`. For example, if you use a BSIM4 model, and the model parameter set includes values for Induced Gate Noise, then `.TRANNOISE` includes this noise in the simulation.

Monte Carlo Noise Analysis

The equations for Monte Carlo transient noise analysis define the Monte Carlo approach:

The time-domain system of equations for transient analysis drives the Modified Nodal Analysis (MNA) system described by the following vector state equation:

Equation 52

$$f(\dot{x}, x, t) = 0$$

Consider this the noiseless system that HSPICE solves for during a normal transient analysis. You can consider Transient noise analysis a similar analysis where we now inject noise from all device noise models to give the modified equation:

Equation 53

$$f(\dot{x}, x, t) = -j_n(t)$$

This system reflects the added noise sources and HSPICE can solved for it in the same manner as transient analysis. However, if we consider each noise source related to a white Gaussian noise function, we must create an ***ensemble*** of waveforms for our unknown *x(t)* vector to predict the output statistics. This type of simulation involves generating multiple uncorrelated noise source waveforms for all noise sources, and then running multiple simulations in a Monte Carlo fashion. This approach is the Monte-Carlo Noise Simulation. This method cannot directly measure the statistics of output signals due to input noise (as with `.NOISE`), but instead models noise sources as independent time-domain stimuli. Generating statistical information in this approach requires running a plurality of simulations over a variety of random noise-source sequences (to create an ensemble of output waveforms) and then analyzing the statistics of the ensemble using histograms of other plots. When the system behavior is ***ergodic***, it is possible to run a single very long duration simulation in order to capture the statistical variations of the output signal over time (as with an eye-diagram). Monte-Carlo modeling of the noise sources uses a sum of sinusoids with random phases [2], or random number generators with the appropriate statistical distributions[3]. An advantage of the Monte-Carlo approach is its ability to capture very nonlinear noise behaviors. This is useful, for example, when you know that the responses of circuits with noise have non-Gaussian variations about their noiseless simulations.

Monte Carlo Noise Techniques Available in HSPICE

HSPICE provides these Monte Carlo techniques for transient noise analysis:

1. Single-run Monte Carlo: (default) A single transient analysis that includes time-varying noise contributions to all output waveforms. With ergodic systems, the statistics of output variables are observable over time.
2. Multi-run Monte Carlo: Multiple transient analyses, each including time-varying noise, with unique seeding from run to run, form an ensemble of simulations. Output waveform statistics are observable across the ensemble at specific time points.

Single-run Monte Carlo features are:

- Simulation includes statistically accurate noise source contributions from all devices.
- Adjustable bandwidth for noise-source frequency responses.
- Fastest approach possible; based on single `.TRAN` analysis.
- Effective for ergodic simulations (viewing statistical variations over time).
- Use the following process for Single Monte Carlo Trannoise analysis
  - Run as typical `.TRAN` analysis.
  - Post process waveforms to measure resulting noise effects.
  - Use `FMIN` to set low-frequency flicker noise limits.
  - Use `FMAX` to set maximum noise-source waveform bandwidth and ensure Nyquist sampling.
  - Vary seed values to re-run simulations with uniquely different noise waveforms.
- The multi-run Monte Carlo approach is useful for characterizing statistics at specific time events.
- Multi-run Monte Carlo provides data ensembles for histogram generation.

Setting up a Monte Carlo .TRANNOISE Analysis

The transient noise analysis requires an accompanying `.tran` analysis which determines the time-sampling, matrix solutions, and deterministic output waveforms. The `.TRANNOISE` command activates transient noise and computes the additional noise variables. This is consistent with how `.NOISE` computes additional noise outputs when you add it to an `.AC` analysis.

The following sections discuss these topics:

- Monte Carlo Input Syntax
- Monte Carlo Output Data
- Controlling Noise in Subcircuits
- Monte Carlo Examples

Monte Carlo Input Syntax

Monte Carlo Single Sample Approach

```
.TRANNOISE output [METHOD=MC] [SEED=val] [START=val]
+ [FMIN=val] [FMAX=val|auto] [SCALE=val]
+ [AUTOCORRELATION=0|1|2|off|on]
+ [PHASENOISE=0|1|2]
+ [REF=srcName]
```

Monte Carlo Multi-Sample Approaches

```
.TRANNOISE output [METHOD=MC] SAMPLES=val [SEED=val]
+ [START=val] [FMIN=val][FMAX=val|auto] [SCALE=val]
+ [AUTOCORRELATION=0|1|2|off|on]
+ [PHASENOISE=0|1|2]
+ [REF=srcName]
```

or

```
.TRANNOISE output [METHOD=MC] [SAMPLES=List(...)]
+ [START=val][FMIN=val][FMAX=val|auto] [SCALE=val]
+ [AUTOCORRELATION=0|1|2|off|on]
```

Keyword	Description
output	(Required) Output node, pair of nodes, or 2-terminal element. HSPICE references Noise calculations to this node (or node pair). Specify a pair of nodes as V(n+,n-). If you specify only one node, V(n+), then HSPICE reads the second node as ground. If you specify a 2-terminal element, HSPICE treats the noise voltage across this element as the output.
METHOD=MC|SDE	Specifies Monte Carlo or SDE transient noise analysis method. The default, or, if METHOD is not specified, is the single-sample Monte Carlo method. Specifying METHOD=SDE is required to select the transient noise analysis SDE method. METHOD=MC | SDE is position independent.

Keyword	Description
SAMPLES=*val*	Specifies the number of Monte Carlo samples to use for the analysis. The default, or if you do not specify SAMPLES, is 1, the single-sample Monte Carlo method. For the multi-sample Monte Carlo method, you must specify SAMPLES as greater than 1.
SEED=*val*	(Optional) Specifies the beginning simulation sample. Default=2, if you do not specify a value for SEED. If you set SEED=1 HSPICE performs a noiseless simulation.
SAMPLES= *List(...)*	Where List can take the form: ■ LIST (num1, num2, num3, …) List of sample SEED values to execute. ■ LIST(<*num1*:*num2*><*num3*><*num4*:*num5*>) List of sample SEED value ranges; for example: from num1 to num2, sample num3, and samples from num4 to num5 are executed.
START=*val*	(Default=0) Start time during transient analysis when noise sources are activated..
FMIN	(Optional) Sets base frequency for modeling frequency-dependent noise sources. Sets low-frequency flicker noise limit for contributing noise sources. (Default: 1/TSTOP); See Note below.
FMAX	(Optional) Maximum frequency used for modeling frequency-dependent noise sources. Sets maximum noise-source waveform bandwidth and ensures Nyquist sampling. Default: 1/(2*TSTEP). When `FMAX=auto`, HSPICE picks this frequency value automatically. See Note below.
SCALE	Scale factor that you can apply to uniformly amplify the intensity of all device noise sources to exaggerate their contributions. Default: 1.0
AUTOCORRELATION	(Optional for MC approaches) Used to enable the autocorrelation function calculation at the specified output. ■ AUTOCORRELATION=0 (OFF) - (default) Does not calculate autocorrelation function. ■ AUTOCORRELATION=1 (ON) - Calculates autocorrelation function at the specified output. ■ AUTOCORRELATION=2 - Calculates autocorrelation function iat the specified output, with normalization applied over the simulation interval.
PHASENOISE=0|1|2	■ PHASENOISE=0: (default) Phase noise calculations are disabled. ■ PHASENOISE=1: Uses Delay-Line measurement approach to compute phase noise. ■ PHASENOISE=2: Uses Phase Detector method for phase noise calculations
REF=*srcName*	Where *srcName* can be either: ■ PULSE voltage or current source. ■ SIN voltage or current source. The rises edges of the SIN or PULSE source are used to establish the phase reference for phase noise calculations.

Note: FMAX has a dramatic effect on TRANNOISE, since it controls the amount of energy each noise source can emit. Therefore, huge values of FMAX (like 100G) can result in huge instantaneous noise levels. FMIN sets the low frequency limit for flicker noise, and therefore controls the energy in flicker noise sources. You can expect some significant differences with FMAX and FMIN changes: Noise power increases linearly with FMAX; flicker noise power can scale as 1/FMIN.

FMAX is a critical parameter for controlling TRANNOISE performance. This keyword sets the noise bandwidth of noise sources which also controls the maximum time step taken. A large FMAX can cause very slow simulation performance. The default value of FMAX is the inverse of the `.TRAN` tstep value. In some circumstances very small tstep values can causes poor performance. A good setting for FMAX is ~2X the fastest clock frequency in the circuit under test.

Monte Carlo Output Data

The `.TRANNOISE` analysis outputs raw data to a `*.trpn0` and `*.printtr0` files. Formatting of data output is consistent with that used for `*.pn0` and `*.snpn0` data files. These output files organize data according to the sample number in the Monte Carlo index.

The first sample (index=1) creates a noise-free simulation, i.e., disables all noise sources for this simulation.With index=2, all subsequent simulations use unique random number seeding to create unique simulation results due to noise.

Controlling Noise in Subcircuits

Substantial performance improvements can be obtained by removing noise from non-critical circuit blocks. It can also be useful to disable various noise contributors to isolate and identify the dominant circuit noise effects. Transient noise analysis allows you to ignore noise contributions at the subcircuit level.

To ignore the noise contribution, add the `NOISE=0` parameter to any subcircuit instance specified by an X-element. `NOISE=0` is treated as a hierarchical parameter, causing noise to be disabled down through the subcircuit's hierarchy.

Monte Carlo Examples

The following example generates 30 Monte Carlo noise simulations that starts with a noiseless (index=1) simulation.

```
.TRANNOISE v(out) SAMPLES=30
```

The following example generates 20 Monte Carlo noise simulations that starts with the seed value (i.e., index) of 31 for the first simulation.

```
.TRANNOISE v(out) SAMPLES=20 SEED=31
```

The following example generates a single noise simulation, with seed value of 50, and amplifies all noise sources by a factor of 10.

```
.TRANNOISE v(out) SEED=50 SCALE=10.0
```

The following example generates six Monte Carlo transient noise simulations with seed values of 1, 3, 4, 5, 9 and 10. Normalized autocorrelation is computed for each `v(out)` output.

```
.TRANNOISE v(out) SAMPLES=LIST(1,3:5,9:10) AUTOCORRELATION=2
```

Correlating Noise Results: .TRANNOISE (Monte Carlo) and .NOISE

Comparing noise results between `.TRANNOISE` and `.NOISE` simulations requires some careful setups:

1. Pay attention to the bandwidth (i.e., frequency sweep) that you are using for `.NOISE`. When you run `.TRANNOISE`, the simulation is always bandwidth-limited, and based on the time-stepping (tsteps) and time interval (tstop) of the simulation. You want to reflect the same bandwidth in the `.NOISE` simulation, as well. (See below for the reason.)
2. If possible, test your circuit for natural bandwidth limitations, i.e., the `onoise` output rolls off substantially beyond some frequency. This is true of post-layout circuits that have some capacitance at every node. If your circuit has natural BW limits, then item #1 above is not critical.
3. Run your `.noise` simulation, and monitor the Total Output Noise Voltage in units of Volts. This is the integrated `ONOISE` over the bandwidth of interest. `ONOISE` values are in Volts-per-unit-sqrt(Hertz). To compare with the time domain, you need to know the Hertz. The (integrated) total output noise voltage prints to the `.lis` file.

4. Set up your `.TRANNOISE` simulation such that the noise does not alter the large-signal behavior of the circuit. `.NOISE` is a small-signal simulation. Try to keep `.trannoise` signals small (e.g., no big PULSE sources) so the comparisons are valid. Note that with `.TRANNOISE`, you do not need an input signal, as the noise becomes the signal.
5. With your `.TRANNOISE` command, set FMAX to match the integration bandwidth (freq range) you use for your `.NOISE` analysis. Note that if you want to see low-frequency range (flicker effects), your `.TRAN` command may need a big TSTOP value. There is no need to set the FMIN parameter, since this sets the noise generation. You still need a big TSTOP to observe this generated noise.
6. To compare with a single-run Monte Carlo simulation, create a `.measure` command to measure RMS voltage for the same node(s) you used for `.NOISE` output. If you cannot avoid a nonlinear transient startup for your circuit, you may need a FROM and TO for this `.MEASURE` to look past it. Your `.MEASURE` result matches your Total Output Noise Voltage if the only signal at this node is due to noise.
7. To compare with a multi-run Monte Carlo simulation, `.MEASURE` the voltage at a particular time point of interest. Then, when you run the simulation, the Monte Carlo report gives results you can compare with `.NOISE`. You may need many samples to get stable statistical results for a single time point.

Stochastic Differential Equation (SDE) Analysis

Transient Noise Analysis predicts waveform statistics at particular time points. One method to accomplish this is to run the multi-sample Monte Carlo approach, and then use the ensemble of simulation results to generate histograms at the time points of interest. Histograms that measure vertical distributions reveal voltage noise. Histograms that measure horizontal time-shift distributions reveal jitter. In those cases where it is desirable to have a highly accurate distribution curve, it may be necessary to simulate with a large number of Monte Carlo samples. SDE analysis can provide a more efficient alternative in such cases. Instead of requiring a large ensemble of results, SDE performs special calculations that directly predict the statistics of the output waveforms.

Let the time-domain signal resulting from a regular transient analysis (i.e., with noise ignored) for a specific output node be written as $v_{out}^{s}(t)$.

Let the time-domain signal resulting from an analysis with signal and noise for the same output node be written as $v_{out}^{s+n}(t)$.

We can define the noise voltage component $v_{out}^{s}(t)$ to be:

Equation 54

$$v_{out}^{n}(t) = v_{out}^{s+n}(t) - v_{out}^{s}(t)$$

We can define a variance equal to the expected value of this noise component at a given time to be:

Equation 55

$$\sigma_n^2(t) = \overline{v_{out}^{n}(t) \cdot v_{out}^{n}(t)}$$

If we assume that the noise variations are small, we can create a linear Stochastic Differential Equation (SDE) for the noise contribution vector $x_n(t)$ (on entry in the vector being the output noise $v_{out}^{n}(t)$). This SDE may be formulated in terms of time-varying coefficient matrices that are evaluated for a normal transient analysis, which are functions of the noise-free solution vector $x_s(t)$ and derived from the noise-free transient analysis computations [1]:

Equation 56

$$A(t)x_n + C(t)\dot{x}_n + B(t)v \cong 0$$
$$x_n(0) = x_{n0}$$

We can then create and solve a linear ordinary differential equation (ODE) system for the time-varying noise correlation matrix $K(t) = \overline{x_n(t)x_n(t)^T}$:

Equation 57

$$\dot{K}(t) = E(t)K(t) + F(t)F(t)^T$$

where $E(t)$ and $F(t)$ are derived from $A(t)$, $B(t)$, and $C(t)$, and T indicates the transpose operation.

In general, simultaneously solving for both the deterministic and stochastic differential equations can therefore give us the complete time-dependent output signal waveform vector $x_s(t)$, as well as the complete time-varying noise correlation/covariance matrix $K(t) = \overline{x_n(t)x_n(t)^T}$. The entries in this matrix represent time-dependent variance values $\sigma_n^2(t)$ for output signals $v^n_{out}(t)$. This output can be interpreted and plotted as a time-varying RMS noise voltage waveforms for $v^n{}_{RMS}(t) = \sqrt{\sigma_n^2(t)}$. The results of such a transient + noise analysis include the usual deterministic transient analysis waveforms, including the mean voltage output $v^s_{out}(t)$, and also its (stochastic) time-varying RMS noise component $v^n{}_{RMS}(t)$. In this sense, the SDE analysis method provides typical SPICE output waveforms for circuit unknowns, plus the additional waveform representing the time-varying statistics of the circuit. The `.TRANNOISE` SDE method therefore activates this special analysis and makes this output available to the user.

Stochastic Differential Equation (SDE) Dynamic noise statistics are computed in the form of a time-varying covariance matrix. SDE techniques allow the output of a variance waveform for any output signal. Variance waveforms can be used to construct probability density plots.

The SDE approach gives probability density information, and, like the multi-run Monte Carlo approach, is useful for characterizing statistics at specific time events.

The following topics are discussed in these sections:

- SDE Approach Input Syntax
- SDE Output Data
- SDE Examples

SDE Approach Input Syntax

```
.TRANNOISE output METHOD=SDE
+ [TIME=(all|val)]
+ [FMIN=val] [FMAX=val] [SCALE=val]
```

Keyword	Description
output	(Required) Output node, pair of nodes, or 2-terminal element. Noise calculations are referenced to this node (or node pair). Specify a pair of nodes as V(n+,n-). If you specify only one node, V(n+), then HSPICE advanced analog analyses reads the second node as ground. If you specify a 2-terminal element, the noise voltage across this element is treated as the output.
METHOD | SDE	Specifies SDE transient noise analysis method.
TIME	(Optional) Used to specify additional time points (breakpoints) where time-domain noise should be evaluated in addition to those time points that will be evaluated as part of the normal time-stepping algorithm. Use this parameter to force noise evaluation at important time points of interest (such as rising/falling edges). ■ TIME=all: (default) causes time-domain noise ONOISE values to be computed and available for output at all time points selected by the .TRAN command time-step algorithm. ■ TIME=val: Specifies a single additional time point at which time domain noise is measured. The value can be numeric or a parameter. A .TRAN analysis at this time point will be forced. Note that time-domain noise calculations require an accompanying .TRAN analysis at each time point. The TIME parameter may therefore add transient analysis time-points (breakpoints) as needed while values given outside the range of the .TRAN command constraints are ignored
FMIN	(Optional) Base frequency used for modeling frequency dependent noise sources. Sets low-frequency flicker noise limit for contributing noise sources. (Default: 1/TSTOP) See Note below.
FMAX	(Optional) Maximum frequency used for modeling frequency dependent noise sources. Sets maximum noise source waveform bandwidth and ensures Nyquist sampling. Default: 1/TSTEP; See Note below.
SCALE	Scale factor that can be applied to uniformly amplify the intensity of all device noise sources to exaggerate their contributions. Default: 1.0

SDE Output Data

The `.TRANNOISE` SDE analysis outputs raw data to the `*.tr0` and `*.printtr0` files consistent with transient analysis. The `.PRINT/.PROBE` output syntax supports the following measurements:

```
.print trannoise ONOISE ONOISE(M) VRMS(n1[,n2])
```

```
.probe trannoise ONOISE ONOISE(M) VRMS(n1[,n2])
```

where:

- The `ONOISE` and `ONOISE(M)` outputs are the same. They represent noise voltage or current at the node or branch specified by the `output` keyword. The `ONOISE` represents the RMS noise voltage component (square root of the variance), units in Volts, of the noise at the specified output present in addition to the noise-less transient voltage.

  Note: `ONOISE` is only output when you use SDE method; when using the Monte Carlo method, `ONOISE` is `0`.

- `VRMS`: The output of RMS noise voltages at other nodes (i.e., the output for general nodal noise voltage values).

The actual instantaneous output voltage is the sum of the signal plus noise components:

Equation 58

$$v_{out}^{s+n}(t) = v_{out}^{s}(t) + v_{out}^{n}(t)$$

Where the noise component $v_{out}^{n}(t)$ has an assumed Gaussian distribution (in x) as:

$$\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-x^2}{2\sigma^2}}$$

And the output signal $v_{out}^{s}(t)$ is that resulting from the (deterministic) `.TRAN` analysis. The time-varying RMS noise voltage waveforms (i.e., for `onoise`) are related to the variance at the specified outputs as given by $v_{RMS}^{n}(t) = \sqrt{\sigma_n^2(t)}$ where: $\sigma_n^2(t) = \overline{v_{out}^{n}(t) \cdot v_{out}^{n}(t)}$.

SDE Examples

Example 1. SDE method with maximum frequency of 5GHz.

```
.TRANNOISE v(7) METHOD=SDE FMAX=5g
```

Example 2. Activates SDE noise analysis, and dumps the `ONOISE` output to the `*.tr0` file:

```
.TRANNOISE v(out) METHOD SDE
.PROBE TRANNOISE ONOISE
```

Example 3. Activates SDE noise analysis, placing a lower bound on flicker noise to be 10kHz, and an upper bound on all noise power at 100MHz:

```
.TRANNOISE v(out) METHOD=SDE FMIN=10k FMAX=100MEG
```

Jitter Measurements from .TRANNOISE Results

While transient noise analysis shows the effect of noise on the signal magnitude, it is also useful to see how the noise affects the timing of the signal. From the transient noise analysis results, jitter can be measured. The two jitter measurements are *time interval error* or TIE and *autocorrelation function*. TIE measures the time-shift behavior relative to a reference signal and is best measured using WaveView. The autocorrelation function tracks the relative time-shift behavior of the signal.

The following topics include:

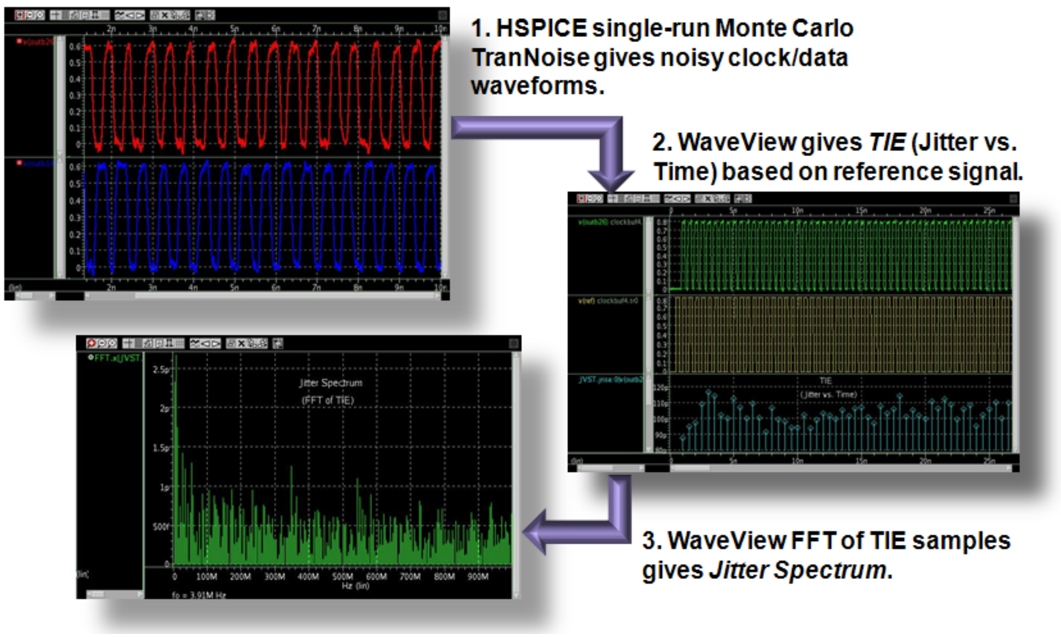
- Output Data Files
- Measure TIE Jitter and Jitter Spectrum Example

Output Data Files

The output data from the autocorrelation function (`.AUTOCORRELATION=1`) calculations is output to the ASCII formatted `*.trnz#` file if you add a `.PROBE` statement to probe ONOISE. For example:

```
.PROBE trannoise onoise
```

Measure TIE Jitter and Jitter Spectrum Example


1. HSPICE single-run Monte Carlo TranNoise gives noisy clock/data waveforms.
2. WaveView gives *TIE* (Jitter vs. Time) based on reference signal.
3. WaveView FFT of TIE samples gives *Jitter Spectrum*.
Jitter Spectrum
(FFT of TIE)
TIE
(Jitter vs. Time)

The following clock buffer example circuit uses a single run Monte Carlo transient noise analysis. Post-processing is performed using WaveView.

Analysis Setup Details

```
* Transient noise example
.option post probe
.option runlvl=5
.param freq=2000MEG period='1/freq'
* Reference Clock
Vsrc  ref  gnd  DC 0 PULSE (0.0 'vdd'
+ '0.975*period' '0.05*period' '0.05*period'
+ '0.45*period' 'period')
* Clock Buffer Circuit
* Analysis setup
.tran '0.05*period' '520*period'
.trannoise v(outb26) FMAX=50G SCALE=10
.probe tran v(ref) v(outa) v(outb14) v(outb26)
.end
```

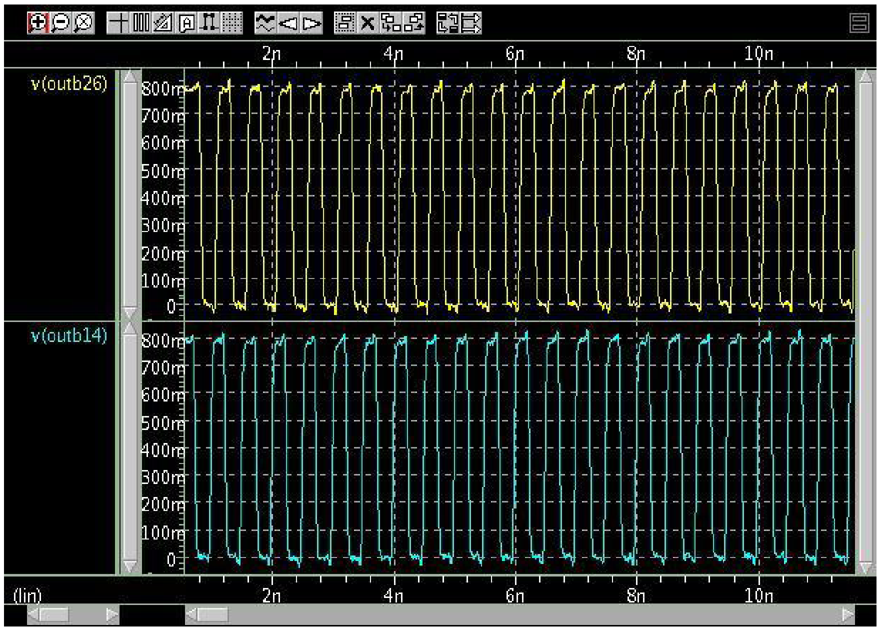


Figure 34 Single-run Monte Carlo Transient Noise gives noisy clock/data waveforms

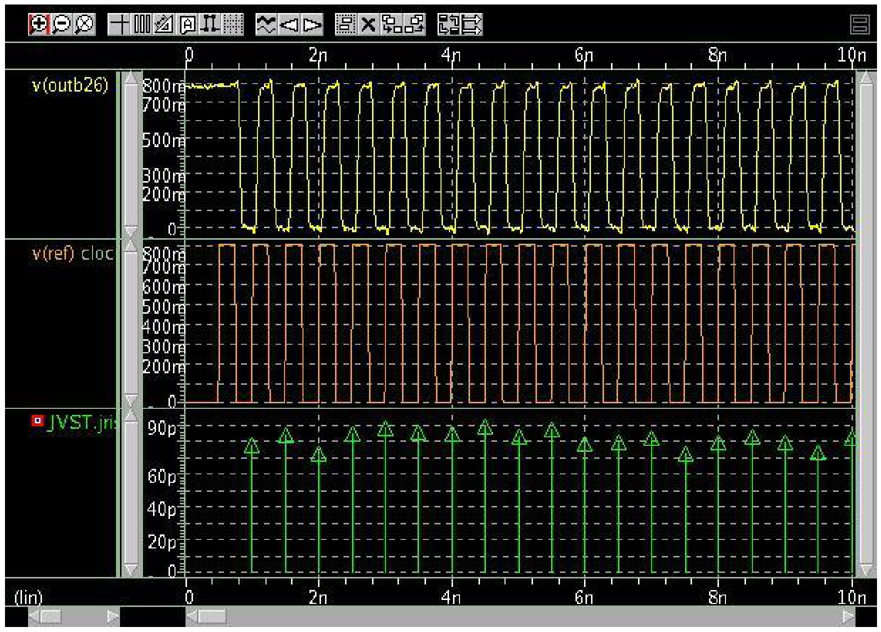


Figure 35 WaveView: Jitter vs. Time measurement to get Timing Interval Error (TIE) Jitter vs. Time. Measurement based on the reference signal

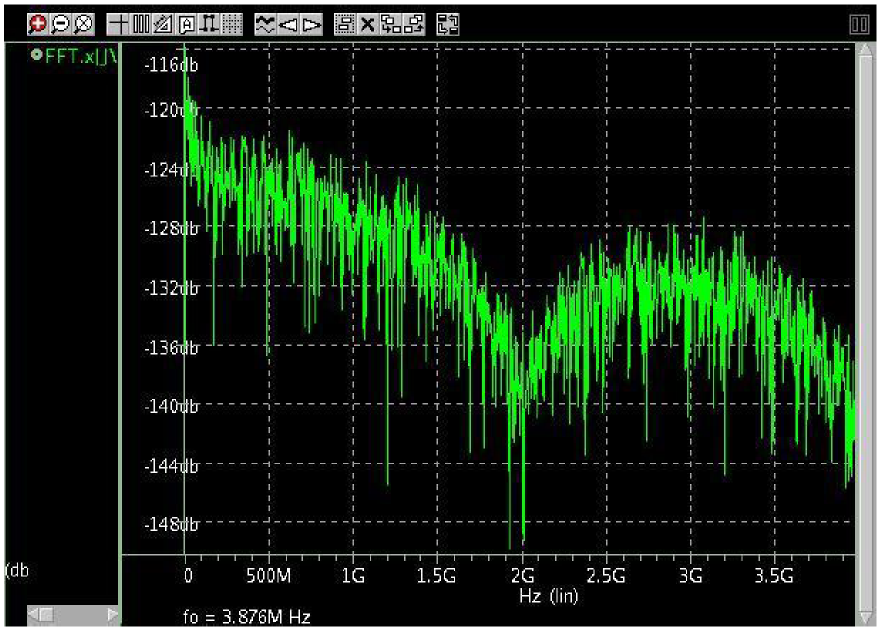


Figure 36 FFT of the TIE Jitter samples show the Jitter Spectrum

Error Handling, Error Recovery, Status Reporting

The following error checks are made with transient noise analysis:

- Verify that the specified output node exists.
- Verify non-negative integer values for “list” entries.

Note: Invalid values for FMIN and FMAX will be ignored and default values will be used.

The SDE solving can be substantially slower than transient analysis. Status reporting includes noise analysis progress.

References

[1] A. Demir, E.W.Y. Liu, A.L. Sangiovanni-Vincentelli, “Time-domain non-Monte Carlo noise simulation for nonlinear dynamic circuits with arbitrary excitations,” IEEE Trans. CAD, vol. 15, no. 5. May 1996.

[2] P. Bolcato and R. Poujois, “A new approach for noise simulation in transient analysis”, Proc. IEEE Int. Symp. Circuits Syst., pp. 887-890, May 1992.

[3] J. A. McNeill, “Jitter in ring oscillators,” Ph.D. dissertation, Boston University, 1994.

[4] M. Okumura and H. Tanimoto, “A time-domain method for numerical noise analysis of oscillators”, Proc. IEEE Asia Pacific Design Automation Conference, 1997.

[5] A. van der Ziel, Noise in Solid State Devices and Circuits, John Wiley & Sons, 1986.

[6] A. Hajimiri, S. Limotyrakis, and T.H. Lee, “Jitter and phase noise in ring oscillators,” IEEE J. Solid-State Circuits, vol. 34, no. 6, pp. 790-804, June 1999.

10

Simulation of Random Noise

Describes the characteristics of random signals, types of noise, component noise models, and noise simulation in HSPICE.

HSPICE ships several examples for your use; see Applications of General Interest Examples in the *HSPICE User Guide: Basic Simulation and Analysis* for paths to demo files.

Note: For `.AC`-related `.NOISE` analysis, see AC Small-Signal and Noise Analysis in the *HSPICE User Guide: Basic Simulation and Analysis*.

The following topics are covered in these sections:

- Introduction to Noise Sources
- Characteristics of Random Signals
- Noise Types
- Component Noise Models and HSPICE Noise Simulation

Introduction to Noise Sources

One of the essential techniques in designing an analog circuit is to be able to identify major noise sources in the circuit and know how to minimize their affect to an acceptable level. Noise can be loosely defined as a disturbance signal with random amplitude, which is usually an unwanted phenomenon in a circuit.

It is important to limit the power of noise to a low level relative to that of signal; otherwise, the signal can be greatly distorted or completely indistinguishable. They rely on simulation tools to help them identify the dominant sources of noise in the circuit, measure the noise levels (in power, current, or voltage), and eventually make changes in the design to reduce the noise levels if necessary.

Two types of noise are commonly defined: inherent and interference.

- Inherent noise is the noise created by the elements internal to the circuit (resistors, diodes, and transistors) caused by movement of electrons in the circuit.
- Interference noise refers to the unwanted signals from the environment absorbed by the circuit.

Eliminating interfering noise can be done by different methods of signal shielding or code modulation so the signal can be differentiated from the noise. Inherent noise, however, can only be affected by changes in the circuit topology and the bias currents.

This chapter discusses only inherent noise and the HSPICE simulations related to it. Whenever the term “noise” is used, it refers to “inherent noise”.

Presented first in the following sections are some theoretical definitions for noise sources in an analog circuit. These are followed by a discussion on how HSPICE performs noise simulation and how to interpret outputs from noise simulation.

Characteristics of Random Signals

The following sections describe noise sources in the time and frequency domains, and methods to deal with multitude noise sources in a circuit, according to these topics:

- Probability Distribution Function versus Power Spectral Density
- Multiple Noise Sources in a Circuit
- Summary

Probability Distribution Function versus Power Spectral Density

Noise is a signal with random amplitude. Figure 37 illustrates a sample noise signal in the time domain. The vertical axis could be either a branch current or a node voltage.

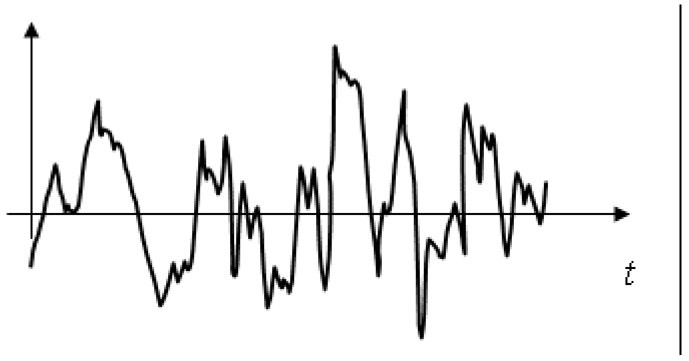


Figure 37 Typical noise signal (voltage or current) in the time domain

To simulate a noise source in the time domain, you have to know the Probability Distribution Function (PDF) of that signal and then create a voltage or current source that produces a random output signal conforming to the PDF. Figure 38 demonstrates a sample PDF for a random signal with Gaussian distribution. Extracting PDF parameters out of circuit components is not straightforward and therefore not common in analyzing noise.

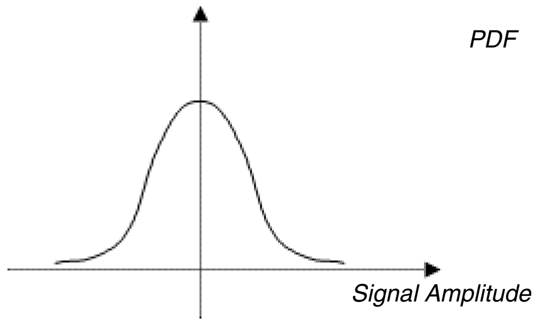


Figure 38 PDF for a random signal with Gaussian probability distribution

However, noise can be easily modeled and measured in the frequency domain. The noise models in the frequency domain also relate directly to the component and circuit characteristics. This is the reason noise analyses are usually done in the frequency domain.

While we cannot obtain a deterministic value for the noise amplitude in the time domain because it is a random variable, it is quite possible to determine the power of the noise if the noise is bandwidth limited. In other words, it is true that the amplitude of a noise signal changes randomly in the time domain. However, its power, as defined by Equation 59 is a fixed and finite value if and only if the signal is bandwidth limited.

Definition of Power for a noise voltage (a) and current (b), normalized for a 1-ohm load:

Equation 59

$$P = lim_{T \to \infty} \cdot 1/T \int_{-T/2}^{T/2} |v(t)|^2 dt \qquad \text{Eq. a}$$

$$P = lim_{T \to \infty} \cdot 1/T \int_{-T/2}^{T/2} |i(t)|^2 dt \qquad \text{Eq. b}$$

The power of a signal can also be described in the frequency domain; by means of a Power Spectral Density (PSD). A PSD identifies the portion of signal power present as a function of signal frequency. Figure 39 illustrates two PSD graphs.

Graph A resembles the PSD of a typical data signal in high speed and advanced analog communication; a signal with a limited bandwidth. It can be seen from the graph that the composing frequencies of the signal are between DC (0 Hz) and 200 MHz, with the bulk of power being concentrated around 100 MHz.

Graph B shows a signal with a flat power distribution; meaning that the power is equally distributed across all frequencies. Such a signal is referred to as “white noise”. Resistor thermal noise possesses such a PSD.

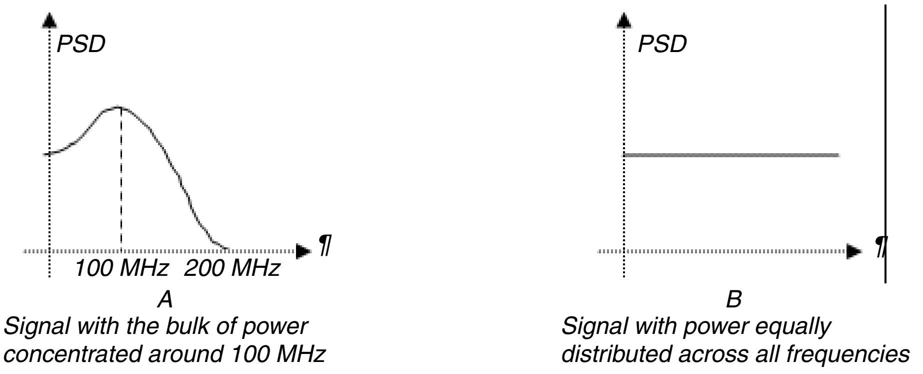


Figure 39 Sample PSD graphs

To obtain the total power of a signal within a certain bandwidth, you must calculate the area under the PSD curve within the given frequency range, as defined in Equation 60.

Definition of power based in the frequency domain, normalized for a 1 ohm load:

Equation 60

$$P = \int_{f\_1}^{f\_2} PSD(f) df$$

An example of power calculation in the frequency domain for a noise signal is shown in Figure 40. In this example, the PSD is a flat curve with a value of 5 x 10_{-20} V_2/Hz.

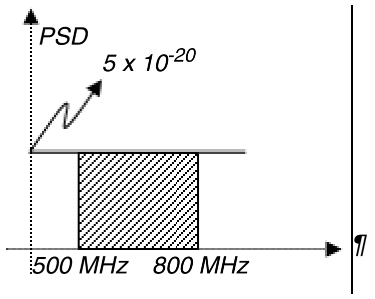


Figure 40 Sample PSD graphs

The power of the noise between the frequencies 500 MHz and 800 MHz is equal to the shaded area under the curve and is given by:

$$P_n = \int_{500M}^{800M} 5 \times 10^{-20} df = (800M - 500M) 5 \times 10^{-20} = 1.5 \times 10^{-11} V^2$$

The root mean square (RMS) value of the noise signal is given by Equation 61. The RMS value of a signal is defined as a DC value that would render the same power as the signal in question. For the general sinusoidal signal Acos(αt), the RMS value is $A / \sqrt{2}$.

RMS value of a signal is the root of the 1 ohm normalized signal power:

Equation 61

$$v_{rms} = \sqrt{P}$$

Which means the RMS voltage value of the noise signal in the above example is:

$$v_{rms} = \sqrt{1.5 \times 10^{-11}} = 5.48 \mu V$$

This means that the power of the noise is identical to that of a sinusoidal signal with a RMS value of 5.48 uV.

Multiple Noise Sources in a Circuit

In the usual case of dealing with a multitude of noise sources in a circuit, it is important to know how to add up the effects of noise to obtain the accumulated noise value.

If the noise sources are completely independent (that is, un-correlated), such as the noises generated by two separate resistors, their powers add up to their RMS values. Figure 41 shows how two independent noise sources, in this case both voltage sources, add up.

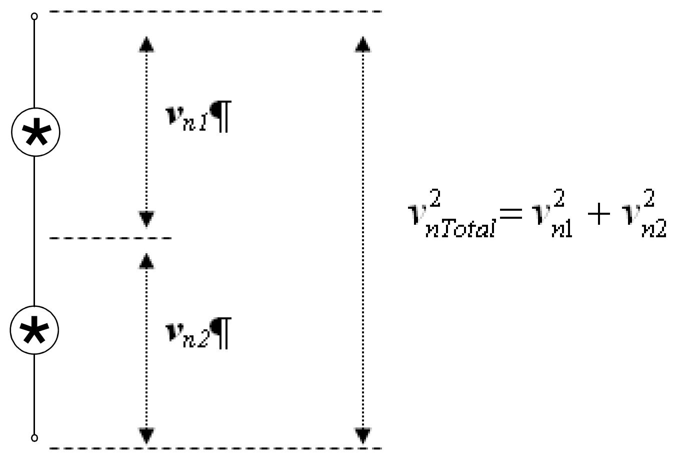


Figure 41 Noise summation: independent noise signals add in squared values

It might not be evident from the equation for V_{nTotal} in Figure 41 that when there are several noise sources in a circuit, only reducing the biggest noise contributors will reduce the total noise significantly. Reducing the smaller sources, even by large amounts does not have a considerable affect on the overall noise value because the power of two of signals magnifies the dominance of bigger sources over smaller ones.

To clarify this point, consider the case where two noise sources are present, with RMS values of v_1 = 2 uV and v_2 = 9 uV. The total noise is given by:

$$v_{nTotal} = \sqrt{(2)^2 + (9)^2} = 9.21\mu V$$

Now if in an attempt to lower the total noise the designer of the circuit cuts the 2 uV source by 50%, the total noise will be:

$$v_{nTotal} = \sqrt{(1)^2 + (9)^2} = 9.06\mu V$$

Now, if instead we cut the 9 uV source by only 10%, the total noise will be:

$$v_{nTotal} = \sqrt{(2^2) + (8.1^2)} = 8.34\mu V$$

This emphasizes the fact that to reduce the total noise, we should first concentrate on the strongest source of noise.

Summary

Important points from this section are:

- Inherent noise is best modeled in the frequency domain by its PSD.
- The power of noise depends on the bandwidth of the system and is defined as the area under the PSD curve between two given frequencies.
- When multiple noise sources are present, the square of their voltage (or current) values add.
- When multiple sources of noise are present in a circuit, the most effective way of reducing the total noise is to focus on reducing the most dominant noise contributor.

Noise Types

The following sections identify and describe three common types of noise in analog circuits:

- Thermal Noise
- Flicker Noise
- Shot Noise
- Summary

Thermal Noise

Thermal noise is generated by random movements of electrons in a resistive conductor. Any circuit element with resistive characteristics, whether a resistor or the base junction series resistor in a BJT's small signal model, disseminates thermal noise. The models for thermal noise voltage source and their equivalent current source are shown in Figure 42.

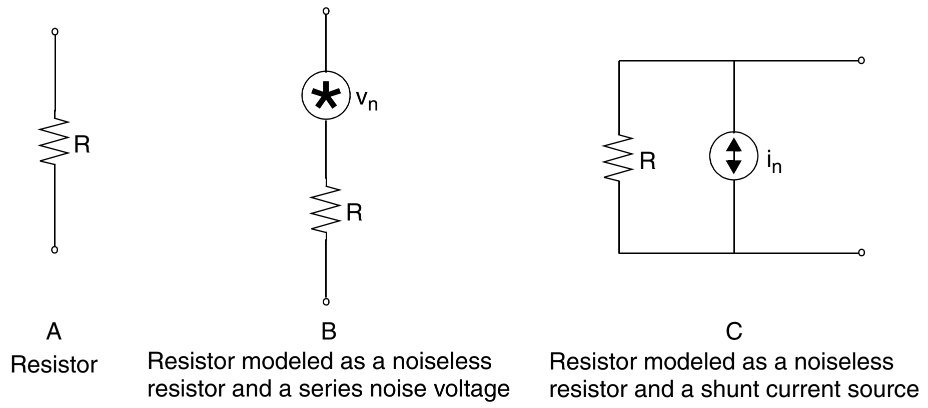

R
v_n
R
R
i_n
A
Resistor
B
Resistor modeled as a noiseless resistor and a series noise voltage
C
Resistor modeled as a noiseless resistor and a shunt current source

Figure 42 Thermal noise voltage sources and their equivalent circuits

The voltage and current PSD functions for resistor thermal noise are:

$$V_n^2(f) = 4kTR \qquad v^2/Hz$$

$$I_n^2(f) = 4kT/R \qquad A^2/Hz$$

where:

k is Boltzmann's constant equal to 1.38 x 10^{-23}.
T is the temperature in Kelvin degrees: 1 °K = °C + 273. For a room temperature of 25 °C (88 °F), the default temperature setting in HSPICE, T will be 298 °K.
R is the resistance in ohms.

It should be noted that the two voltage and current PSD models are equivalent, they both produce the same amount of open circuit voltage or short circuit current and can be used interchangeably during noise analysis.

Notice that thermal noise voltage PSD is proportional to the resistance while the current PSD is inversely proportional to the resistance. By looking at the

PSD models, a general conclusion can be made; presence of bigger resistors in a circuit usually translates to a bigger thermal noise voltage.

The PSD models have been explicitly described as functions of frequency to re-emphasize the fact that the PSD is only meaningful in the frequency domain context.

The above models suggest that PSDs are flat across all frequencies. This assumption will no longer be true for frequencies above a few hundred gigahertz.

Example 1

What will be the RMS voltage value of thermal noise generated by a 10 kohm resistor operating in the 1- to 1.2-GHz frequency range? Assuming a room temperature of 25 °C or 298 °K, the PSD function will be (see also Figure 43):

$$V_n^2(f) = 4kTR = 4(1.38 \times 10^{-23})(298)(10K) = 1.64 \times 10^{-16} \qquad V^2/Hz$$

$$v_{nrms}^2 = \int PSD(f)df = \int (1.64 \times 10^{-16})df = (1.64 \times 10^{-16}) \times (0.2G) = 3.28 \times 10^{-8} \qquad V^2$$

$$v_{nrms} = \sqrt{3.28 \times 10^{-8}} = 181.11\mu V$$

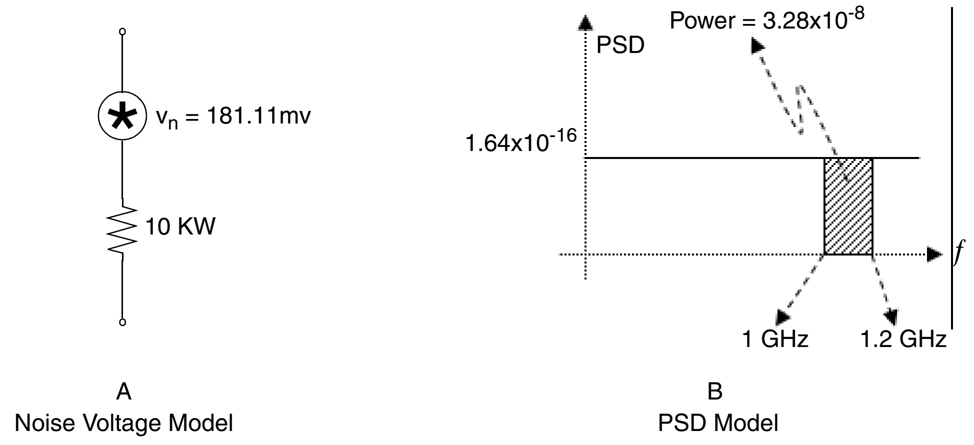


Figure 43 Noise models for 10 kohm resistor

Flicker Noise

Flicker noise is noise generated by fluctuations in the average current travel time in a conductor. As electrons take different random paths to get from one end of a resistor to the other at any given time, the average current will experience different resistance along the way, leading to fluctuations in current, hence the generation of flicker noise. Figure 44 on page 266 illustrates how electrons can fall into paths with different lengths while flowing through a resistor.

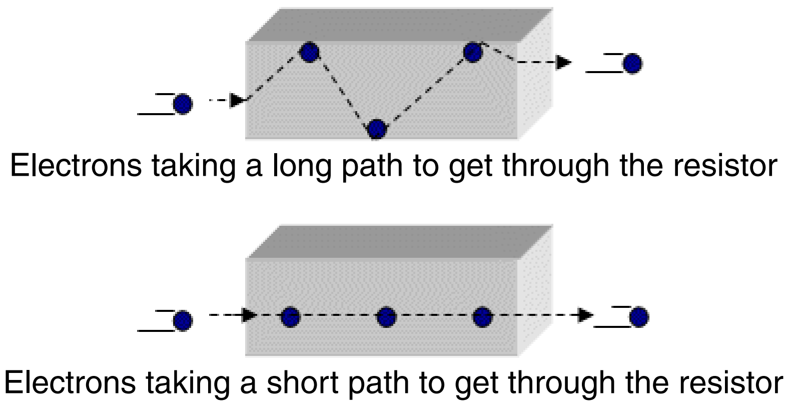

Electrons taking a long path to get through the resistor
Electrons taking a short path to get through the resistor

Figure 44 Paths for current through a resistor

By shrinking the cross-section area of the conducting material, there will be smaller differences between different current paths and the flicker noise will shrink. That is why thin film resistors have a much smaller flicker noise than carbon rod resistors, which have a much larger conducting cross-section.

In addition, at high frequencies, the current tends to travel through the outer surface of the conductor, a phenomenon know as “skin effect”, effectively shrinking the conducting cross-section to a very thin layer. This results in smaller differences in the current path lengths and consequently smaller flicker noise. Because of stronger flicker noise effects at higher frequencies, flicker noise is inversely proportional to frequency. That is why it is also called “1/f noise”. Flicker noise is only significant at lower frequency values, but it can have a major impact on nonlinear circuits at advanced analog frequencies.

The major sources of flicker noise in analog circuits are semiconductor components. Resistors can be manufactured to produce small amount of flicker noise and are not usually taken into account for calculation of flicker noise in a circuit.

Flicker noise is usually represented as a current PSD for resistors and semiconductors with this equation:

$$I_n^2(f) = K_f I/f \qquad A^2/Hz$$

Where,
I is the current through the resistor.
f is frequency in hertz.
K_f is a constant which depends on the characteristics and geometry of the conductor.

The PSD for flicker noise is depicted in Figure 45.

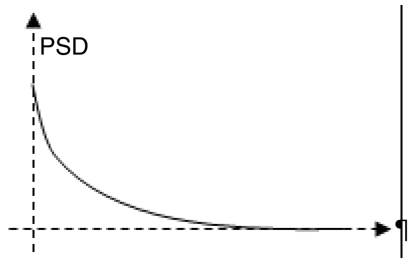


Figure 45 PSD for flicker noise

HSPICE uses more general models for CMOS flicker noise PSDs. One of the models used for CMOS transistor is the SPICE2/Berkeley noise model:

$$I_n^2\ (f) = K_f I^{A_f} / (C_{ox} L^2 f^{E_f}) \qquad A^2/Hz$$

where:
K_f is the flicker noise parameter.
A_f is the current exponent.
C_{ox} is the Gate oxide capacitance.
L is the effective length of the transistor.
E_f is the frequency exponent.

Another model used by HSPICE for flicker noise calculation is the BSIM model, which looks very similar to the above equation. It is important to note that you usually must not be concerned with the value settings of these parameters. They must be defined and set to proper values in the transistor models defined in the technology library.

The combined PSD for flicker and thermal noise is shown in Figure 46 on page 268.

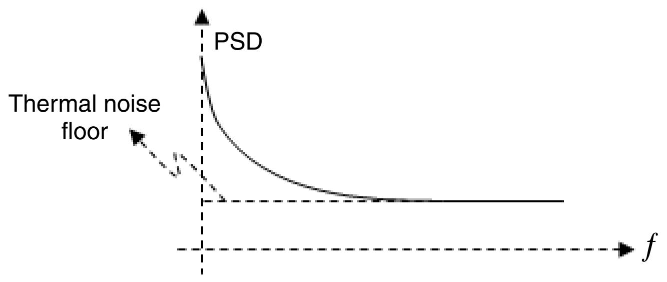


Figure 46 PSD of combined thermal and flicker noise

Note that flicker noise is the dominant source of noise at lower frequencies and as its magnitude shrinks at higher frequencies, the thermal noise will become the dominant source of noise. Thermal noise is also referred to as “the noise floor” because that is the minimum possible amount of noise in a circuit.

Shot Noise

Shot noise is only generated by semiconductor elements and is caused by random passage of electrons and holes across a potential barrier, such as a P-N junction. Shot noise is often represented by a current PSD, known as “Schottky formula”:

$$I_n^2(f) = 2qI \qquad A^2/Hz$$

Where,

q = 1.6 x 10^{-19} C is the charge of an electron.
I is the bias current of the junction and can be I_c, collector bias currents for a BJT transistor.

Shot noise PSD curves are flat across all frequencies.

Summary

The three major types of noise in an analog circuit are:

- Thermal noise, also known as white noise, is generated by resistors in the circuit. Thermal noise is a function of conductor resistance.
- Flicker noise, also called 1/f noise, is mainly generated by transistors in a circuit. It is a function of component geometry and its magnitude drops as frequency increases.
- Shot noise is caused by bias currents in the base and collector of BJT transistors.

Component Noise Models and HSPICE Noise Simulation

The following sections describes how HSPICE models noisy elements, calculates the total output noise PSD, obtain the total output noise voltage PSD, and how the RMS output noise voltage can be deduced from the total output noise PSD.

The following topics are discussed:

- Element Noise Models
- HSPICE Noise Simulation
- Summary

Element Noise Models

Each resistor, diode, and transistor in a circuit generates at lease some type of inherent noise. HSPICE models the noise generating elements in a circuit as a noiseless element combined with a noise current or voltage source with a known PSD. Figure 47 includes the listing of the four noise generating elements in analog circuits and their equivalent noise models.

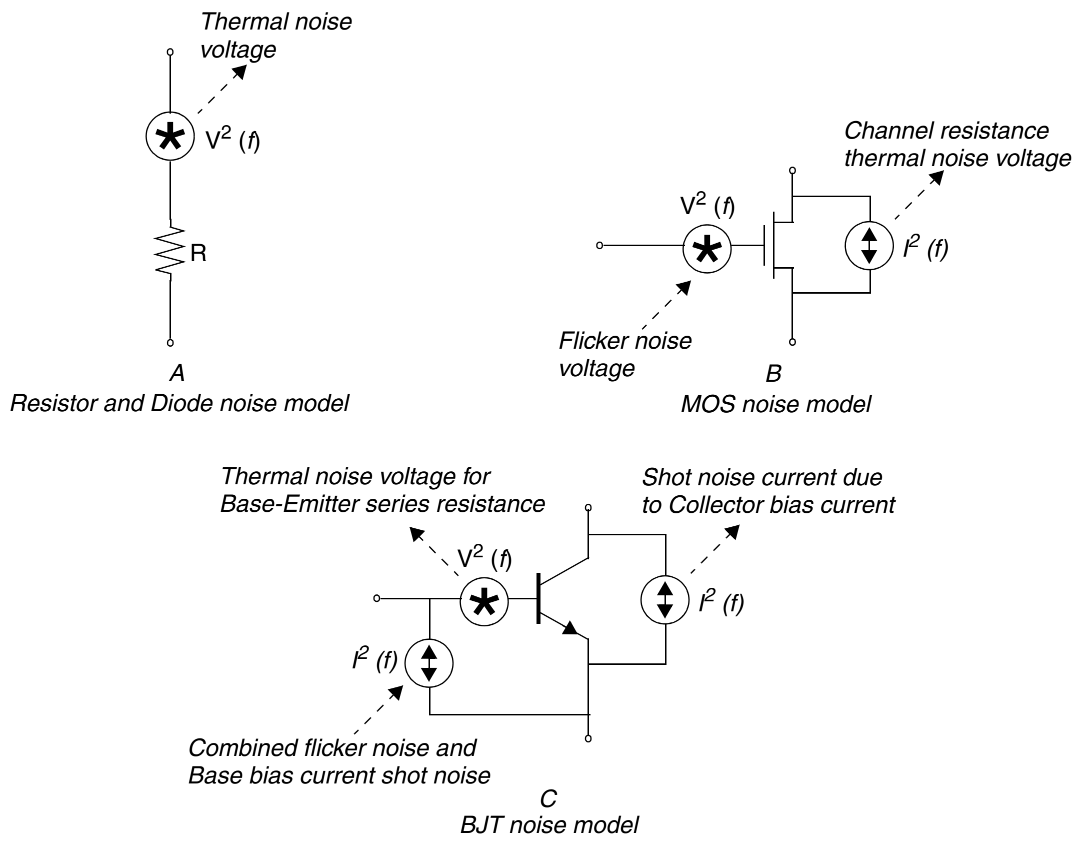

Thermal noise voltage
$V^2(f)$
R
A
Resistor and Diode noise model
Channel resistance thermal noise voltage
$V^2(f)$
$I^2(f)$
Flicker noise voltage
B
MOS noise model
Thermal noise voltage for Base-Emitter series resistance
Shot noise current due to Collector bias current
$V^2(f)$
$I^2(f)$
$I^2(f)$
Combined flicker noise and Base bias current shot noise
C
BJT noise model

Figure 47 Noise generating circuit elements and their noise models

The noise model for a diode is identical to that of a resistor with a resistance of:

$$R_d = kT/qI_d$$

Where,
R_d is the equivalent resistance of the diode.
k is Boltzmann's constant.
q is the charge of an electron.
I_d is the bias current of the diode.

HSPICE Noise Simulation

To perform noise analysis in HSPICE, a `.NOISE` statement along with an `.AC` statement must be used. The `.NOISE` syntax for the noise analysis is:

```
.NOISE v(out ,ref) src interval
```

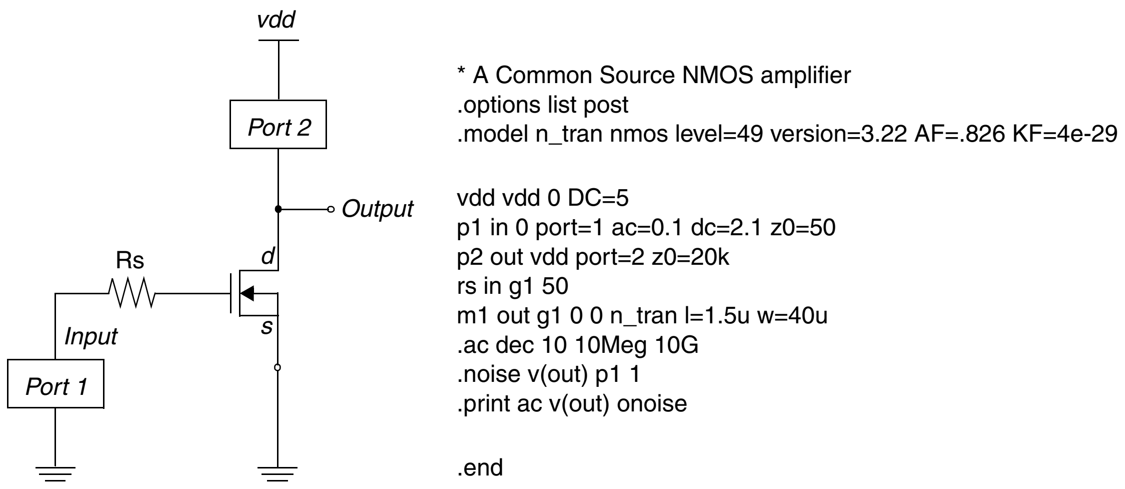
The frequency points at which the noise calculations are performed are the same points defined by the `.AC` statement. The noise calculations for each frequency point will be output to the listing file.

RX Transfer Function

RX is the transfer function from the noise source to the circuit output. Since the noise sources for built-in devices are all current sources with PSDs (power spectral densities) in A^2/Hz, and the output is usually a voltage (specified on `.NOISE` statements), RX is a transimpedance (V/A).

The total output noise voltage is the integrated output noise. The output noise at a given frequency is a PSD in V^2/Hz. This can be integrated over the frequency range (using the `.AC` command) to get a total output noise in V^2, if you take the square root, you get the total output noise in V.

For example (see Example 48 on page 271), take a simple common source NMOS amplifier to show how HSPICE calculates the output noise. Note that in this example, the output port element (P2) is used as a pull-up resistor in the circuit. For the path to the demo file, `noise_app_orig.sp`, see Applications of General Interest Examples in *HSPICE User Guide: Basic Simulation and Analysis*. The full path to this example is `$`*`installdir`*`/demo/hspice/apps/noise_app_orig.sp`.



```
* A Common Source NMOS amplifier
.options list post
.model n_tran nmos level=49 version=3.22 AF=.826 KF=4e-29

vdd vdd 0 DC=5
p1 in 0 port=1 ac=0.1 dc=2.1 z0=50
p2 out vdd port=2 z0=20k
rs in g1 50
m1 out g1 0 0 n_tran l=1.5u w=40u
.ac dec 10 10Meg 10G
.noise v(out) p1 1
.print ac v(out) onoise

.end
```

Figure 48 A common source NMOS amplifier and its netlist

The first step in the noise analysis is to set all the signal voltage and current sources to 0. The equivalent circuit for our example, after setting `vdd=0`, is shown in Figure 49.

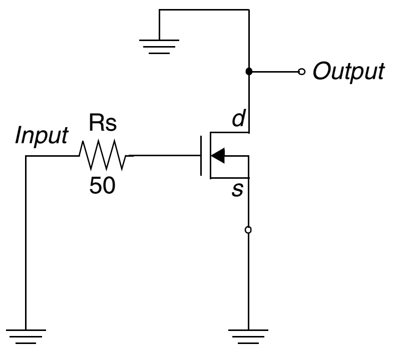


Figure 49   Equivalent circuit after independent V and I sources are set 0

HSPICE models each resistor, diode, and transistor with its noise model, and then calculates the output voltage resulting from the noise signal, one element at a time.

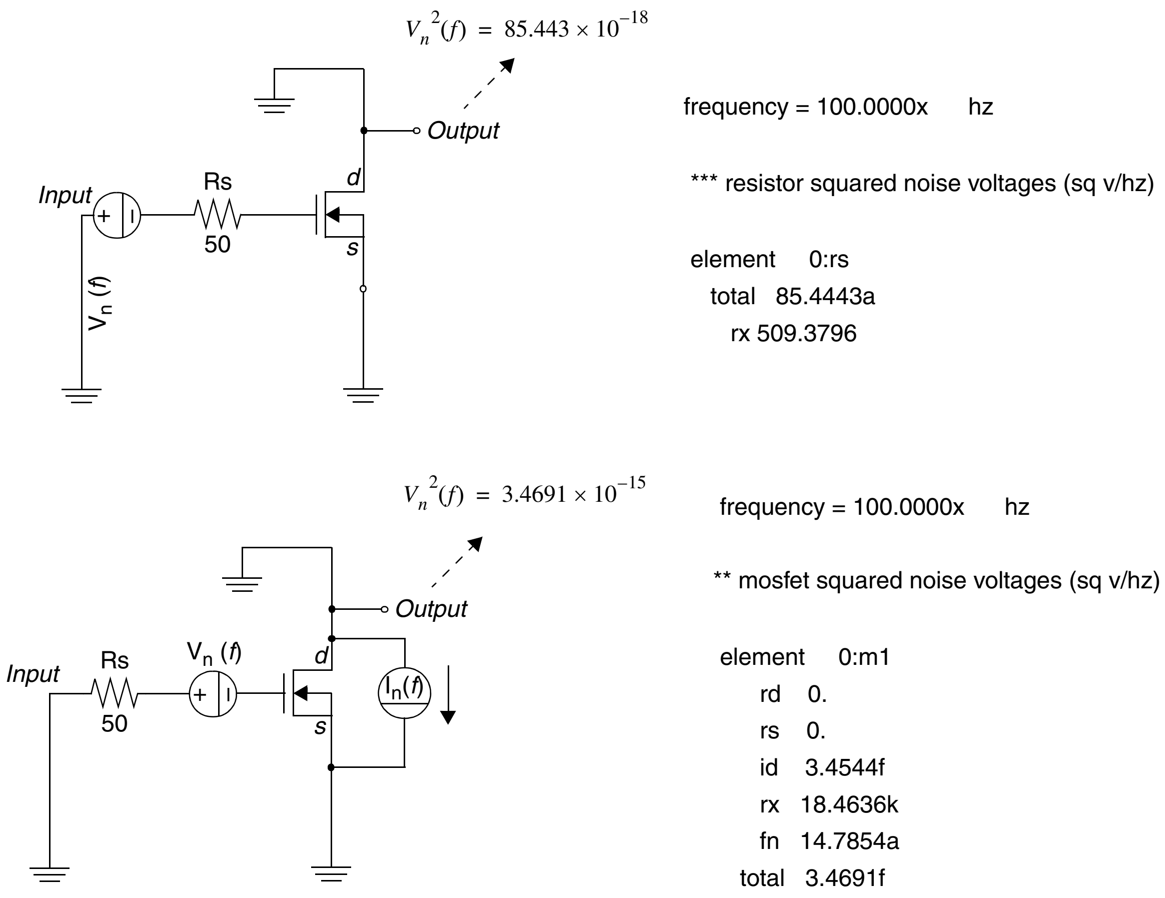
To start, it replaces Rs with its noise model, as shown in Figure 50, and calculates the PSD of the noise voltage as seen at the output port. HSPICE reports an output voltage PSD of 85.4443x10^{-18} V^2/Hz, caused by the thermal noise model of Rs. The value rx shown is used to obtain the voltage transfer function from the Rs noise source to the output port:

$$\text{Voltage Transfer Function} = rx = \frac{\text{output\_voltage}}{\text{element\_noise\_current}}$$

HSPICE uses *rx* for its internal calculations and you need not pay particular attention to it.

Figure 50   Circuit noise model for Rs and analysis output at 100 MHz

The circuit for calculating the PSD of output noise generated by the NMOS is given in Figure 51.


$V_n^2(f) = 85.443 \times 10^{-18}$
Output
Input
Rs
50
d
s
$V_n(f)$
$V_n^2(f) = 3.4691 \times 10^{-15}$
Output
Input
Rs
50
$V_n(f)$
d
s
$I_n(f)$

```
frequency = 100.0000x     hz

*** resistor squared noise voltages (sq v/hz)

element     0:rs
  total   85.4443a
     rx 509.3796
```

```
frequency = 100.0000x     hz

** mosfet squared noise voltages (sq v/hz)

element     0:m1
     rd    0.
     rs    0.
     id    3.4544f
     rx   18.4636k
     fn   14.7854a
  total    3.4691f
```

Figure 51   Circuit noise model for NMOS and analysis output

The total PSD at 100 MHz will be the sum of all individual PSDs (the square of noise signals add):

$$V_{nTotal}^2\ (f = 100MHz) = 85.4443 \times 10^{-18} + 3.4691 \times 10^{-15} = 3.5546 \times 10^{-15} \qquad V^2/Hz$$

$$V_{nTotal}(f = 100MHz) = \sqrt{3.5546 \times 10^{-15}} = 59.6204 \times 10^{-9} \qquad V/\sqrt{Hz}$$

It is common to represent the total noise generated by a circuit as the equivalent input noise (otherwise called *input referred noise*). The input referred noise voltage/current is the voltage/current that if applied at the input of the noise-free circuit, would have generated the same output voltage as the total output noise voltage.

To calculate the equivalent input noise signal, you have to obtain the transfer function from input to output. For the example circuit above, the transfer function will be:

$$TF = \frac{\text{Output Voltage}}{\text{Input Voltage}} = \frac{V(out)}{VS}$$

In the case where the input source is a current source instead of a voltage source, the TF would be the ratio of output voltage over input current.

The equivalent input noise voltage (or current in case of the input current source) is given by:

$$\text{Input Referred Noise} = \frac{\text{Total Output Noise Voltage}}{TF}$$

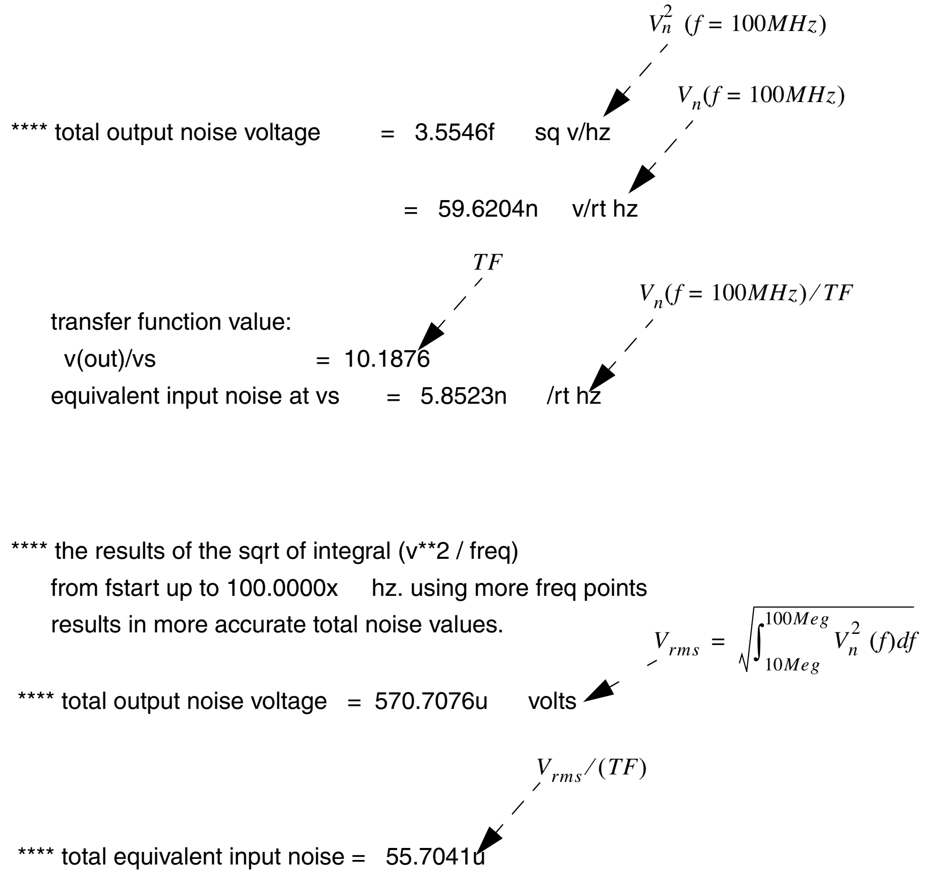
The input referred noise is useful to calculate the circuit's “noise figure,” which is a measure of how much a circuit is generating inherent noise, a large noise figure indicates a high level of noise generation by the circuit. Noise figure is defined as:

$$\text{Noise Figure} = 10 \cdot \log \cdot (\text{Noise Factor})$$

Where Noise Factor is given by:

$$\text{Noise Factor} = 1 + \frac{\text{Input Referred Inherent Noise Power}}{\text{Power of External Noise at the Input}}$$

Example 52 on page 275 shows the HSPICE noise analysis summary output for the above circuit.



```
**** total output noise voltage        =  3.5546f     sq v/hz

                                  =  59.6204n     v/rt hz

  transfer function value:
    v(out)/vs                  = 10.1876
  equivalent input noise at vs      =  5.8523n      /rt hz

**** the results of the sqrt of integral (v**2 / freq)
     from fstart up to 100.0000x     hz. using more freq points
     results in more accurate total noise values.

 **** total output noise voltage  =  570.7076u      volts

 **** total equivalent input noise =   55.7041u
```

Figure 52   Noise summary for noise performed at 100 MHz

Summary

To summarize,

- HSPICE model elements as noiseless elements connected to noise-generating voltage and current sources.
- To calculate the total output noise PSD, set all signal and supply sources to zero. The tool then, one-by-one, replaces the noise generating elements with their equivalent noise models, and calculates the noise voltage PSD at

the output caused by the element's noise source (similar to an `.AC` analysis). It then repeats the same process for the next noise-generating element.

- Once the output noise PSDs are calculated for all elements, HSPICE adds the PSDs together to obtain the total output noise voltage PSD.
- The RMS output noise voltage and the input referred noise can be deduced from the total output noise PSD.

Part 4: Behavioral Modeling Applications

This Part contains the following chapters/topics.

- Chapter 11, Behavioral Modeling
- Chapter 12, Modeling Filters and Networks

11
Behavioral Modeling

Describes how to create and use behavioral models.

Behavioral modeling substitutes more abstract, less computationally intensive, circuit models for lower-level descriptions of analog functions. These simpler models emulate the transfer characteristics of the circuit elements that they replace, but with increased efficiency. Behavioral modeling substantially reduces the actual simulation time per circuit. At the level of an entire design and simulation cycle, design efficiency greatly increases, and you can complete a design (from concept to marketable product) in substantially less time.

HSPICE ships numerous examples for your use; see Behavioral Application Examples for paths to demo files.

These topics are presented in the following sections:

- Behavioral Design Process
- Using Behavioral Elements
- Voltage and Current Controlled Elements
- Modeling with Digital Behavioral Components
- Calibrating Digital Behavioral Components
- Analog Behavioral Elements
- Op-Amps, Comparators, and Oscillators
- Phase-Locked Loops (PLL)

Behavioral Design Process

HSPICE provides specific modeling elements that promote the use of behavioral and mixed signal techniques. These models include controllable sources that you can configure, to emulate op-amps, single-input or multi-input logic gates, or any system with a continuous algebraic transfer function.

- You can create these functions in algebraic form, or in the form of coordinate pairs.
- You can use digital stimulus files, to enter logic waveforms into the simulation deck, rather than using piecewise linear sources to enter digital waveforms.
- You can define clock rise times, fall times, periods, and voltage levels.

With HSPICE behavioral models, the typical design cycle for a circuit or system is:

1. Fully simulate a subcircuit, with pertinent inputs, characterizing its transfer functions.
2. Determine which HSPICE elements, singularly or in combination, accurately describe the transfer function.
3. Reconfigure the subcircuit appropriately.
4. After you verify the behavioral model, substitute the model into the larger system, in place of the lower-level subcircuit.

Using Behavioral Elements

Behavioral elements offer a higher level of abstraction, and faster processing, compared to a lower-level description of an analog function.

- System-level designers can use function libraries of subcircuits, containing these elements, to describe parts such as:
    - Op-amps
    - Vendor-specific output buffer drivers
    - TTL drivers
    - Logic-to-analog converters

   - Analog-to-logic simulator converters
- Integrated Circuit designers can use these elements to reduce design time, especially when designing filters and signal processors.

Behavioral elements use an arbitrary algebraic equation, as a transfer function to either a voltage (E) or current (G) source. This function can include:

- Nodal voltages
- Element currents
- Time
- Other parameters which you define

A good example of this is a VCO, where `control` is the input voltage node, and `osc` is the oscillator output:

```
Evco osc 0 VOL='voff+gain*\\
     SIN(6.28*freq*(1+V(control))*TIME)'
```

You can use subcircuits to encapsulate a function.

- If you split the function definition from the use, you create a hierarchy.
- If you pass parameters into the subcircuit, you create a parameterized cell.
- If you create a full transistor cell library, and a behavioral representation library, you can include mixed-signal functions within HSPICE.

You can use the built-in `OPTIMIZE` function to calibrate the behavioral elements from a full transistor circuit.

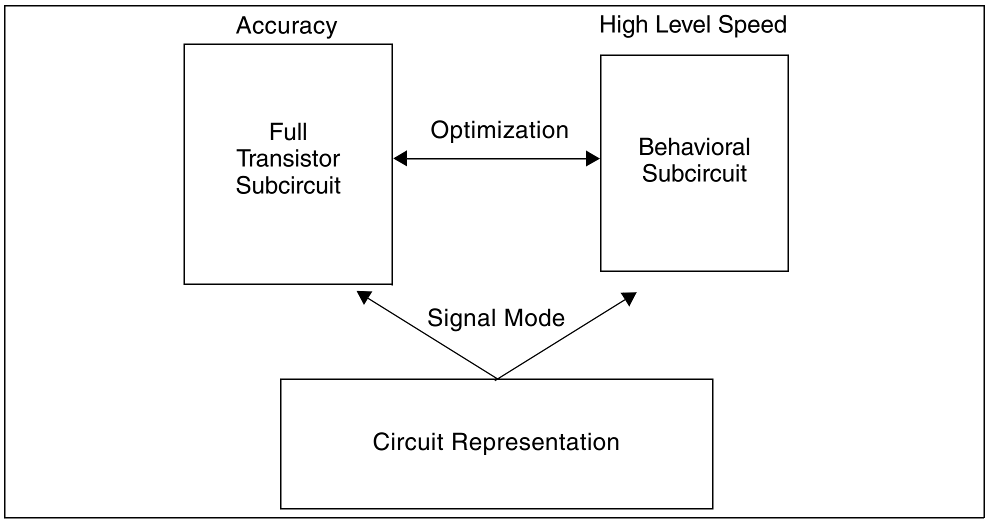


Figure 53 Netlisting by Signal Mode

The following sections discuss these topics:

- Controlled Sources
- Libraries

Controlled Sources

Controlled sources model both analog and digital circuits, at the behavioral level. This reduces simulation times for mixed signals, and models system-level operations. Controlled sources also model gate-switching action, for behavioral modeling of digital circuits. For analog behavioral modeling, you can program the controlled sources as mathematical functions. These functions can be either linear or non-linear, depending on other nodal voltages and branch currents.

Libraries

The Discrete Device Library contains standard industry IC components. You can use this library to model board-level designs that contain any of the following:

- Transistors
- Diodes

- Opamps
- Comparators
- Converters
- IC pins
- Printed circuit board traces
- Coaxial cables

You can also model drivers and receivers, to analyze transmission line effects, power line noise, and signal line noise.

Voltage and Current Controlled Elements

HSPICE provides two voltage-controlled and two current-controlled elements, known as E, F, G, and H-elements. For a description of these elements, see Sources and Stimuli, in *HSPICE User Guide: Basic Simulation and Analysis*.

Modeling with Digital Behavioral Components

The following sections show how to model, using digital behavioral components and discuss these topics:

- Behavioral AND and NAND Gates
- Behavioral D-Latch
- Behavioral Double-Edge Triggered Flip-Flop

Behavioral AND and NAND Gates

The following example uses a G Element to model a 2-input AND gate. An E Element models a two-input NAND gate. Figure 54 shows the resulting waveforms. This example is located in the following directory:

$installdir/demo/hspice/behave/behave.sp

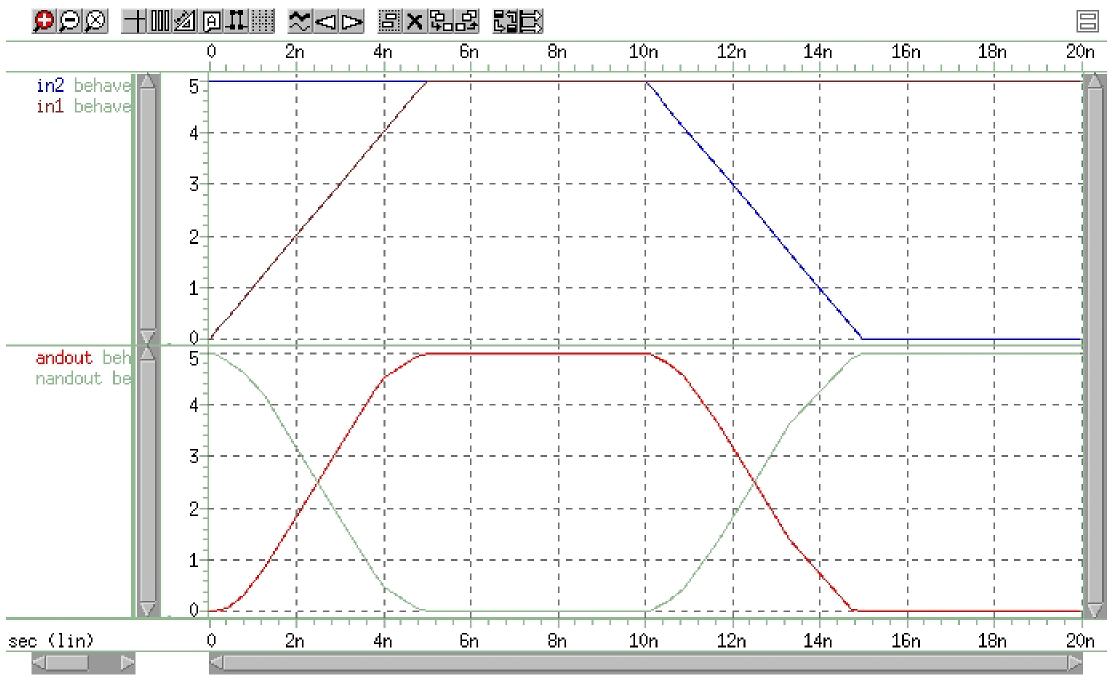


Figure 54 NAND/AND Gates

Behavioral D-Latch

This example uses one input NAND gates, and `NPWL`/`PPWL` functions, to model a D flip-flop.

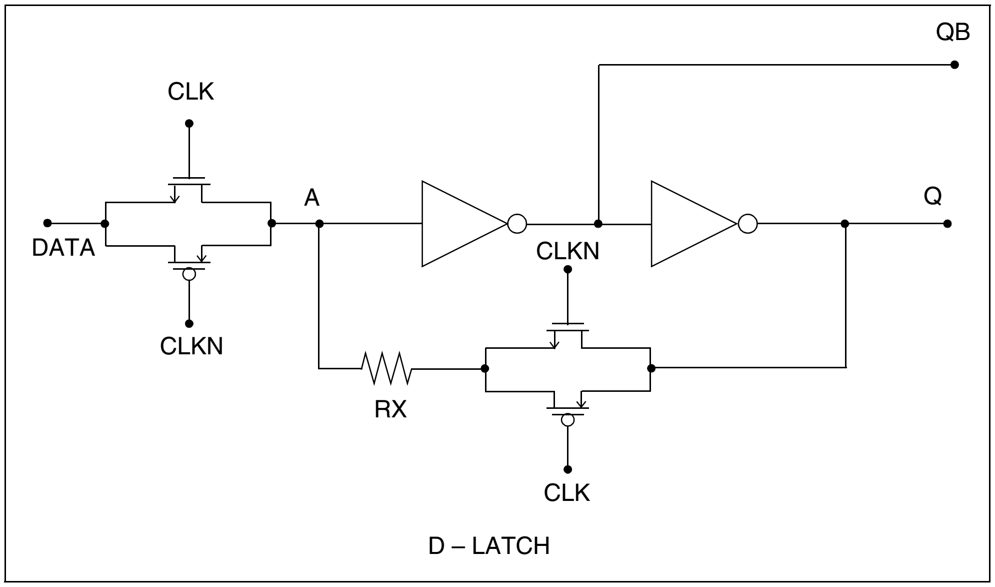


Figure 55 D-Latch

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/dlatch.sp

The file contains the following examples:

- Waveforms
- Subcircuit Definitions for Behavioral N-Channel MOSFET
- Behavioral P-Channel MOSFET

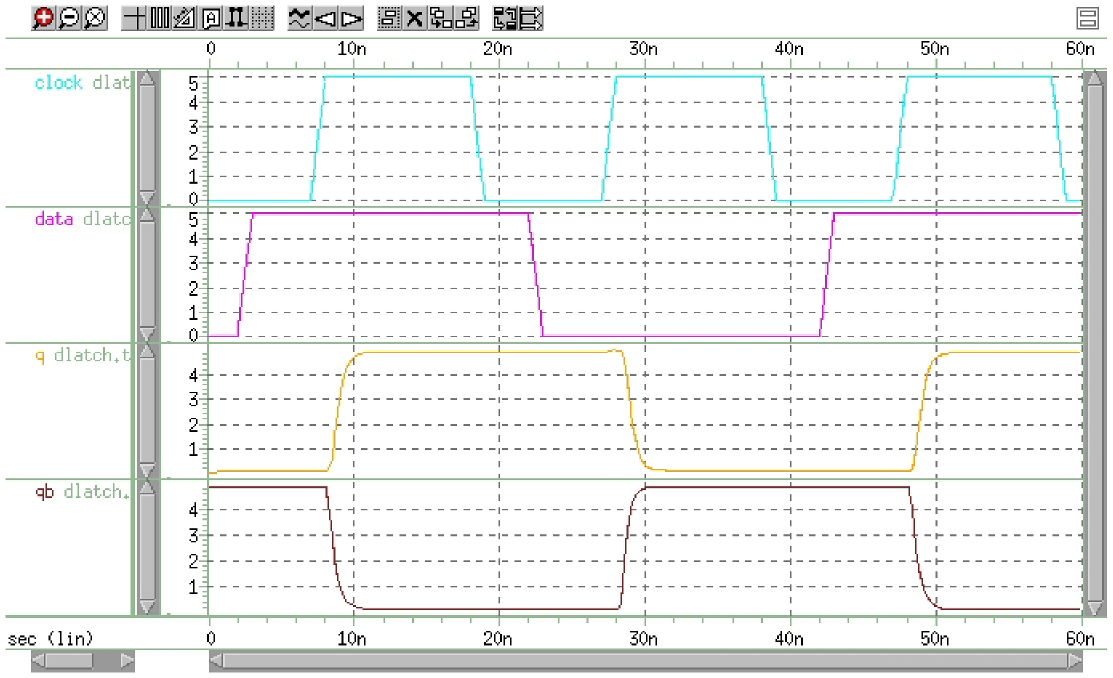


Figure 56 D-Latch Response

Behavioral Double-Edge Triggered Flip-Flop

This example uses the D_LATCH subcircuit from the previous example, and several NAND gates, to model a double-edged, triggered flip-flop.

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/det_dff.sp

- Main Circuit
- Subcircuit Definitions

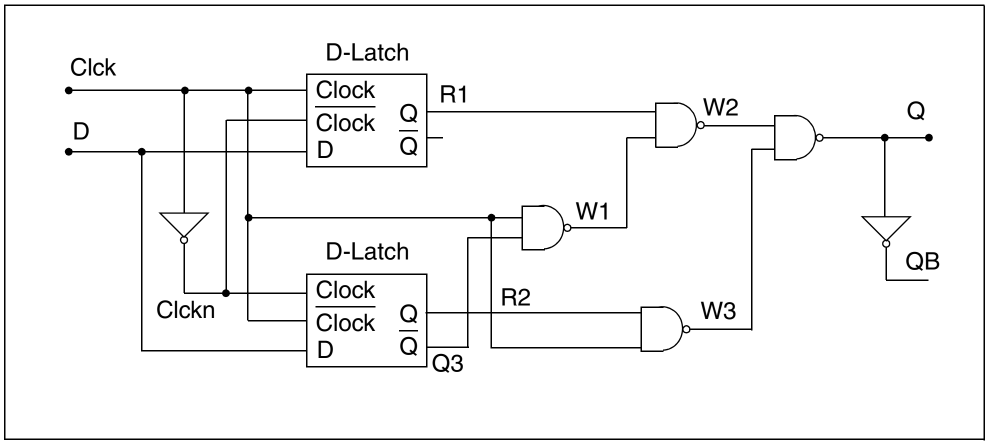


Figure 57   Double-Edge Triggered Flip-Flop

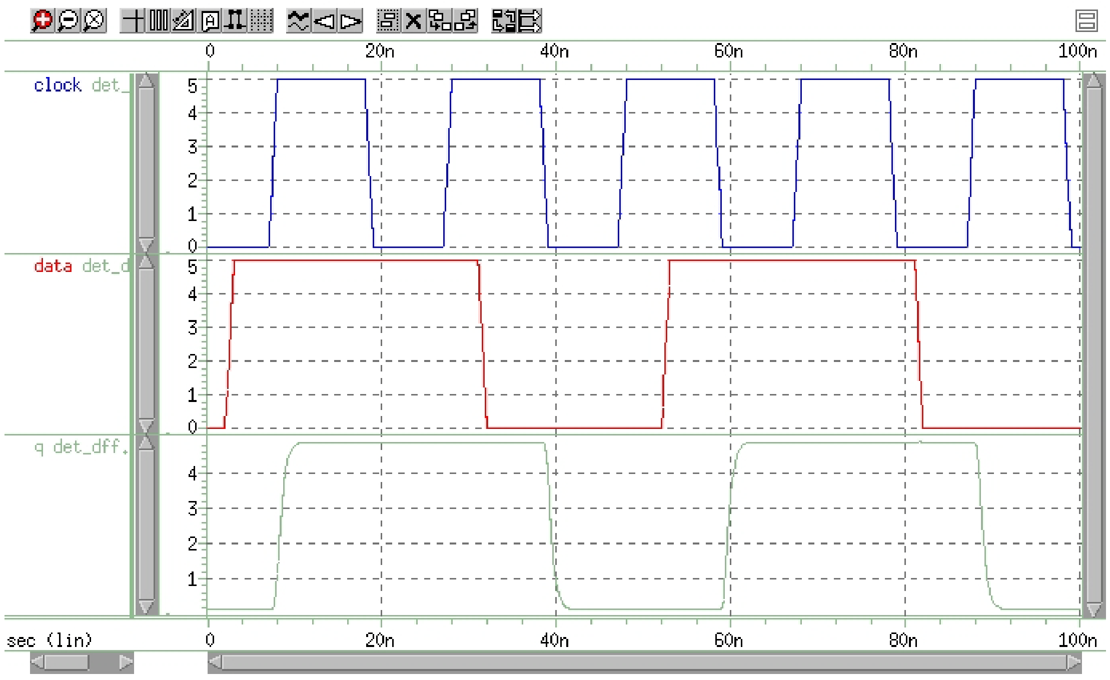


Figure 58   Double Edge Triggered Flip-Flop Response

Calibrating Digital Behavioral Components

- Building Behavioral Lookup Tables
- Optimizing Behavioral CMOS Inverters
- Optimizing Behavioral Ring Oscillators

Building Behavioral Lookup Tables

The following simulation demonstrates an ACL family output buffer, with 2ns delay, and 1.8ns rise and fall time. It also shows ground and VDD supply currents, and internal ground bounce due to the package.

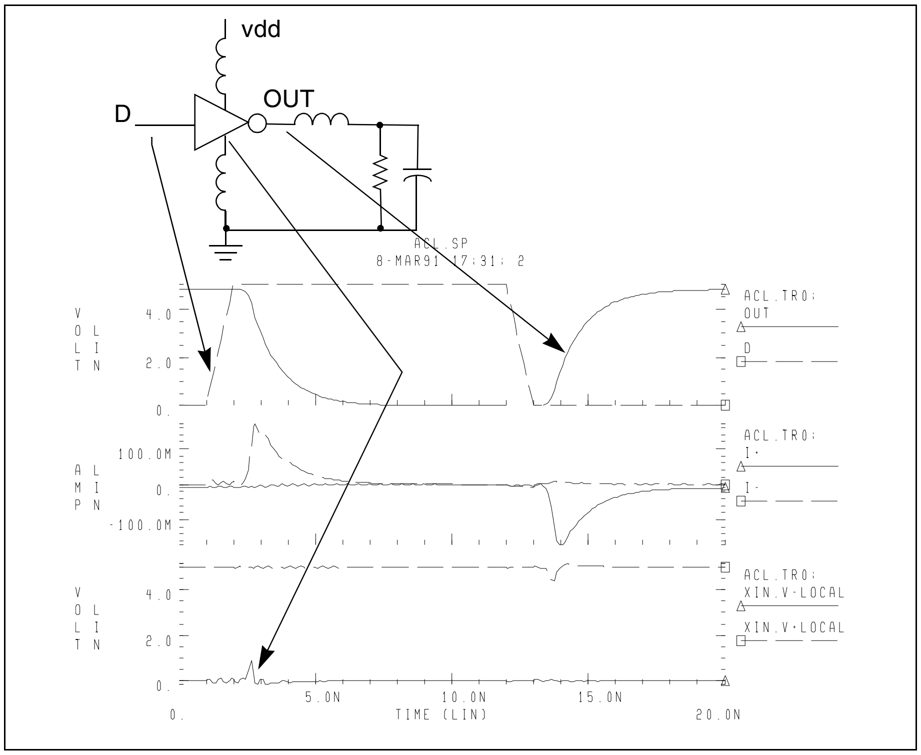


Figure 59 ACL Family Output Buffer

The following commands automatically measure the datasheet quantities, such as TPHL, risetime, maximum power dissipation, and ground bounce.

```
.MEAS tphl trig v(D) val='.5*vdd' rise=1
+ targ v(out) val='.5*vdd' fall=1
.MEAS risetime trig v(out) val='.1*vdd' rise=1
+ targ v(out) val='.9*vdd' rise=1
.MEAS max_power max power
.MEAS bounce max v(xin.v_local)
```

The inverter consists of capacitors, diodes, one-dimensional lookup table MOSFETs, and a special low-pass delay element. A property of the low-pass delay element, attenuates pulses that are narrower than the delay value.

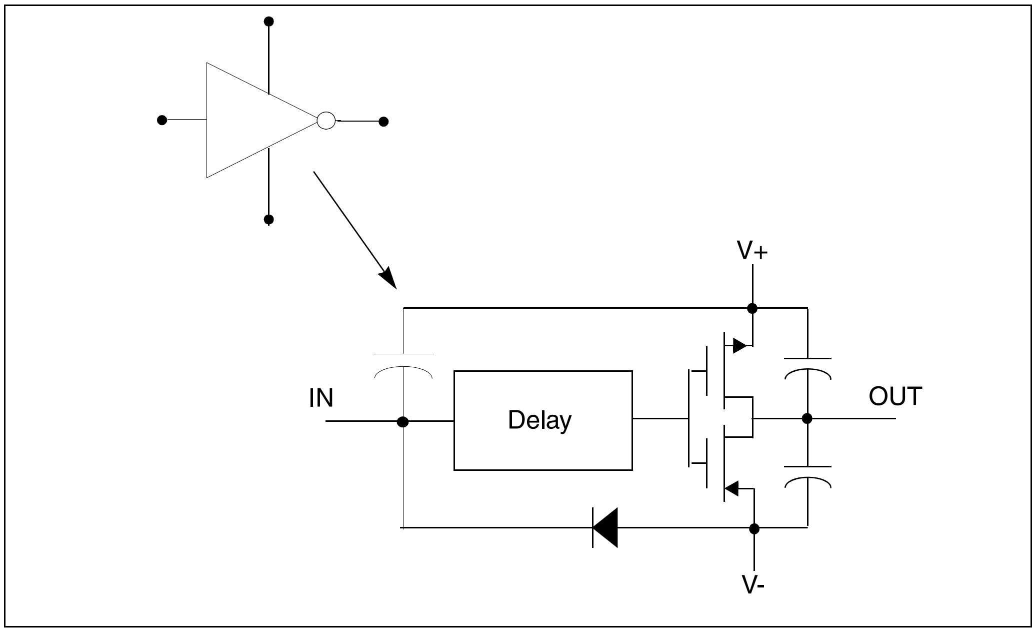


Figure 60   Inverter

Subcircuit Definition

```
.subckt inv in out v+ v-
cout+ out_l v+ 2p
cout- out_l v- 2p
xmp out_l inx v+ pmos
xmn out_l inx v- nmos
e inx v- delay in v- td=1n
din v- in dx
.model dx d cjo=2pf
chi in v+ .5pf
.ends inv
```

One-dimensional lookup tables represent the behavioral MOSFETs.

Behavioral N-Channel MOSFET

The following example is a Drain Gate source.

```
.subckt nmos 1 2 3
gn 3 1 VCR npwl(1) 2 3 scale=0.008
* VOLTAGE   RESISTANCE
+ 0.    495.8840g
+ 200.00000m    456.0938g
+ 400.00000m    141.6902g
+ 600.00000m    7.0624g
+ 800.00000m    258.9313meg
+ 1.00000    6.4866meg
+ 1.20000    842.9467k
+ 1.40000    21.6882k
+ 1.60000   170.8367k
+ 1.80000    106.4944k
+ 2.00000    72.7598k
+ 2.20000    52.4632k
+ 2.40000    38.5634k
+ 2.60000    8.8056k
+ 2.80000    5.2543k
+ 3.00000    4.3553k
+ 3.40000    3.4950k
+ 3.80000    2.0534k
+ 4.20000    2.7852k
+ 4.60000    2.5k
+ 5.0    2.3k
.ends nmos
```

The preceding example is a voltage-versus-resistance table. It shows, for example, that the resistance at 5 V is 2.3 kohms.

Creating a Behavioral Inverter Lookup Table

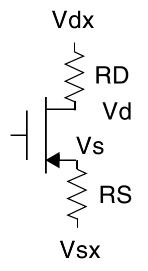
You can create an inverter lookup table in three simple steps:

1. Simulate an actual transistor level inverter, using a DC sweep of the input.
2. Print the V/I output, for the output pullup and pulldown transistors.
3. Copy the printed output into the volt lookup table element, for the controlled resistor.

The following test file, inv_vin_vout.sp, calculates `RN` (the effective pulldown resistor transfer function) and `RP` (the pullup transfer function).

- `RN` is calculated as Vout/I(mn), where mn is the pulldown transistor.
- `RP` is calculated as (VCC-Vout)/I(mp), where mp is the pullup transfer function.

The actual calculation uses a more accurate method, to obtain the series resistance of the transistor, as in Figure 61.



Rtot= (Vds-Vsx)/Ids

For greater accuracy:

Rtot= RD + RS + (vd-vs)/Ids
RD = 1/LV16(mn)
RS = 1/LV17(mn)
(vd-vs) = LX3(mn)
Ids = LX4(mn)

Figure 61 VIN versus VOUT

The first graph in Figure 62 shows `VIN` versus `VOUT`.

The second graph shows the computed transfer resistances (`RP` and `RN`), as a function of `VIN`.

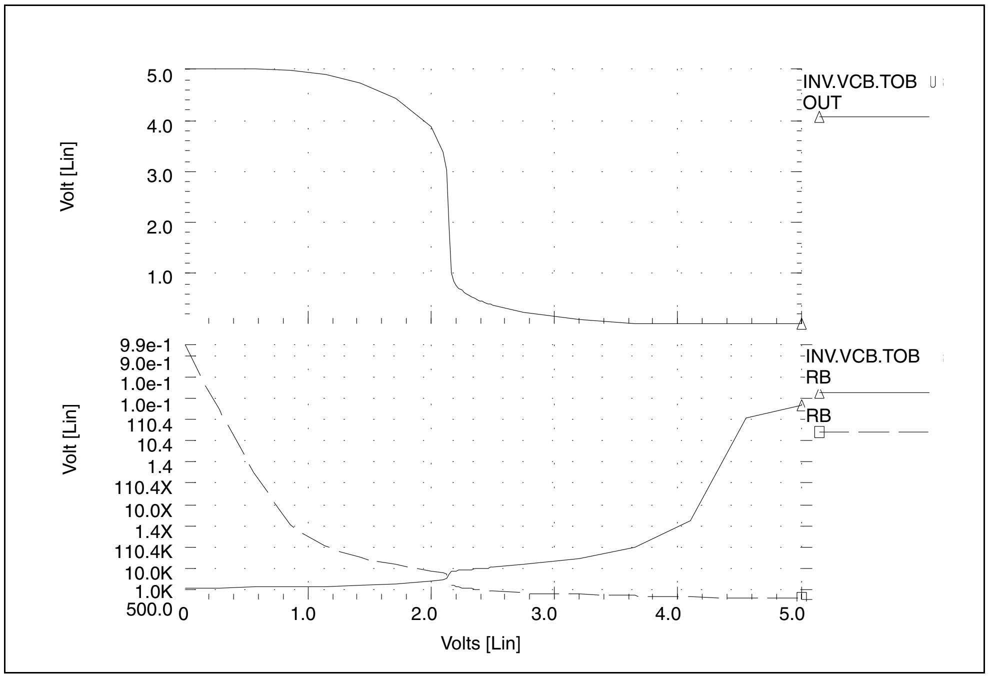


Figure 62 RP and RN as a Function of VIN

The HSPICE file used to calculate RP and RN is located in the following directory:

$installdir/demo/hspice/behave/inv_vin_vout.sp

Optimizing Behavioral CMOS Inverters

To calibrate behavioral models, run HSPICE on the full transistor version of a cell. Then optimize the behavioral model to this data.

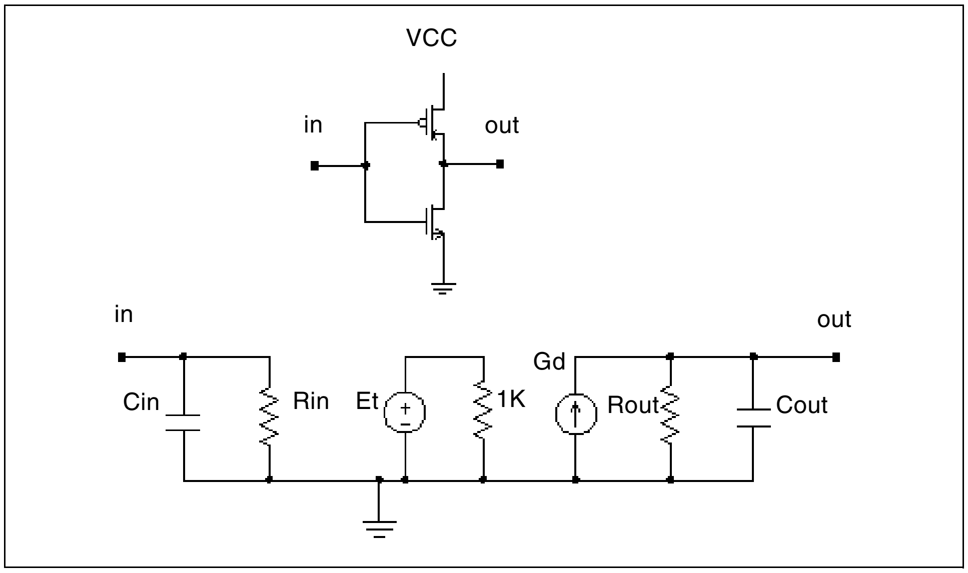


Figure 63 CMOS Inverter and its Equivalent Circuit

In this example, HSPICE uses the LEVEL 3 MOSFET model to simulate the CMOS inverter.

1. To obtain the input and output resistances, HSPICE performs a `.TF` transfer function analysis (`.TF V(out) Vin`).
2. To obtain the transfer function table of the inverter, HSPICE performs the DC analysis, and sweeps the input voltage (`.DC Vin 0 5 .1`).
3. HSPICE uses this table, in the PWL element, to represent the transfer function of the inverter.
4. A voltage-controlled PWL capacitance adjusts the rise and fall time of the inverter, in the equivalent circuit, across the output resistance.
5. The delay element obtains the propagation delay, across the output RC circuit.
6. HSPICE uses the inverter in a ring oscillator, to adjust the input capacitance.
7. HSPICE uses optimization analysis for all adjustments in this example. The data file and the results are shown.

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/invb_op.sp

The `invb_op.sp` file contains the following sections:

- Subcircuit Definition
- Inverter Using Model
- Optimization Results
- Optimization Completed
- Optimized Parameters OPTINV
- Optimize Results Measure Names and Values

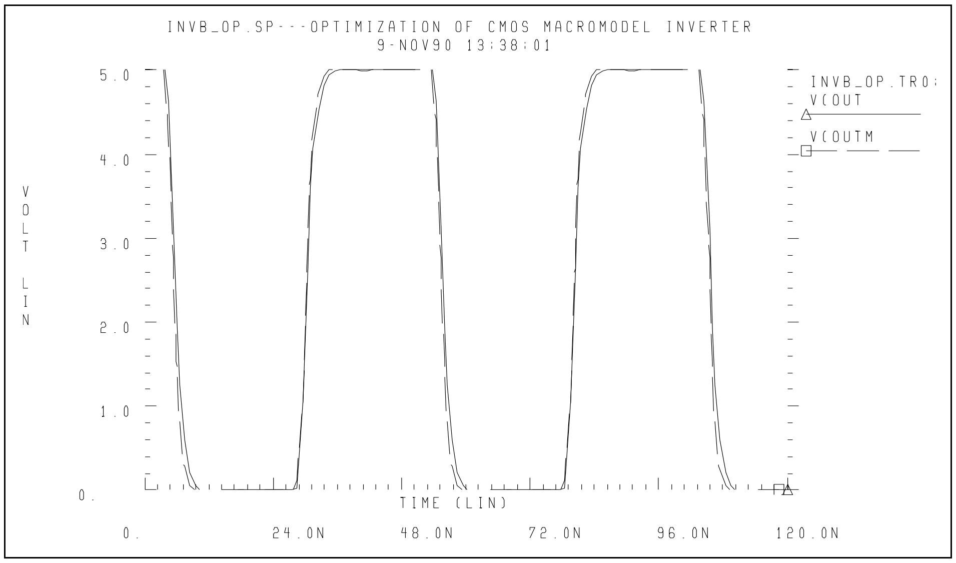


Figure 64 CMOS Inverter Response

Optimizing Behavioral Ring Oscillators

To optimize behavioral ring oscillator performance, review the examples in this section.

Example Five-Stage Ring Oscillator

This example is located in the following directory:

$installdir/demo/hspice/behave/ring5bm.sp

The `ring5bm.sp` file also contains the results of the five-stage ring oscillator example.

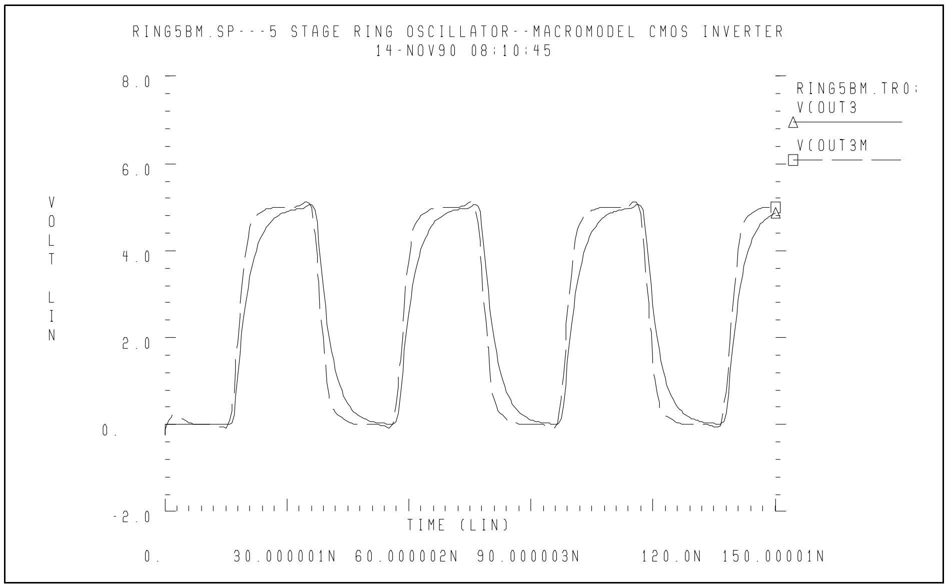


Figure 65   Ring Oscillator Response

Analog Behavioral Elements

The following components are examples of analog behavioral building blocks. Each component demonstrates a basic HSPICE feature:

- integrator: ideal op-amp E-element source
- differentiator: ideal op-amp E-element source
- ideal transformer: ideal transformer E-element source
- AM modulator: algebraic G-element source
- data sampler: algebraic E-element source

HSPICE uses an ideal op-amp to model the integrator circuit, and a VCVS to adjust output voltage. The following equation calculates output of the integrator:

$$Vout = -\frac{gain}{Ri \cdot Ci} \cdot \int_0^t Vin \cdot dt + Vout(0)$$

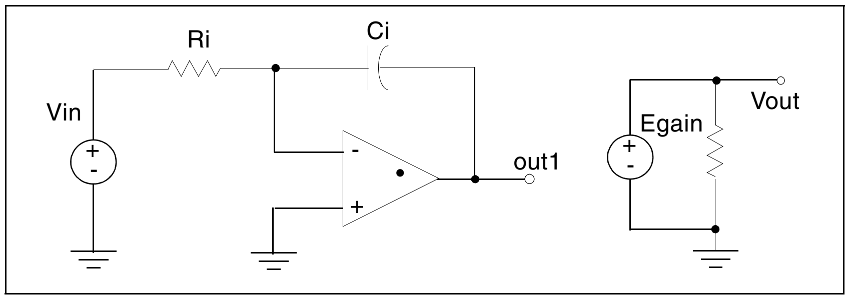


Figure 66 Integrator

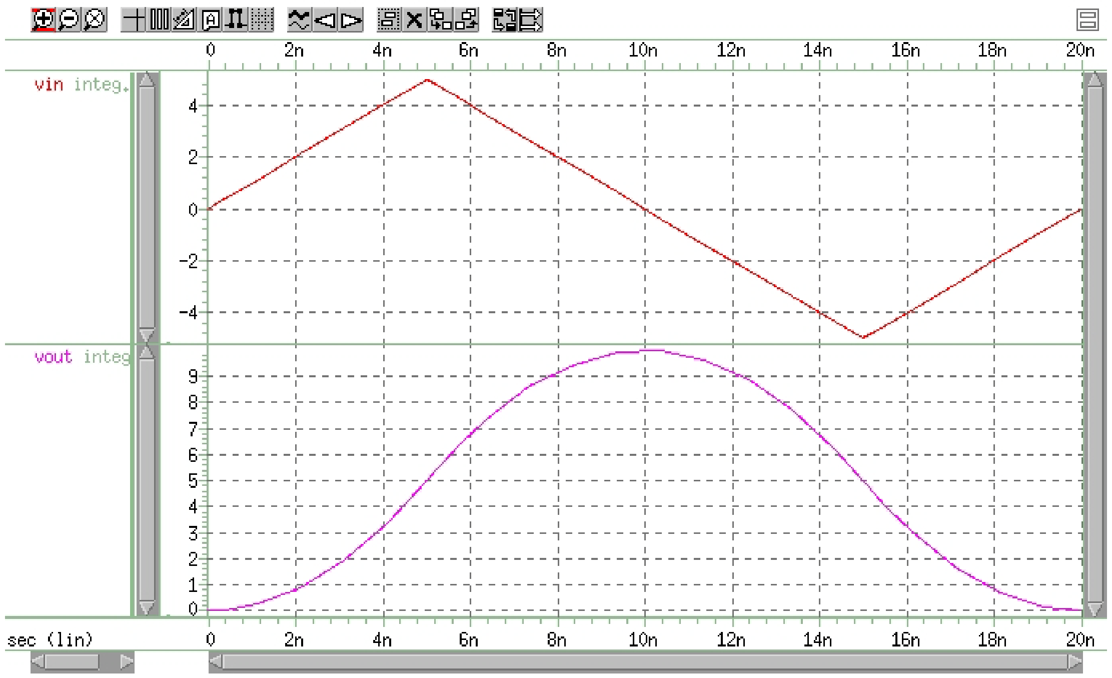


Figure 67 Response of Integrator to a Triangle Waveform

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/integ.sp

The `integ.sp` file also contains the following sections:

- Control and options
- Subcircuit definition
- Circuit

The following sections discuss these topics:

- Behavioral Differentiator
- Ideal Transformer
- Behavioral Amplitude Modulator
- Behavioral Data Sampler

Behavioral Differentiator

HSPICE uses an ideal op-amp to model a differentiator, and a VCVS to adjust the magnitude and polarity of the output. The following equation calculates the differentiator response:

$$Vout = -gain \cdot Rd \cdot Cd \cdot \frac{d}{dt}Vin$$

For a high-frequency signal, the output of a differentiator can overshoot the edges. To smooth this out, you can use a simple RC filter.

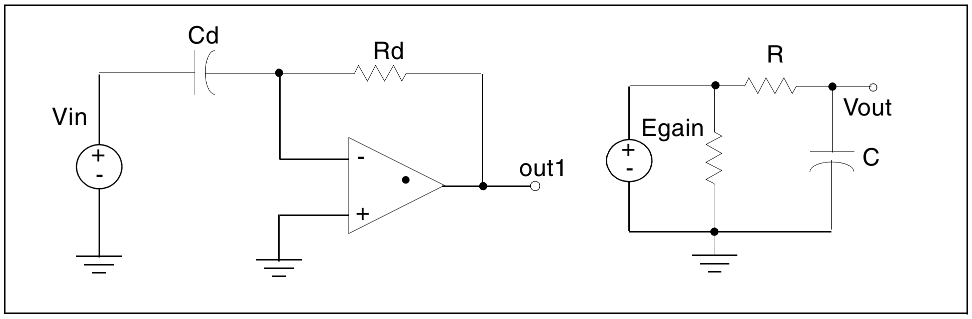


Figure 68 Differentiator

Example

This example is located in the following directory:

$installdir/demo/hspice/behave/diff.sp

The `diff.sp` file also contains the following sections:

- Control and Options
- Subcircuit Definition
- Circuit

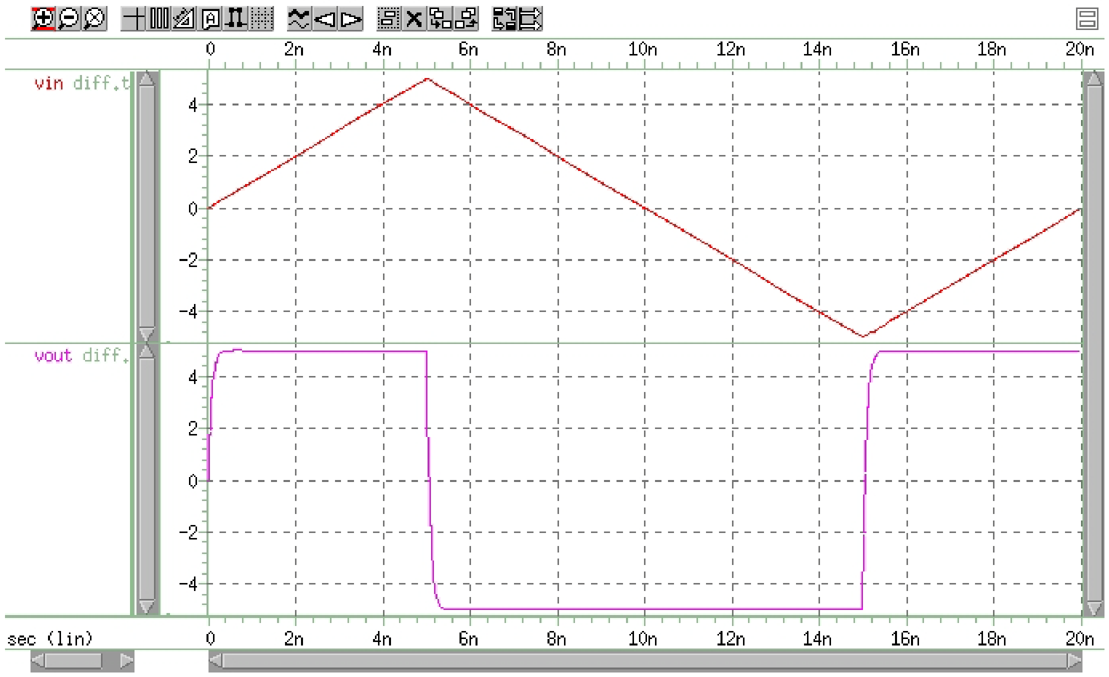


Figure 69 Response of a Differentiator to a Triangle Waveform

Ideal Transformer

The following example uses the ideal transformer to convert 8-ohms impedance of a loudspeaker, to 800 ohms impedance. This is a proper load value for a power amplifier, $Rin = n^2 \bullet RL$.

```
MATCHING IMPEDANCE BY USING IDEAL TRANSFORMER
E OUT 0 TRANSFORMER IN 0 10
RL OUT 0 8
VIN IN 0 1
.OP
.END
```

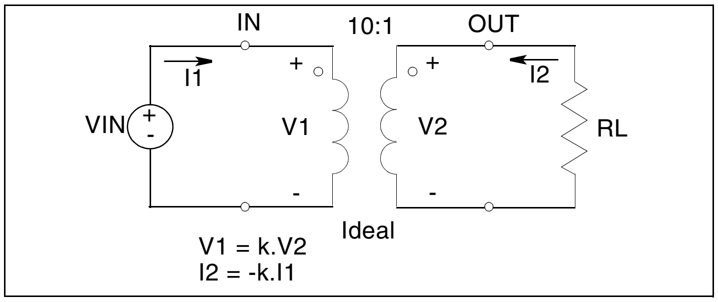


Figure 70 Ideal Transformer Example

Behavioral Amplitude Modulator

This example, which uses a G-element as an amplitude modulator with a pulse waveform carrier, is located in the following directory:

$*installdir*/demo/hspice/behave/amp_mod.sp

See also AM Modulation in *HSPICE User Guide: Basic Simulation and Analysis*.

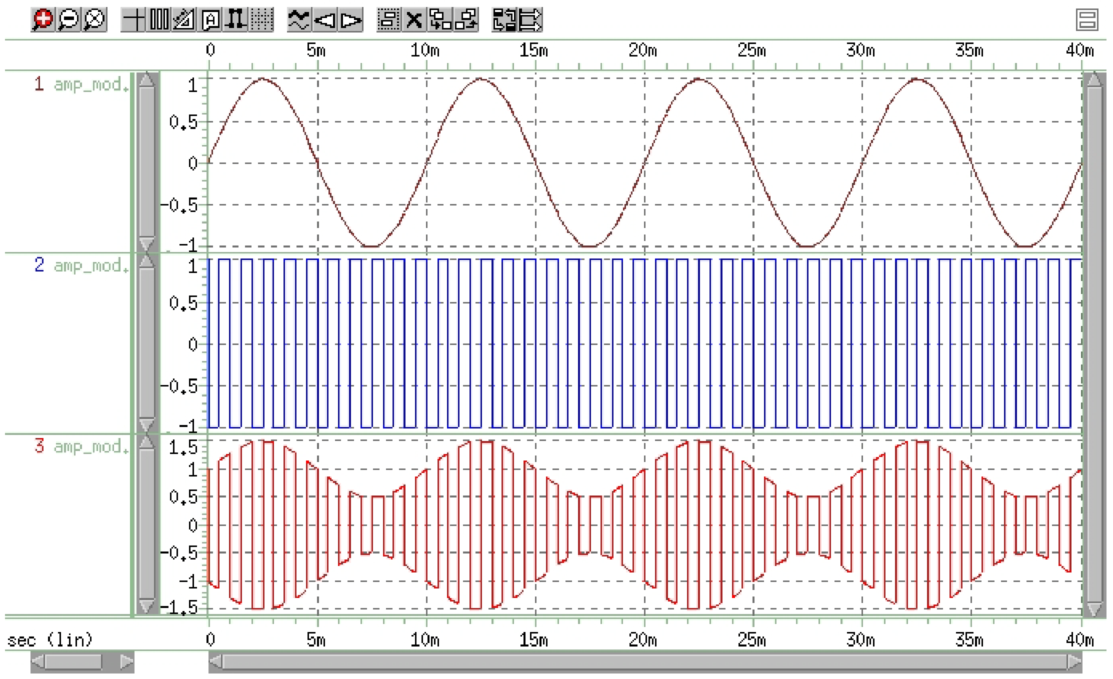


Figure 71 Amplitude Modulator Waveforms

Behavioral Data Sampler

An example of sampling behavioral data is located in the following directory:

$installdir/demo/hspice/behave/sampling.sp:

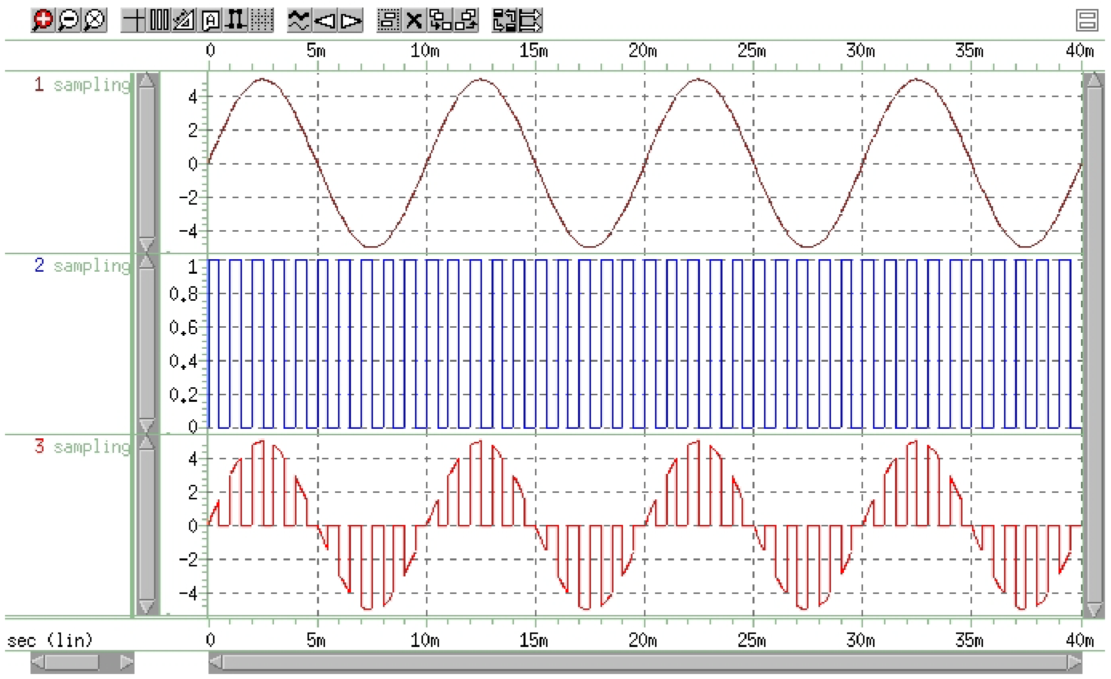


Figure 72 Sampled Data

Op-Amps, Comparators, and Oscillators

- 741 Op-Amp from Controlled Sources
- Inverting Comparator with Hysteresis
- Voltage-Controlled Oscillator (VCO)
- LC Oscillator

741 Op-Amp from Controlled Sources

To model the μA741 operational amplifier, use PWL controlled sources. A piecewise linear CCVS (source “h”) limits the output to ±15 V.

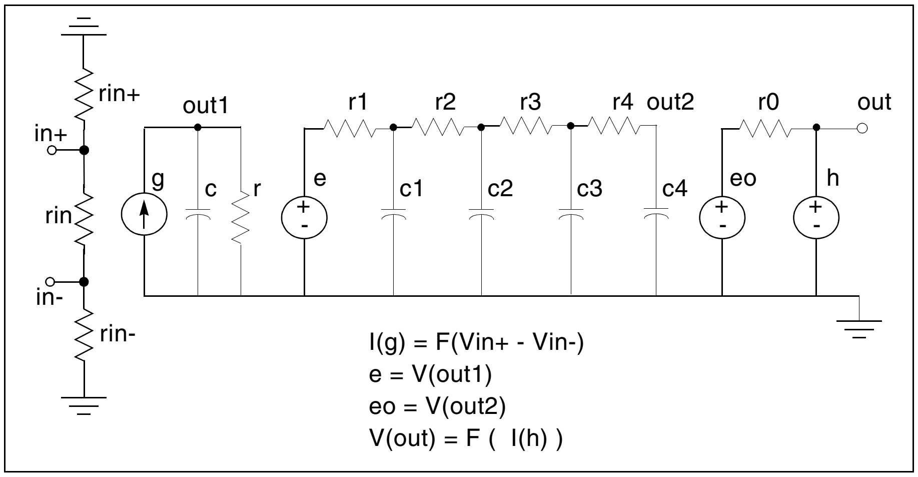


Figure 73 Operational Amplifier

Example

This example is located in the following directory:

$*installdir*/demo/hspice/behave/op_amp.sp

The `op_amp.sp` file also contains the following sections:

- Main Circuit
- RC Circuit With Pole At 9 MHz
- Output Limiter to 15 V

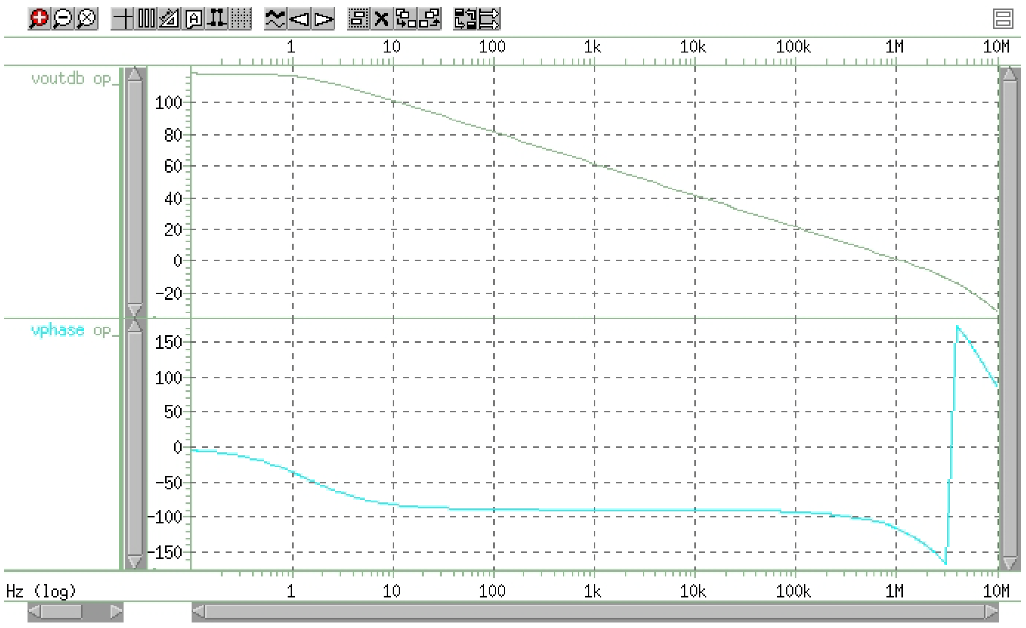


Figure 74   AC Analysis Response

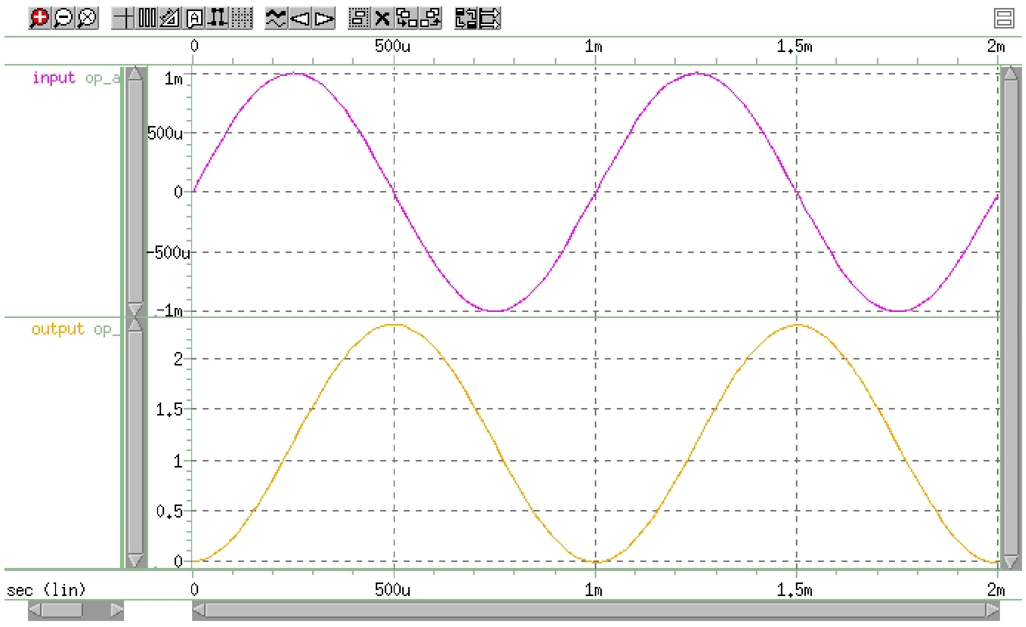


Figure 75   Transient Analysis Response

Inverting Comparator with Hysteresis

A piecewise-linear VCVS models an inverting comparator.

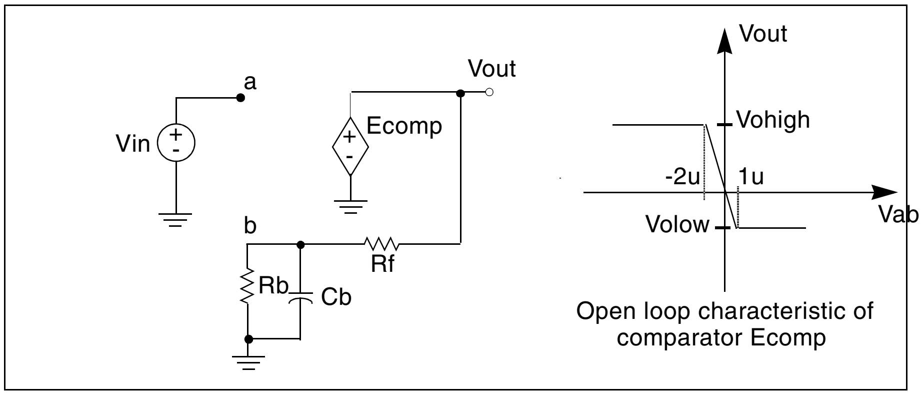


Figure 76   Inverting Comparator with Hysteresis

Two reference voltages correspond to the volow and vohigh voltages of Ecomp:

$$Vreflow = \frac{Volow \cdot Rb}{Rb + Rf} \qquad Vrefhigh = \frac{Vohigh \cdot Rb}{Rb + Rf}$$

When Vin exceeds Vrefhigh, the Vout output changes to Volow. For Vin values less than Vreflow, the output changes to Vohigh.

An example is located in the following directory:

$installdir/demo/hspice/behave/compar.sp

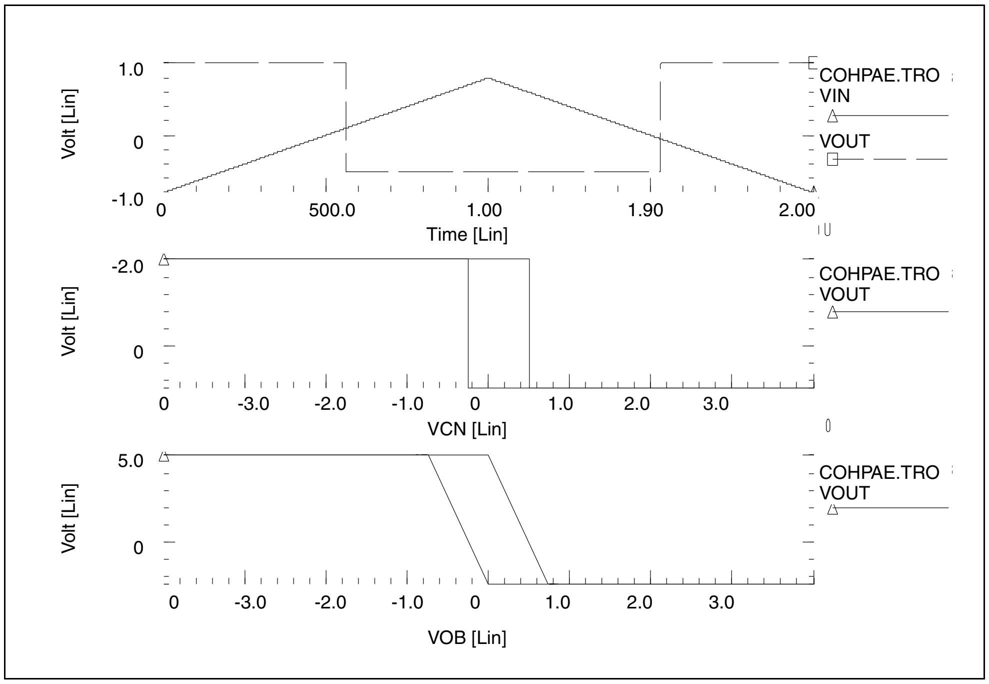


Figure 77 Response of Comparator

Voltage-Controlled Oscillator (VCO)

In this example, a one-input NAND (functioning as an inverter) models a five-stage ring oscillator. PWL capacitance switches the load capacitance of this inverter from 1pF to 3 pF. As the simulation results indicate, the oscillation frequency decreases, as the load capacitance increases.

Example

This example is based on demonstration netlist vcob.sp, which is available in directory $<installdir>/demo/hspice/behave. This file also contains a sample subcircuit definition.

```
vcob.sp voltage controlled oscillator using pwl functions
.option post
.global ctrl
.tran 1n 100n
.ic v(in)=0 v(out1)=5
.probe tran v(in)
x1 in out1 inv
x2 out1 out2 inv
x3 out2 out3 inv
x4 out3 out4 inv
x5 out4 in inv
vctrl ctrl 0 pwl(0,0 35n,0 40n,5)
*
* macro definitions
*
.subckt inv in out rout=1k
gcout out 0 pwl(1) ctrl 0 level=2 delta=.01
+ 4.5 1p
+ 4.6 3p
rout out 0 rout
gn 0 out nand(1) in 0 scale='1.0k/rout'
+ 0. 5.00ma
+ 0.25 4.95ma
+ 0.5 4.85ma
+ 1.0 4.75ma
+ 1.5 4.42ma
+ 3.5 1.00ma
+ 4.000 0.50ma
+ 4.5 0.20ma
+ 5.0 0.05ma
.ends inv
*

.end
```

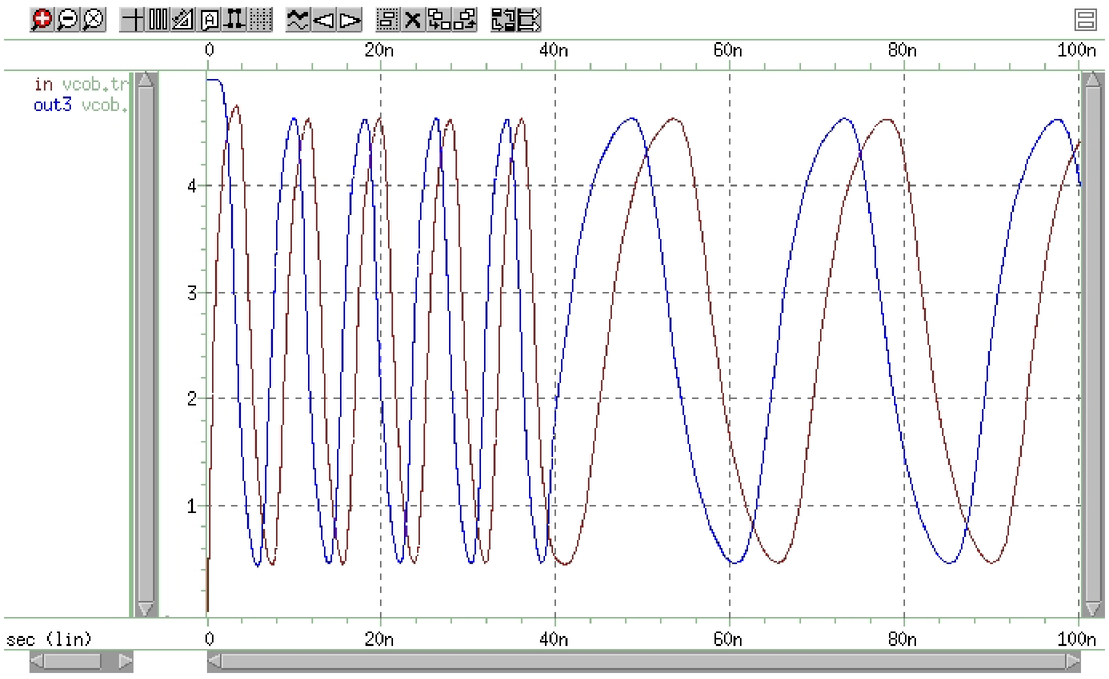


Figure 78 Voltage Controlled Oscillator Response

LC Oscillator

The initial capacitor charge is 5 V. The value of capacitance is the function of voltage, at node 10. The capacitance value becomes four times higher, at the t2 time. The following equation calculates the frequency of this LC circuit:

$$freq = \frac{1}{6.28 \cdot \sqrt{L \cdot C}}$$

At the t2 time, the frequency must be halved. The amplitude of oscillation depends on the condition of the circuit, when the capacitance value changes.

The stored energy is:

$$E = (0.5 \cdot C \cdot V^2) + (0.5 \cdot L \cdot I^2)$$

$$E = 0.5 \cdot C \cdot Vm^2, I = 0 \qquad E = 0.5 \cdot L \cdot Im^2, V = 0$$

At the t2 time, when V=0, if C changes to A ·C, then:

$$0.5 \cdot L \cdot Im^2 = 0.5 \cdot Vm^2 = 0.5 \cdot (A \cdot C) \cdot Vm'^2$$

and from the above equation:

$$Vm' = \frac{Vm}{\sqrt{A}}$$

$$Qm' = \sqrt{A} \cdot Vm$$

The second condition that HSPICE considers is when V=Vin, if C changes to A·C, then:

$$Qm = Qm'$$

$$C \cdot Vm = A \cdot C \cdot Vm'$$

$$Vm' = \frac{Vm}{A}$$

Therefore, HSPICE modifies the voltage amplitude, between Vm/sqrt(A) and Vm/A, depending on the circuit condition when the circuit switches. This example tests the `CTYPE=0` and `1` results. The result for `CTYPE=1` must be correct because capacitance is a function of voltage at node 10, not a function of the voltage across the capacitor itself.

Example

This example is based on demonstration netlist *calg2.sp*, which is available in directory $*installdir*/demo/hspice/behave:

```
* in this example the ctype 0 and 1 is tested. the result for
* ctype=1 must be correct because capacitance is function of
* voltage at node 10, not voltage across itself.
*
.option post
.ic v(1)=5 v(2)=5
c1 1 0 c='1e-9*v(10)' ctype=1
l1 1 0 1m
*
c2 2 0 c='1e-9*v(10)' ctype=0
l2 2 0 1m
*
v10 10 0 pwl(0sec,1v t1,1v t2,4v)
r10 10 0 1
```

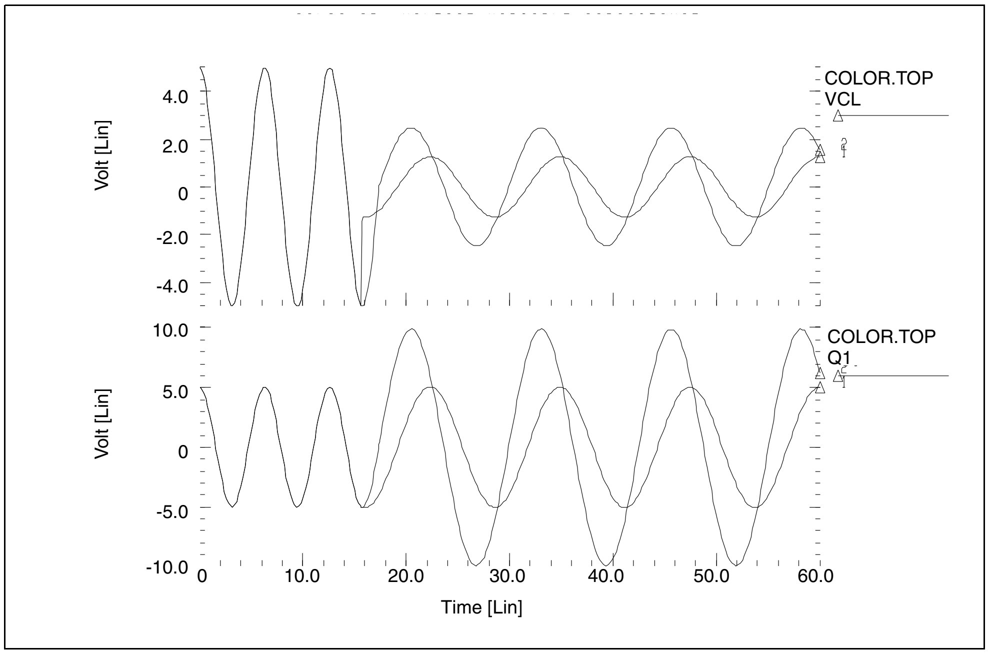


Figure 79   Correct Result Corresponding to CTYPE=1

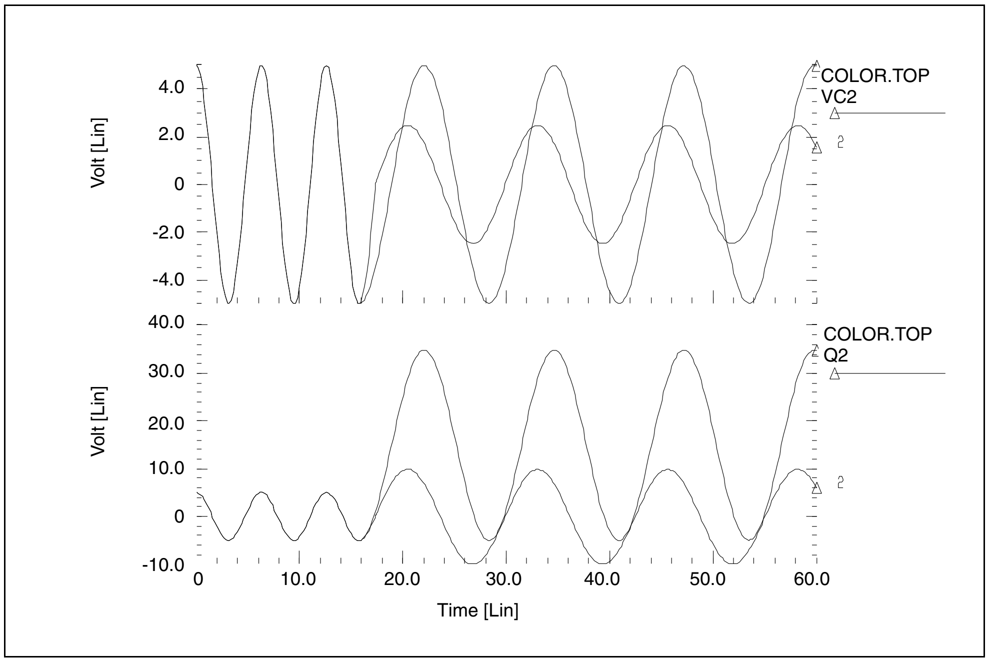


Figure 80   Incorrect Result Corresponding to CTYPE=0

Phase-Locked Loops (PLL)

The following sections explain material having to do with phase-locked loops.

These are the topics discussed:

- Phase Detector, with Multi-Input NAND Gates
- Phase Locked Loop Modeling

Phase Detector, with Multi-Input NAND Gates

This circuit uses behavioral elements, to implement the inverters, with 2, 3, and 4 input NAND gates.

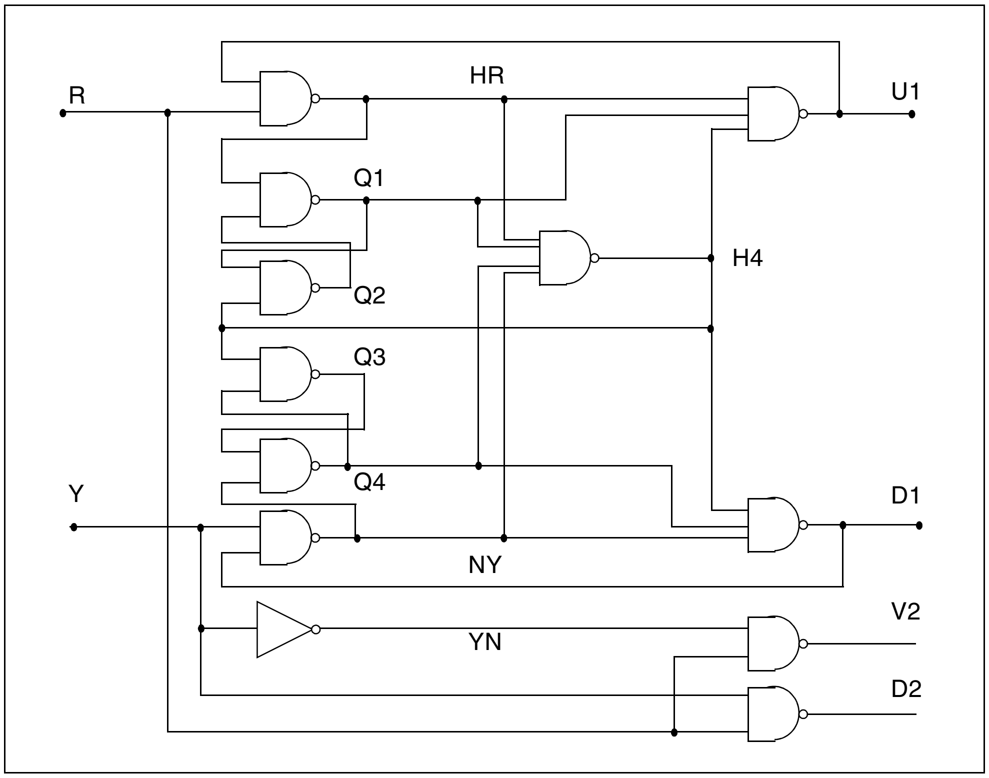


Figure 81 Phase Detector

Example

An example is located in the following directory:

$installdir/demo/hspice/behave/pdb.sp

This file also contains sample subcircuit definitions.

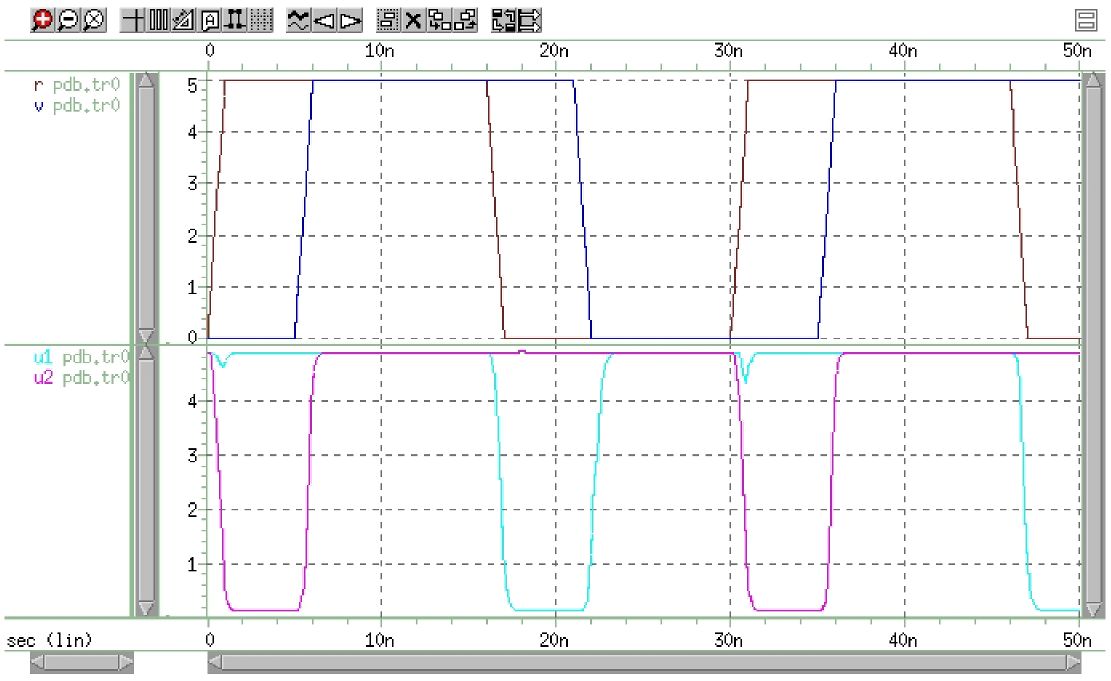


Figure 82 Phase Detector Response

Phase Locked Loop Modeling

A Phase-locked Loop (PLL) circuit synchronizes to an input waveform, within a selected frequency range. This returns an output voltage that is proportional to variations in the input frequency. It has three basic components:

- A voltage-controlled oscillator (VCO), which returns an output waveform that is proportional to its input voltage.
- A phase detector, which compares the VCO output to the input waveform, and returns an output voltage, depending on their phase difference.
- A loop filter, which filters phase detector voltage. Returns output voltage, which forms the VCO input (and external voltage output) of the PLL.

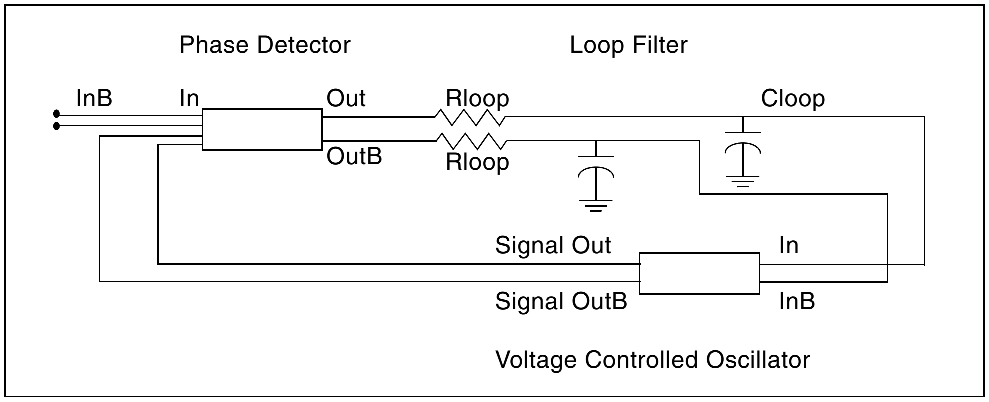


Figure 83 Behavioral Phase-Locked Loop

The PLL can be implemented using behavioral elements (Figure 83) or using bipolar transistors (Figure 84 on page 313 and Figure 85 on page 314).

The netlist for the behavioral PLL is the *pll_bvp.sp* file and the netlist for the full bipolar PLL is *pll.sp* file. The netlist for the full bipolar PLL contains the loop filter and the output circuit. Both netlist files are available in directory: $*installdir*/demo/hspice/behave.

The PLL transfer function shows a linear region of voltage vs. (periodic) time which is defined as the "lock" range.

The results of transient simulations (Figure 84) show minimal difference between implementations. However, run time statistics show that the behavioral model reduces simulation time, to one-third that of the full circuit.

If you use this PLL in a larger system simulation (for example, an AM tracking system), include the behavioral model. This model substantially reduces simulation run time, and still accurately represents the subcircuit.

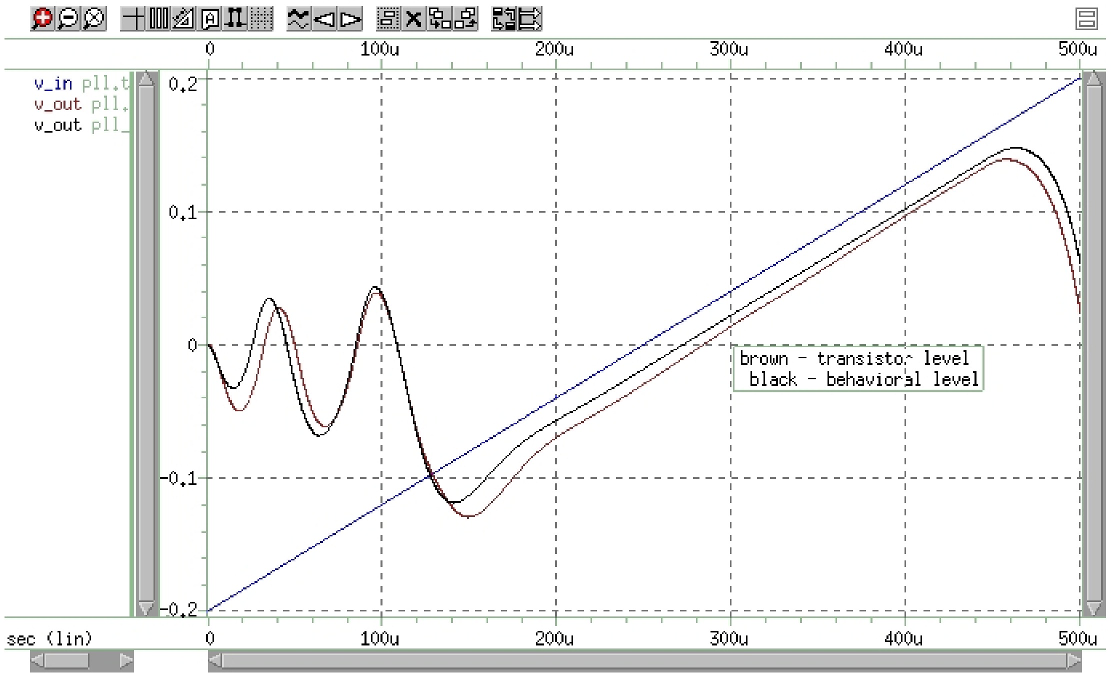


Figure 84 Behavioral (PLL_BVP Curve) vs. Bipolar (PLL_Curve) Simulation

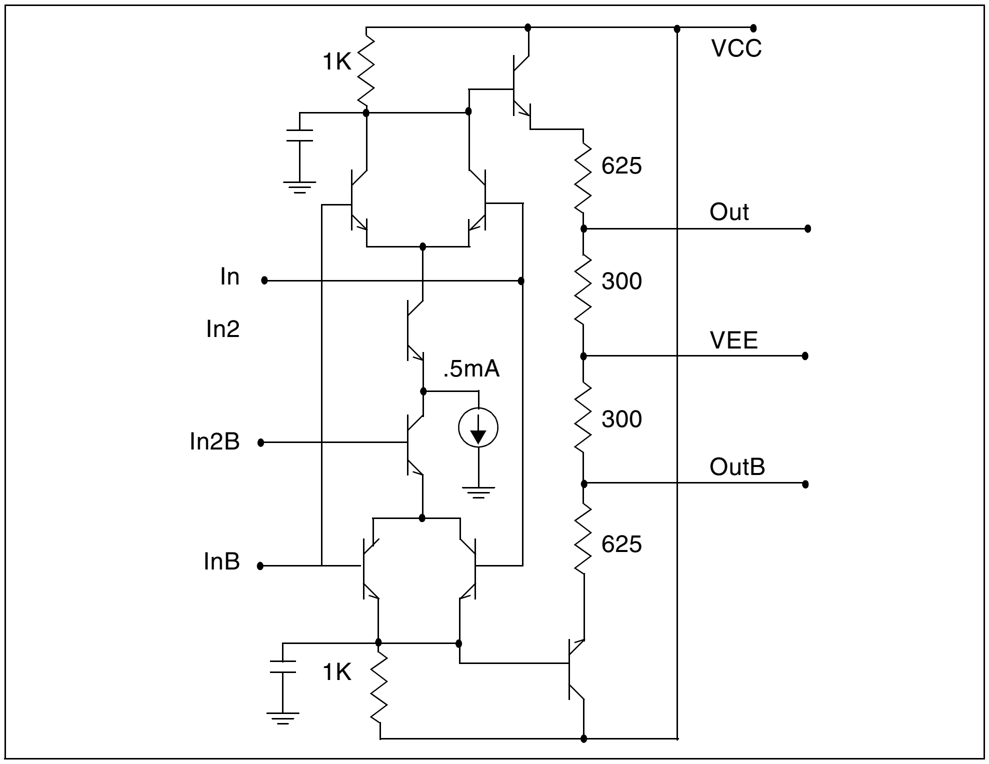


Figure 85 Bipolar Phase Detector

References

[1] Chua & Lin. *Computer Aided Analysis of Electronic Circuits*. Englewood Cliffs: Prentice-Hall, 1975, page 117. See also “SPICE2 Application Notes for Dependent Sources,” by Bert Epler, *IEEE Circuits & Devices Magazine*, September 1987.

12

Modeling Filters and Networks

Describes modeling filters and networks, including Laplace transforms.

When you apply Kirchhoff's laws to circuits that contain energy storage elements, the result is simultaneous differential equations, in the time domain. A simulator must solve these equations, to analyze the circuit's behavior. Solving any equation that is higher than first order can be difficult, and classical methods cannot easily solve some driving functions.

In both cases, to simplify the solution, you can use Laplace transforms. These transforms convert time domain equations, containing integral and differential terms, into algebraic equations in the frequency domain.

HSPICE ships numerous examples for your use; see Filters Examples in *HSPICE User Guide: Basic Simulation and Analysis* for paths to demo files.

The following sections discuss these topics:

- Transient Modeling
- Using G- and E-elements
- Laplace and Pole-Zero Modeling
- Modeling Switched Capacitor Filters
- References

Transient Modeling

The Laplace transform method provides an easy way to relate a circuit's behavior, in time and frequency-domains. This facilitates simultaneous work in those domains.

The algorithm that Synopsys HSPICE uses for Laplace and pole/zero transient modeling, offers better performance than the Fast Fourier Transform (FFT) algorithm. To invoke Laplace and pole/zero transient modeling, use a `LAPLACE` or `POLE` function call in a source element statement.

Laplace transfer functions are especially useful in top-down system design, when you use ideal transfer functions instead of detailed circuit designs. In HSPICE, you can also mix Laplace transfer functions, with transistors and passive components. Using this capability, you can model a system as the sum of the contributing ideal transfer functions. You can then progressively replace these functions with detailed circuit models, as they become available. Conventional uses of Laplace transfer functions include control systems, and behavioral models that contain non-linear elements.

Laplace transforms reduce the time needed to design and simulate large interconnect systems, such as clock distribution networks. You can use asymptotic waveform evaluation (AWE) and other methods, to create a Laplace transfer function model. The AWE model can use only a few poles to represent the large circuit. You can input these poles through a Laplace transform model, to closely approximate the delay and overshoot characteristics of many networks, in a fraction of the original simulation time.

You can use pole/zero analysis to help determine the stability of the design. You can use the `POLE` function in HSPICE when the poles and zeros of the circuit are specified, or you can use the `.PZ` statement (see .PZ in *HSPICE Reference Manual: Commands and Control Options*) to derive the poles and zeros from the transfer function.

Frequency response is an important analog circuit property. It is normally the ratio of two complex polynomials (functions of complex frequencies), with positive real coefficients. The form of frequency response can be either the locations of poles and zeros, or a frequency table.

The usual way to design complex circuits is to interconnect smaller functional blocks of known frequency responses, either in pole/zero or frequency table form. For example, to design a band-reject filter, you can interconnect a low-pass filter, a high-pass filter, and an adder. Study the function of the complex circuit, in terms of its component blocks, before you design the actual circuit. After you test the functionality of the component blocks, you can use these blocks as a reference in optimization techniques, to determine the value of the complex element.

Using G- and E-elements

- Laplace Transform Function Call
- Element Statement Parameters
- Z Transform Function Call
- G- and E-element Notes
- Laplace Band-Reject Filter
- Laplace Low-Pass Filter
- Circular Convolution Example

Laplace Transform Function Call

Use the G- and E-elements as linear functional blocks, or as elements with specific frequency responses.

In the following equations, *H*(s) denotes the frequency response (also called the impulse response), where s is a complex frequency variable ($s = j2\pi f$). To obtain the frequency response, perform an AC analysis, and set `AC=1` in the input source (the Laplace transform of an impulse is 1). The following expression relates the input and output of the G- and E-elements, with specified frequency response:

$$Y(j2\pi f) = Hj(2\pi f) \cdot Xj(2\pi f)$$

where X is the input, Y is the output, and H is the transfer function, at the f frequency.

AC analysis uses the above relation, at any frequency, to determine the frequency response. For operating point and DC sweep analysis, the relation is the same, but the frequency is zero.

The transient analysis is more complicated than the frequency response. The output is a convolution of the input waveform, with the impulse response h(t):

$$y(t) = \int_{-\infty}^{t} x(\tau) \cdot h(t-\tau) \cdot d\tau$$

In discrete form, the output is:

$$y(k\Delta) = \Delta \sum_{m=0}^{k} x(m\Delta) \cdot h[(k-m) \cdot \Delta]\text{, k = 0, 1, 2, ...}$$

where you can obtain h(t) from H(f), using the inverse Fourier integral:

$$h(t) = \int_{-\infty}^{\infty} H(f) \cdot e^{j2\pi ft} \cdot df$$

The following equation calculates the inverse discrete Fourier transform:

$$h(m\Delta) = \frac{1}{N \cdot \Delta} \sum_{n=0}^{N-1} H(f_n) \cdot e^{\frac{j2\pi nm}{N}}\text{, m = 0, 1, 2, ..., N-1}$$

where N is the number of equally-spaced time points, and Δ is the time interval or time resolution.

For the frequency response table form (FREQ) of the `LAPLACE` function, HSPICE uses a performance-enhanced algorithm, to convert H(f) to h(t). This algorithm requires N to be a power of 2. The following equation determines the f_n frequency point:

$$f_n = \frac{n}{N \cdot \Delta}, \quad \text{n = 0, 1, 2, ..., N-1}$$

where n > N/2 represents the negative frequencies. The following equation determines the Nyquist critical frequency:

$$f_c = f_{N/2} = \frac{1}{2 \cdot \Delta}$$

Because the negative frequency responses are the image of the positive responses, you need to specify only N/2 frequency points, to evaluate N time points of h(t). The larger the value of f_c is, the more accurate the transient analysis results are. However, for large f_c values, the Δ becomes smaller, and computation time increases.

The maximum frequency of interest depends on the functionality of the linear network. For example, in a low-pass filter, you can set f_c to the frequency at which the response drops by 60 dB (a factor of 1000).

$$|H(f_c)| = \frac{|H_{max}|}{1000}$$

After you select or calculate f_c, the following equation can determine Δ:

$$\Delta = \frac{1}{2 \cdot f_c}$$

The following equation calculates the frequency resolution:

$$\Delta f = f_1 = \frac{1}{N \cdot \Delta}$$

which is inversely proportional to the maximum time (NΔ), over which HSPICE evaluates h(t). Therefore, the transient analysis accuracy also depends on the frequency resolution, or the number of points (N).

You can specify the frequency resolution (`DELF`) and the maximum frequency (`MAXF`) in the G- or E-element statement. To calculate N, HSPICE uses 2 ·`MAXF`/`DELF`. Next, HSPICE modifies N as a power of 2. The effective `DELF` is `2` ·MAXF/N, to reflect the changes in N.

Element Statement Parameters

These keywords are common to the three forms described above:

- Laplace
- Pole-zero
- Frequency response table

Table 8 Element Statement Parameters

Parameter	Description
ACCURACY	Used only with the frequency response table.
	0: Default. This method generates more accurate results and achieves better performance. 1: Provides more accurate results for frequency table forms, as compared to ACCURACY=0
DELF, DELTA	Frequency resolution Δf. The inverse of `DELF` is the time window, over which HSPICE calculates h(t) from H(s). A smaller DELF value means more accurate transient analysis, and longer CPU time. The number of points (N) used to convert H(s) to h(t) is N=2·MAXF/DELF. Because N must be a power of 2, HSPICE adjusts the DELF value. The default is 1/TSTOP. In the G-element, with FREQ and ACCURACY = 0 or 1, to perform circular convolution for periodic input, HSPICE limits the period. To do this, HSPICE sets DELF = 1/T:T < TSTOP.

Table 8 Element Statement Parameters (Continued)

Parameter	Description
FREQ	Keyword to indicate that a frequency response table describes the transfer function. Do not use FREQ as a node name in a G- or E-element. This is not the same as the `FREQ` model parameter that plots symbol frequency in `.PRINT`/`.PROBE` statements (or the deprecated `.PLOT`/`.GRAPH` statements.
LAPLACE	Keyword to indicate that a Laplace transform function describes the transfer function. Do not use LAPLACE as a node name, on a G- or E-element.
LEVEL	Used only in elements with a frequency-response table. Set this parameter to 1, if the element is a high-pass filter.
M	G-element multiplier. This parameter represents M G-elements in parallel. Default is 1.
MAXF, MAX	Maximum, or the Nyquist critical frequency. The larger the MAXF value, the more accurate the transient results are, and the longer the CPU time. The default is $1024 \cdot DELF$. These parameters apply only when you also use the FREQ parameter.
POLE	Keyword to indicate that the pole and zero location describes the transfer function. Do not use POLE as a node name, on a G- or E-element.
SCALE	Element value multiplier.
TC1,TC2	First-order and second-order temperature coefficients. The default is zero. The temperature updates the SCALE: $SCALEeff = SCALE \cdot (1 + TC1 \cdot \Delta t + TC2 \cdot \Delta t^2)$

Z Transform Function Call

Z transform functions are used for G- and E-elements (controlled

behavioral sources), which is similar to the Laplace form function (see Laplace Transform Function Call for more details).

H(z) denotes the frequency response, where z is a complex frequency variable. Its value is: $z = e^{(j \cdot w)}$

where, $w = 2 \cdot Pl \cdot f/fs$ and $fs = 2 \cdot \mathrm{MAXF}$
MAXF denotes the Nyquist critical frequency.

Z Transform Syntax

Transconductance H(z):

```
Gxxx n+ n- ZTRANS in+ in- k0, k1, ..., kn
+ / d0, d1, ..., dm
+ [MAXF=val][SCALE=] [TC1=val] [TC2=val] [M=val]
```

Voltage Gain H(z):

```
Exxx n+ n- ZTRANS in+ in- k0, k1, ..., kn
+ / d0, d1, ..., dm
+ [MAXF=val][SCALE=] [TC1=val] [TC2=val]
```

H(z) is a rational function, in the following form:

$$H(z) = \frac{k0 + k1 \cdot z^{-1} + k2 \cdot z^{-2} + ... + kn \cdot z^{-n}}{d0 + d1 \cdot z^{-1} + d2 \cdot z^{-2} + ... + dm \cdot z^{-m}}$$

You can use parameters to define the values of all coefficients

$(k0, k1, ..., d0, d1, ...)$

Example:

```
Glow_pass 0 out ZTRANS in 0 0.0317 0.0951
+ 0.0951 0.0317 / 1.0 -1.459 0.9104 -0.1978
Ehigh_pass out 0 ZTRANS in 0 -0.0082 -0.1793
+ 0.6579 -0.1793 -0.0082 / 1
```

The `Glow_pass` element statement describes a third-order low-pass filter, with the transfer function:

$$H(z) = \frac{0.0317 + 0.0951^{-1} + 0.0951^{-2} + 0.0317z^{-3}}{1.0 - 1.459z^{-1} + 0.9104z^{-(2)} + 0.1978z^{-3}}$$

The `Ehigh_pass` element statement describes a fourth-order highpass filter, with the transfer function:

$$H(z) = 0.0082 - 0.1793z^{-1} + 0.06579z^{-2} - 0.1793z^{-3}$$

G- and E-element Notes

- If elements in the data file specify frequency responses, do not use pole/ zero analysis. If you specify `MAXF=<`*par*`>` in a G- or E-element Statement, HSPICE warns that it is ignoring `MAXF`. This is normal.
- HSPICE performs circular convolution, when G-element `ACCURACY = 0` or `1` (see Figure 92 on page 329).

Laplace Band-Reject Filter

This example models an active band-reject filter, with 3-dB points at 100 and 400 Hz, and <35 dB of attenuation, between 175 and 225 Hz. The band-reject filter contains low-pass and high-pass filters, and an adder. The low-pass and high-pass filters are fifth-order Chebyshev, with 0.5-dB ripple.

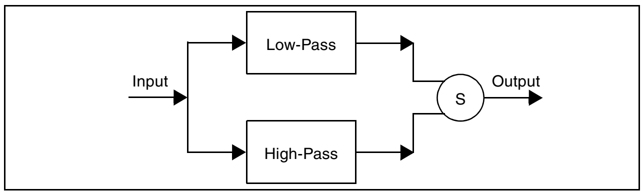


Figure 86 Band-Reject Filter

Example

This example is located in the following directory:
$*installdir*/demo/hspice/filters/BandstopL.sp

The BandstopL.sp file also contains a sample band-reject filter circuit.

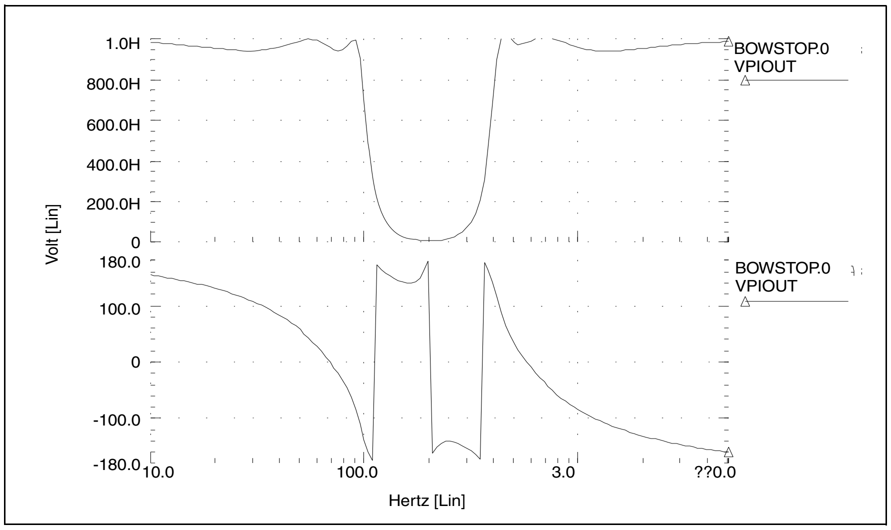


Figure 87 Frequency Response of the Band-Reject Filter

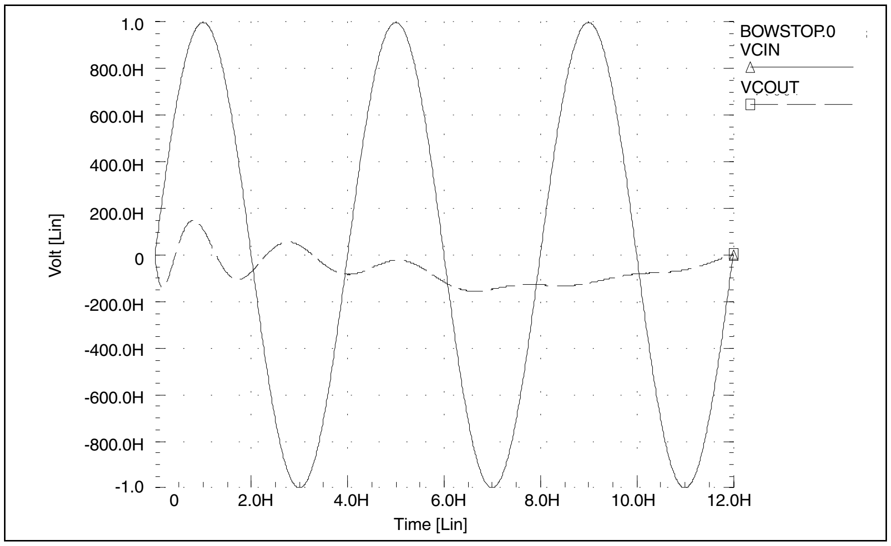


Figure 88 Transient Response of the Band-Reject Filter to a 250 Hz Sine Wave

Laplace Low-Pass Filter

This example simulates a third-order low-pass filter, with a Butterworth transfer function. It also compares the results of both the actual circuit and the functional G Element, with the third-order Butterworth transfer function, for AC and transient analysis.

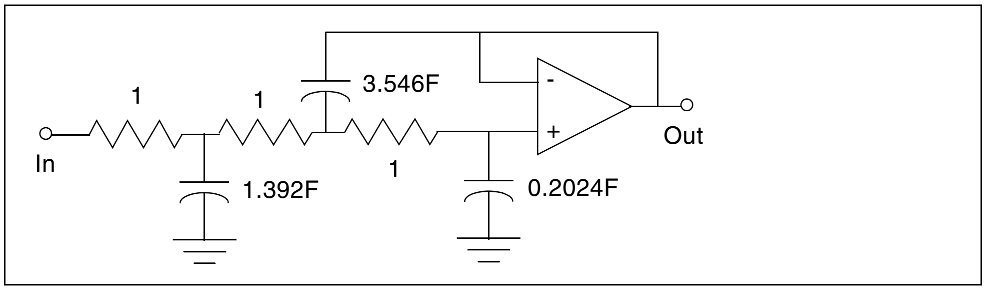


Figure 89 Third-Order Active Low-Pass Filter

The third-order Butterworth transfer function that describes the above circuit is:

$$H(s) = \frac{1.0}{1.0 \cdot (s+1)(s+0.5+j2\pi \cdot 0.1379)(s+0.5-(j2\pi \cdot 0.1379))}$$

The following example is the input listing for the above filter. Parameters set the pole locations for the G Element. Also, this listing specifies only one of the complex poles. The program derives the conjugate pole. The output of the circuit is the `out` node, and the output of the functional element is `outg`.

Example

An example of a third-order low-pass Butterworth filter is located in the following directory:
$installdir/demo/hspice/filters/Low_Pass.sp

The Low_Pass.sp file also contains a sample circuit description.

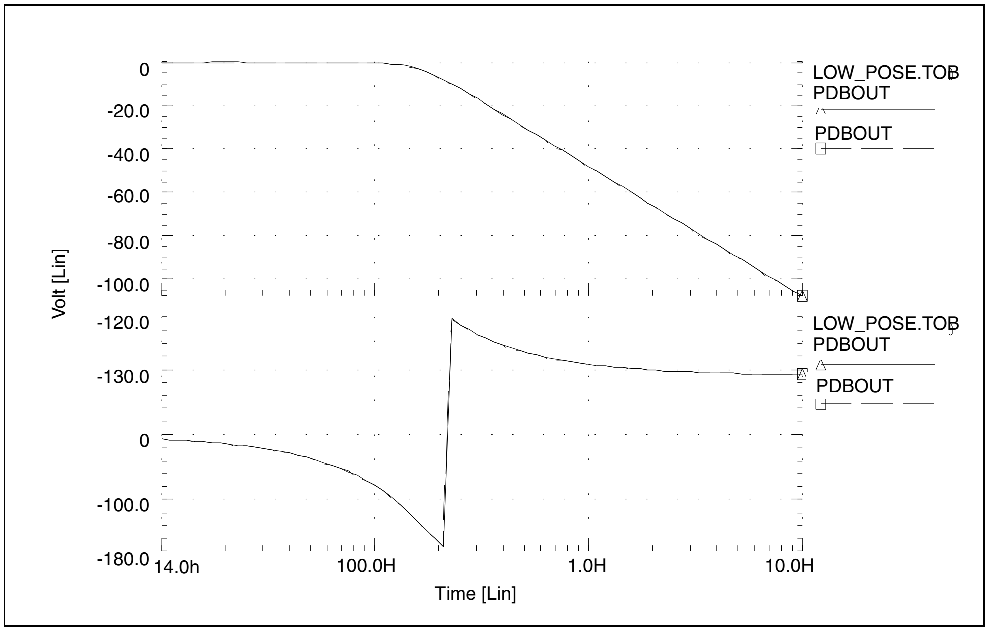


Figure 90 Frequency Response of Circuit and Functional Element

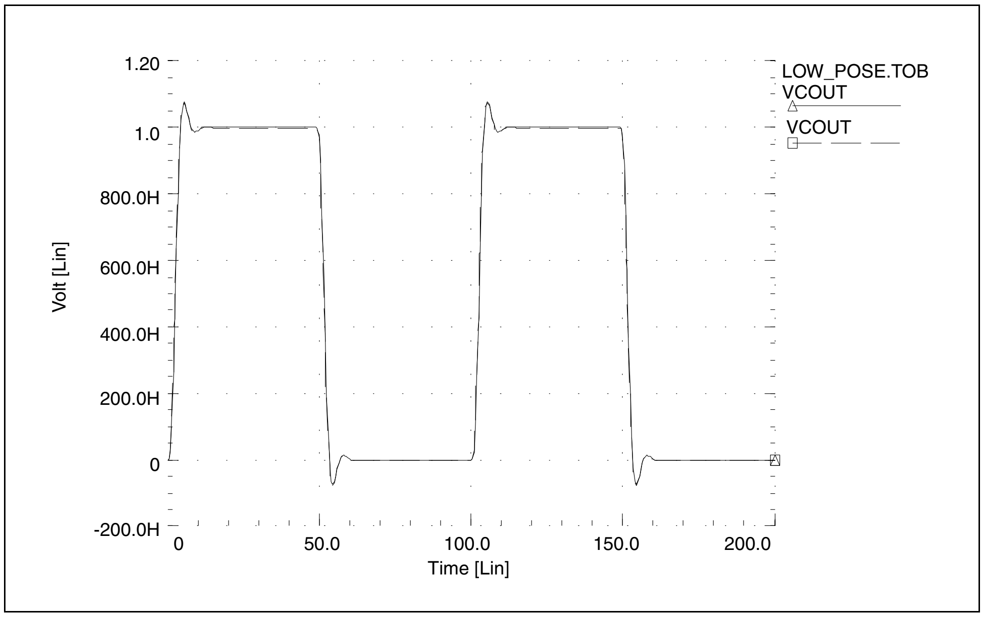


Figure 91 Transient Response of Circuit and Functional Element to a Pulse

Circular Convolution Example

This 30-degree phase-shift filter uses circular convolution. `If DELF=10 MHz`, HSPICE uses inverse fast Fourier transform (IFFT) to obtain the period of time domain response for the G-element. This value is based on the input frequency table, and is 100 ns. The `FREQ` G-element performs the convolution integral from t - T to t, assuming that all control voltages at t<0 are zero. t is the target time point, and T is the period of the time domain response, for the G-element.

In this example, during time points from 0 to 100 ns, HSPICE uses harmonic components higher than 10 MHz, due to the input transition at t=0. So the circuit does not behave as a phase shift filter. After one period (t>100 ns), HSPICE performs circular convolution, based on a period of 100 ns. The transient result represents a 30-degree phase shift, for continuous periodic control voltage.

Notes

- V(ctrl): control voltage input.
- V(expected): node. Represents an ideal 30-degree shifted wave for the input.
- V(test): output of the G Element.

30-Degree Phase Shift Circuit File

This example illustrates a 30-degree phase-shift filter. It is based on demonstration netlist phaseshift.sp, which is available in directory $<installdir>/demo/hspice/filters.

```
****
.tran 0.1n 300n
.OPTION post ingold=2 accurate
Vctrl ctrl gnd sin (0 1 10e6)
Gtest gnd test freq ctrl gnd
+ 1.0e00 0 30
+ 1.0e01 0 30
+ 1.0e02 0 30
+ 1.0e03 0 30
+ 1.0e04 0 30
+ 1.0e05 0 30
+ 1.0e06 0 30
+ 1.0e07 0 30
+ 1.0e08 0 30
+ 1.0e09 0 30
+ 1.0e10 0 30
+ MAXF=1.0e9 DELF=10e6
Rtest test gnd 1
Iexpected gnd 3 sin (0 1 10e6 0 0 30)
Vmes 3 expected 0v
Rexpected expected gnd 1
.end
```

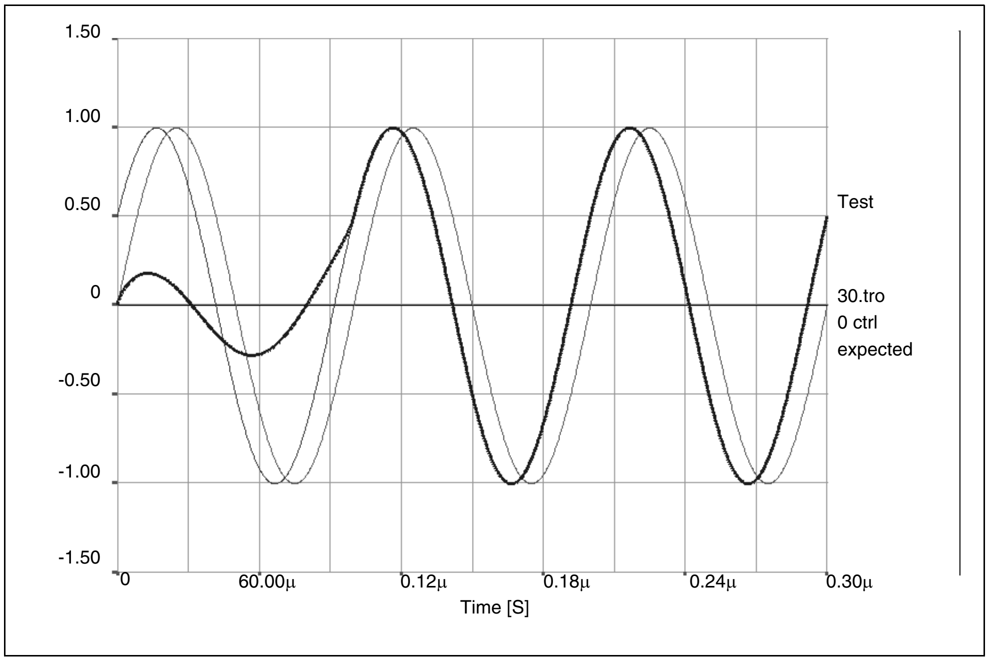


Figure 92 Transient Response of the 30 Degree Phase Shift Filter

Laplace and Pole-Zero Modeling

The following sections discuss these topics:

- Laplace Transform (LAPLACE) Function
- Laplace Transform POLE (Pole/Zero) Function
- AWE Transfer Function Modeling
- Y-parameter Line Modeling
- Comparison of Circuit and Pole/Zero Models

Laplace Transform (LAPLACE) Function

HSPICE provides two types of `LAPLACE` function calls: one for transconductance, and one for voltage gain transfer functions. See Using G- and E-elements on page 319 for the general forms, and Element Statement Parameters on page 321 for descriptions of the parameters.

General Form of the Transfer Function

To use `LAPLACE` modeling function, you must find the k_0, ..., k_n and d_0, ..., d_m coefficients of the transfer function. The transfer function is the s-domain (frequency domain) ratio, of the output for a single-source circuit, to the input, with initial conditions set to zero. The following equation represents the Laplace transfer function:

$$H(s) = \frac{Y(s)}{X(s)}$$

where:

- s is the complex frequency, $j2\pi f$.
- Y(s) is the Laplace transform of the output signal.
- X(s) is the Laplace transform of the input signal.

Note: To obtain the impulse response H(s), HSPICE performs AC analysis, where `AC=1` represents the input source. The Laplace transform of an impulse is 1. For an element with an infinite response at DC (such as a unit step function H(s)=1/

s), HSPICE calculations use the value of the `EPSMIN` option (the smallest number possible on the platform) as the transfer function.

The general form of the transfer function H(s) in the frequency domain, is:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

The order of the numerator for the transfer function cannot be greater than the order of the denominator. The exception is differentiators, for which the transfer function H(s) = ks. You can use parameter values for all k and d coefficients of the transfer function, in the circuit descriptions.

Finding the Transfer Function

The first step in determining the transfer function of a circuit is to convert the circuit to the s-domain. To do this, transform the value for each element, into its s-domain equivalent form.

Table 9 on page 331 and Table 10 on page 332 show transforms that convert some common functions to the s-domain. The next section provides examples of using transforms, to determine transfer functions.

Table 9 Laplace Transforms for Common Source Functions

f(t), t>0	**Source Type**	**L{f(t)}= F(s)**
σ(t)	impulse	1
u(t)	step	$\frac{1}{s}$
t	ramp	$\frac{1}{s^2}$
e-at	exponential	$\frac{1}{s+a}$
sin ωt	sine	$\frac{w}{s^2 + w^2}$

Table 9 Laplace Transforms for Common Source Functions (Continued)

f(t), t>0	Source Type	L{f(t)}= F(s)
cos ωt	cosine	$\frac{s}{s^2 + w^2}$
$\sin(\omega t + \theta)$	sine	$\frac{s\sin(\theta) + \omega\cos(\theta)}{s^2 + \omega^2}$
$\cos(\omega t + \theta)$	cosine	$\frac{s\cos(\theta) - \omega\sin(\theta)}{s^2 + \omega^2}$
$\sinh \omega t$	hyperbolic sine	$\frac{\omega}{s^2 - \omega^2}$
$\cosh \omega t$	hyperbolic cosine	$\frac{s}{s^2 - \omega^2}$
te-at	damped ramp	$\frac{1}{(s+a)^2}$
$e^{-at} \sin \omega t$	damped sine	$\frac{\omega}{(s+a)^2 + \omega^2}$
$e^{-at} \cos \omega t$	damped cosine	$\frac{s+a}{(s+a)^2 + \omega^2}$

Table 10 Laplace Transforms for Common Operations

f(t)	L{f(t)} = F(s)
$Kf(t)$	$KF(s)$

Table 10 Laplace Transforms for Common Operations (Continued)

f(t)	L{f(t)} = F(s)
$f_1(t) + f_2(t) - f_3(t) + \ldots$	$F_1(s) + F_2(s) - F_3(s) + \ldots$
$\frac{d}{dt}f(t)$	$sF(s) - f(0- \quad)$
$\frac{d^2}{dt^2}f(t)$	$s^2F(s) - sf(0) - \frac{d}{dt}f(0- \quad)$
$\frac{d^n}{dt^n}f(t)$	$s^nF(s) - s^{n-1}f(0) - s^{n-2}\frac{d}{dt}f(0)$
$\int_{-\infty}^{t} f(t)dt$	$\frac{F(s)}{s} + \frac{f^{-1}(0)}{s}$
$f(t-a)u(t-a), a > 0$ (u is the step function)	$e^{-as}F(s)$
$e^{-at}f(t)$	$F(s+a)$
$f(at), a > 0$	$\frac{1}{a}F\left(\frac{s}{a}\right)$
$tf(t)$	$-\frac{d}{ds}(F(s))$
$t^nf(t)$	$-(1)^n\frac{d^n}{ds^n}F(s)$
$\frac{f(t)}{t}$	$\int_{s}^{\infty} F(u)du$ (u is the step function)

Table 10 Laplace Transforms for Common Operations (Continued)

f(t)	**L{f(t)} = F(s)**
$f(t-t_1)$	$e^{-t_1 s}F(s)$

Determining the Laplace Coefficients

The following examples describe how to determine the appropriate coefficients, for the Laplace modeling function call.

Laplace Example 1 – Voltage Gain Transfer Function

To find the voltage-gain transfer function for the circuit in Figure 93 on page 334, convert the circuit to its equivalent s-domain circuit, and solve for v_o / v_g.

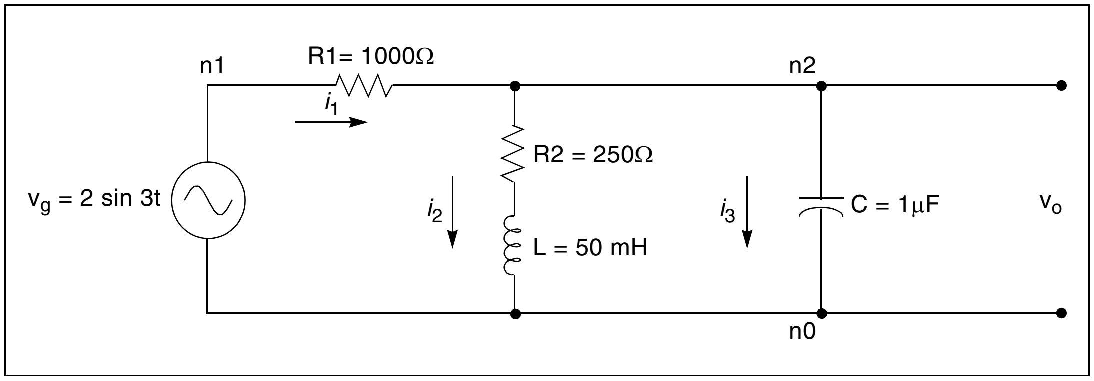


Figure 93 LAPLACE Example 1 Circuit

Use transforms fromTable 10 on page 332 to convert the inductor, capacitor, and resistors. L{f(t)} represents the Laplace transform of f(t):

$$L\left\{L\frac{d}{dt}f(t)\right\} = L\cdot(sF(s)-f(0)) = 50\times10^{-3}\cdot(s-0) = 0.05s$$

$$L\left\{\frac{1}{C}\int_0^t f(t)dt\right\} = \frac{1}{C}\cdot\left(\frac{F(s)}{s}+\frac{f^{-1}(0)}{s}\right) = \frac{1}{10^{-6}}\cdot\left(\frac{1}{s}+0\right) = \frac{10^6}{s}$$

$$L\{R1 \cdot f(t)\} = R1 \cdot F(s) = R1 = 1000 \quad W$$

$$L\{R2 \cdot f(t)\} = R2 \cdot F(s) = R2 = 250 \quad W$$

To convert the voltage source to the s-domain, use the sin ω t transform from Table 9 on page 331:

$$L\{2\sin 3t\} = 2 \cdot \frac{3}{s^2 + 3^2} = \frac{6}{s^2 + 9}$$

Figure 94 on page 335 displays the s-domain equivalent circuit.

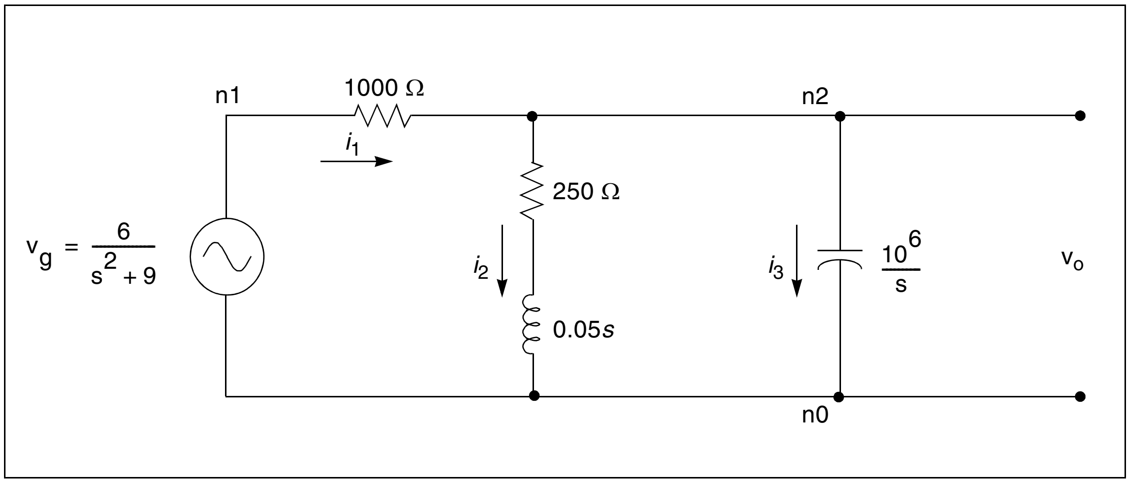


Figure 94 S-Domain Equivalent of the LAPLACE Example 1 Circuit

Summing the output currents from the n2 node:

$$\frac{v_o - v_g}{1000} + \frac{v_o}{250 + 0.05s} + \frac{v_o s}{10^6} = 0$$

Solve for v_o:

$$v_o = \frac{1000(s + 5000)v_g}{s^2 + 6000s + 25 \times 10^6}$$

The voltage-gain transfer function is:

$$H(s) = \frac{v_o}{v_g} = \frac{1000(s + 5000)}{s^2 + 6000s + 25 \times 10^6} = \frac{5 \times 10^6 + 1000s}{25 \times 10^6 + 6000s + s^2}$$

For the `Laplace` function call, use the k_n and d_m coefficients for the transfer function, in the form:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

The coefficients from the above voltage-gain transfer function are:

$k_0 = 5 \times 10^6$ $k_1 = 1000$

$d_0 = 25 \times 10^6 d_1 = 6000$ $d_2 = 1$

Using these coefficients, the following is a Laplace modeling function call, for the voltage-gain transfer function of the circuit in Figure 93 on page 334:

LAPLACE Example 2 – Differentiator

To model a differentiator, use either G or E elements, as shown in the following example.

In the frequency domain:

E-element: $V_{out} = ksV_{in}$

G-element: $I_{out} = ksV_{in}$

In the time domain:

E-element: $v_{out} = k\frac{dV_{in}}{dt}$

G-element: $i_{out} = k\frac{dV_{in}}{dt}$

For a differentiator, the voltage gain transfer function is:

$$H(s) = \frac{V_{out}}{V_{in}} = ks$$

In the general form of the transfer function:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

If you set k_1 = k and d_0 = 1, and the remaining coefficients are zero, then the equation becomes:

$$H(s) = \frac{ks}{1} = ks$$

Using the k_1 = k and d_0 = 1 coefficients, in the Laplace modeling, the circuit descriptions for the differentiator are:

```
Edif out GND LAPLACE in GND 0 k / 1
Gdif out GND LAPLACE in GND 0 k / 1
```

LAPLACE Example 3 – Integrator

You can use G or E Elements to model an integrator, as follows:

In the frequency domain:

E Element: $V_{out} = \frac{k}{s} V_{in}$

G Element: $I_{out} = \frac{k}{s} V_{in}$

In the time domain:

E Element: $v_{out} = k\int V_{in} dt$

G Element: $i_{out} = k\int V_{in} dt$

For an integrator, the voltage gain transfer function is:

$$H(s) = \frac{V_{out}}{V_{in}} = \frac{k}{s}$$

In the general form of the transfer function:

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m}$$

As in the previous example, if you set k_0 = k and d_1 = 1, then the equation becomes:

$$H(s) = \frac{k + 0 + \ldots + 0}{0 + s + \ldots + 0} = \frac{k}{s}$$

Laplace Transform POLE (Pole/Zero) Function

The following sections describe the general form of the pole/zero transfer function. It also provides examples of converting specific transfer functions, into pole/zero circuit descriptions.

The topics discussed are as follows:

- POLE Function Call
- General Form of the Transfer Function
- Reduced Form of the Transfer Function
- RC Line Modeling

POLE Function Call

You can use the `POLE` function if the poles and zeros of the circuit are available. You can derive the poles and zeros from the transfer function, as described in this chapter, or you can use the `.PZ` statement to find them, as described in .PZ in *HSPICE Reference Manual: Commands and Control Options*.

HSPICE provides two forms of the `LAPLACE` function call: one for transconductance, and one for voltage gain transfer functions. See Using G- and E-elements on page 319 for the general forms, and for optional parameters.

To use the `POLE` pole/zero modeling function, find the a, b, f, and α coefficients of the transfer function. The transfer function is the s-domain (frequency domain) ratio of the output, for a single-source circuit to the input, with initial conditions set to zero.

General Form of the Transfer Function

The general expanded form of the pole/zero transfer function H(s) is:

$$H(s) = \frac{a(s+\alpha_{z1}+j2\pi f_{z1})(s+\alpha_{z1}-j2\pi f_{z1})\ldots(s+\alpha_{zn}+j2\pi f_{zn})(s+\alpha_{zn}-j2\pi f_{zn})}{b(s+\alpha_{p1}+j2\pi f_{p1})(s+\alpha_{p1}-j2\pi f_{p1})\ldots(s+\alpha_{pm}+j2\pi f_{pm})(s+\alpha_{pm}-j2\pi f_{pm})}$$

You can use parameters to set the a, b, α, and f values.

The following is an example:

```
Ghigh_pass 0 out   POLE in 0 1.0   0.0,0.0 / 1.0  0.001,0.0
Elow_pass out 0    POLE in 0   1.0 / 1.0, 1.0,0.0   0.5,0.1379
```

The `Ghigh_pass` statement describes a high pass filter, with the transfer function:

$$H(s) = \frac{1.0 \cdot (s + 0.0 + j \cdot 0.0)}{1.0 \cdot (s + 0.001 + j \cdot 0.0)}$$

The `Elow_pass` statement describes a low-pass filter, with the transfer function:

$$H(s) = \frac{1.0}{1.0 \cdot (s+1)(s + 0.5 + j2\pi \cdot 0.1379)(s + 0.5 - (j2\pi \cdot 0.1379))}$$

To write a pole/zero circuit description for an element, you need to know H(s) transfer function of the element, in terms of the a, b, f, and α coefficients.

Before you use the values of these coefficients in `POLE` function calls (in the circuit description), you must simplify the transfer function, as described in the next section.

Reduced Form of the Transfer Function

Complex poles and zeros occur in conjugate pairs (a set of complex numbers differ only in the signs of their imaginary parts):

$(s + \alpha_{pm} + j2\pi f_{pm})(s + \alpha_{pm} - j2\pi f_{pm})$, for poles.

$(s + \alpha_{zn} + j2\pi f_{zn})(s + \alpha_{zn} - j2\pi f_{zn})$, for zeros.

To write the transfer function in pole/zero format, supply coefficients for one term of each conjugate pair. HSPICE provides the coefficients for the other term. If you omit the negative complex roots, the result is the reduced form of the transfer function, Reduced{H(s)}.

To find the reduced form, collect all general-form terms that have negative complex roots:

$$H(s) = \frac{a(s + \alpha_{z1} + j2\pi f_{z1})\ldots(s + \alpha_{zn} + j2\pi f_{zn})}{b(s + \alpha_{p1} + j2\pi f_{p1})\ldots(s + \alpha_{pm} + j2\pi f_{pm})} \cdot \frac{a(s + \alpha_{z1} - j2\pi f_{z1})\ldots(s + \alpha_{zn} - j2\pi f_{zn})}{b(s + \alpha_{p1} - j2\pi f_{p1})\ldots(s + \alpha_{pm} - j2\pi f_{pm})}$$

Then discard the right-hand term, which contains all terms with negative roots. What remains is the reduced form:

$$Reduced\{H(s)\} = \frac{a(s + \alpha_{z1} + j2\pi f_{z1})\ldots(s + \alpha_{zn} + j2\pi f_{zn})}{b(s + \alpha_{p1} + j2\pi f_{p1})\ldots(s + \alpha_{pm} + j2\pi f_{pm})}$$

For this function, find the a, b, f, and α coefficients to use in a `POLE` function, for a voltage-gain transfer function. The following examples show how to determine the coefficients, and write `POLE` function calls for a high-pass filter and a low-pass filter.

POLE Example 1 – Highpass Filter

For a high-pass filter with a transconductance transfer function, such as:

$$H(s) = \frac{s}{(s + 0.001)}$$

Find the a, b, α , and f coefficients needed to write the transfer function in the general form shown previously. You can then see the conjugate pairs of complex roots. You need to supply only one of each conjugate pair of roots, in the `Laplace` function call. HSPICE automatically inserts the other root.

To transform the function into a form that is more similar to the general form of the transfer function, rewrite the transconductance transfer function as:

$$H(s) = \frac{1.0(s + 0.0)}{1.0(s + 0.001)}$$

Because this function has no negative imaginary parts, it is already in the HSPICE reduced form (reference number 2) shown previously.

You can now identify the a, b, f, and α coefficients, so that the H(s) transfer function matches the reduced form. This matching process obtains the values:

n = 1, m = 1

a = 1.0 α_{z1} = 0.0 f_{z1} = 0.0

b = 1.0 α_{p1} = 0.001 f_{p1} = 0.0

Using these coefficients in the reduced form, provides the transfer function:

$$\frac{s}{(s + 0.001)}$$

So the general transconductance transfer function `POLE` function call:

```
Gxxx n+ n- POLE in+ in- a α z1,fz1... α zn,fzn /
b  α p1,fp1... α pm,fpm
```

for an element named Ghigh_pass, becomes:

```
Ghigh_pass gnd out POLE in gnd 1.0 0.0,0.0 /
+ 1.0 0.001,0.0
```

POLE Example 2 – Low-Pass Filter

For a low-pass filter, with the following voltage-gain transfer function:

$$H(s) = \frac{1.0}{1.0(s + 1.0 + j2\pi \cdot 0.0)(s + 0.5 + j2\pi \cdot 0.15)(s + 0.5 - j2\pi \cdot 0.15)}$$

you need to find the a, b, α, and f coefficients, to write the transfer function in the general form, so that you can identify the complex roots with negative imaginary parts.

To separate the reduced form, Reduced{H(s)}, from the terms with negative imaginary parts, rewrite the voltage-gain transfer function as:

$$H(s) = \frac{1.0}{1.0(s + 1.0 + j2\pi \cdot 0.0)(s + 0.5 + j2\pi \cdot 0.15)} \cdot \frac{1.0}{(s + 0.5 - j2\pi \cdot 0.15)}$$
$$= Reduced\{H(s)\} \cdot \frac{1.0}{(s + 0.5 - j2\pi \cdot 0.15)}$$

So:

$$Reduced\{H(s)\} = \frac{1.0}{1.0(s + 1.0)(s + 0.5 + j2\pi \cdot 0.15)}$$

or:

$$\frac{a(s + \alpha_{z1} + j2\pi f_{z1})\ldots(s + \alpha_{zn} + j2\pi f_{zn})}{b(s + \alpha_{p1} + j2\pi f_{p1})\ldots(s + \alpha_{pm} + j2\pi f_{pm})} = \frac{1.0}{1.0(s + 1.0 + j2\pi \cdot 0.0)(s + 0.5 + j2\pi \cdot 0.15)}$$

Now assign coefficients in the reduced form, to match the specified voltage transfer function. The following coefficient values produce the transfer function:

n = 0, m = 2,
a =1.0   b = 1.0 α_{p1} = 1.0 f_{p1} = 0 α_{p2} = 0.5 f_{p2} = 0.15

You can substitute these coefficients in the `POLE` function-call, for a voltage-gain transfer function:

Exxx n+ n- POLE in+ in- a α_{z1},f_{z1}...α_{zn},f_{zn} /
b α_{p1},f_{p1}...α_{pm},f_{pm}

for an element named `Elow_pass`, to obtain the following statement:

```
Elow_pass out GND POLE in 1.0 / 1.0 1.0,0.0 0.5,0.15
```

RC Line Modeling

Most RC lines can use very simple models, with only a single dominant pole. You can use AWE methods to find the dominant pole, computed based on the total series resistance and capacitance, or determined using the Elmore delay.

The Elmore delay uses the (d1-k1) value as the time constant, for a single-pole approximation to the complete H(s), where H(s) is the transfer function of the RC network for a specified output. The inverse Laplace transform of h(t) is H(s):

$$\tau_{DE} = \int_0^{\infty} t \cdot h(t)dt$$

Actually, the Elmore delay is the first moment of the impulse response, and so corresponds to a first-order AWE result.

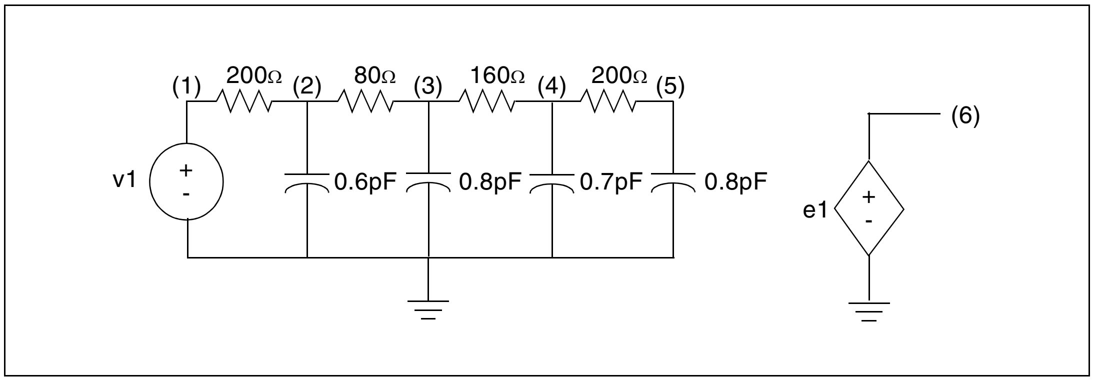


Figure 95 Circuits for a RC Line

Example

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

```
****
.Tran 0.02ns 3ns
.OPTION Post Accurate List Probe
v1 1 0 PWL 0ns 0 0.1ns 0 0.3ns 5 1.3ns 5 1.5ns 0
r1 1 2 200
c1 2 0 0.6pF
r2 2 3 80
c2 3 0 0.8pF
r3 3 4 160
c3 4 0 0.7pF
r4 4 5 200
c4 5 0 0.8pF
e1 6 0 LAPLACE 1 0 1 / 1 1.16n
.Probe v(1) v(5) v(6)
.Print v(1) v(5) v(6)
.End
```

To closely approximate the output of the RC circuit (shown in Figure 95), you can use a single-pole response, as shown in Figure 96.

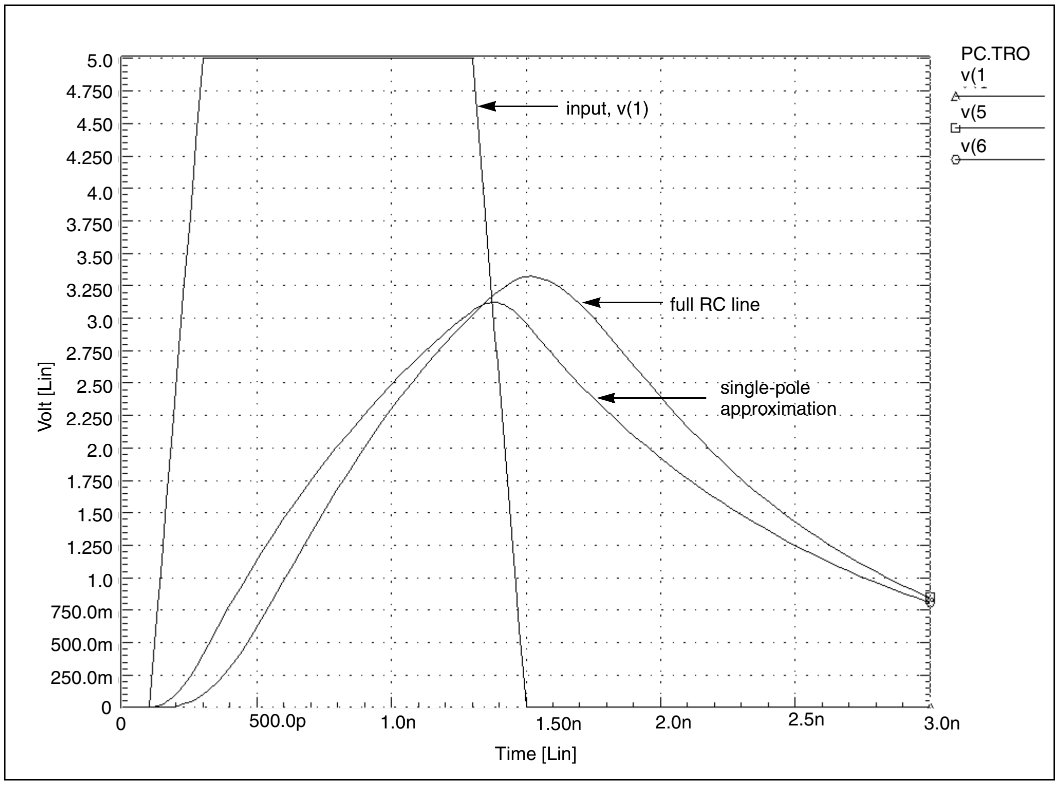


Figure 96   Transient Response of the RC Line and Single-Pole Approximation

In Figure 96, the single-pole approximation has less delay: 1 ns, compared to 1.1 ns for the full RC line model, at 2.5 V. The single-pole approximation also has a lower peak value than the RC line model. All other things being equal, a circuit with a shorter time constant results in less filtering, and allows a higher maximum voltage value. The single-pole approximation produces a lower amplitude (and less delay) than the RC line because the single pole neglects the other three poles in the actual circuit. However, a single-pole approximation still provides very good results for many problems.

AWE Transfer Function Modeling

Approximations, using single-pole transfer functions, can cause larger errors for low-loss lines than for RC lines because lower resistance allows ringing. Circuit ringing creates complex pole pairs in transfer function approximation. You need at least one complex pole pair, to represent low-loss line response. Figure 97 is a typical low-loss line, and the transfer function sources used to test various approximations. To obtain the transfer functions, HSPICE uses asymptotic waveform evaluation.

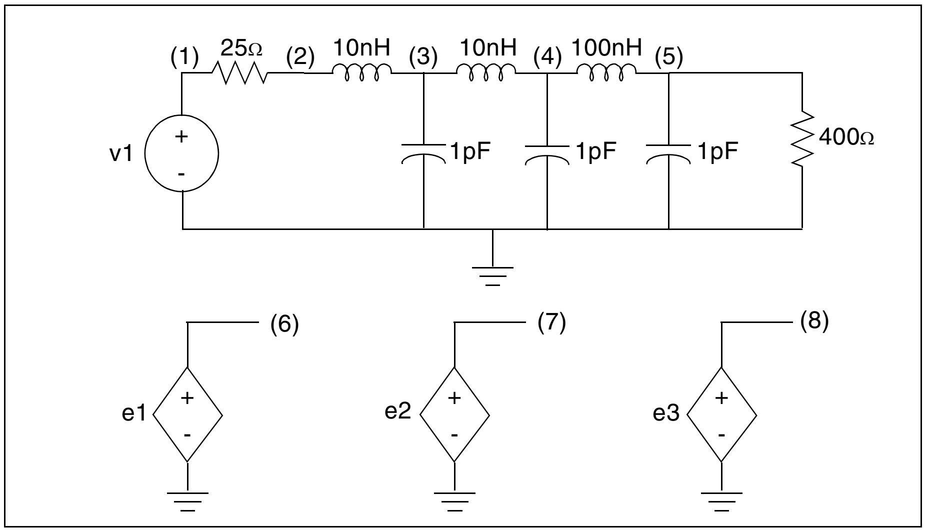


Figure 97 Circuits for a Low-Loss Line

The sample file located in the following directory is a Low-Loss Line circuit file:
$installdir/demo/hspice/filters/lowloss.sp

Figure 98 shows a transient response of a low-loss line. It also shows E Element Laplace models, using one, two, and four poles. The single-pole model shows none of the ringing of the higher-order models. Also, all E models must adjust the gain of their response, for the finite load resistance, so the models are not independent of the load impedance. The 0.94-gain multiplier in the models take care of the 25-ohm source, and the 400-ohm load-voltage divider. These approximations are good delay estimations.

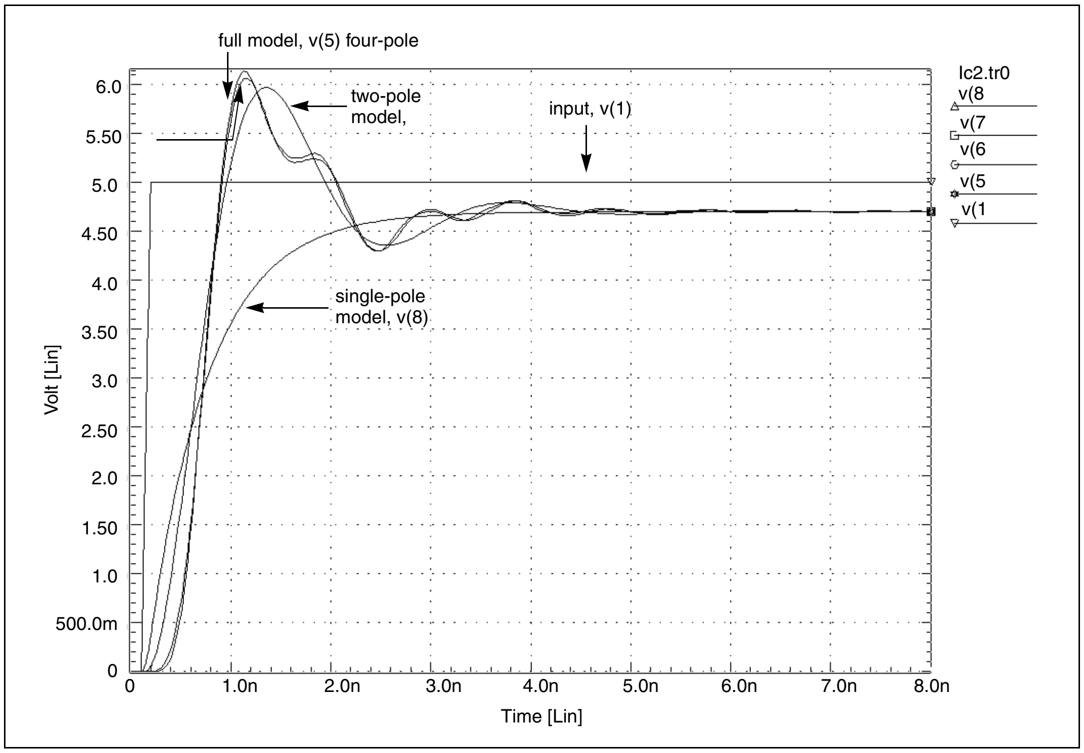


Figure 98 Transient Response of the Low-Loss Line

Although the two-pole approximation provides reasonable agreement with the transient overshoot, the four-pole model offers almost perfect agreement. The actual circuit has six poles. You can use scaling to bring some of the very small numbers in the Laplace model, above the 1e-28 limit of HSPICE. The `SCALE` parameter multiplies ever Y-parameter in the LAPLACE specification, by the same value (in this case 1.0E-20).

A low-loss line allows reflections between the load and source, compared to the loss of an RC line, which usually isolates the source from the load. So you can either incorporate the load into the AWE transfer function approximation, or create a HSPICE device model that allows source/load interaction. If you allow source/load interaction, you do not need to perform the AWE expansions each time that you change load impedances. This allows HSPICE to handle nonlinear loads, and removes the need for a gain multiplier, as in the circuit file shown. You can use four voltage-controlled current sources, or G Elements, to create a Y-parameter model for a transmission line. The Y-parameter network provides the needed source/load interaction. The next example shows such a Y-parameter model, for a low-loss line.

Y-parameter Line Modeling

A model that is independent of load impedance, is more complicated. You can still use AWE techniques, but you need a way for the load voltage and current to interact with the source impedance. For a transmission line of 100 ohms and 0.4 ns total delay (as shown in Figure 99), to compare the response of the line, use a Y-parameter model and a single-pole model.

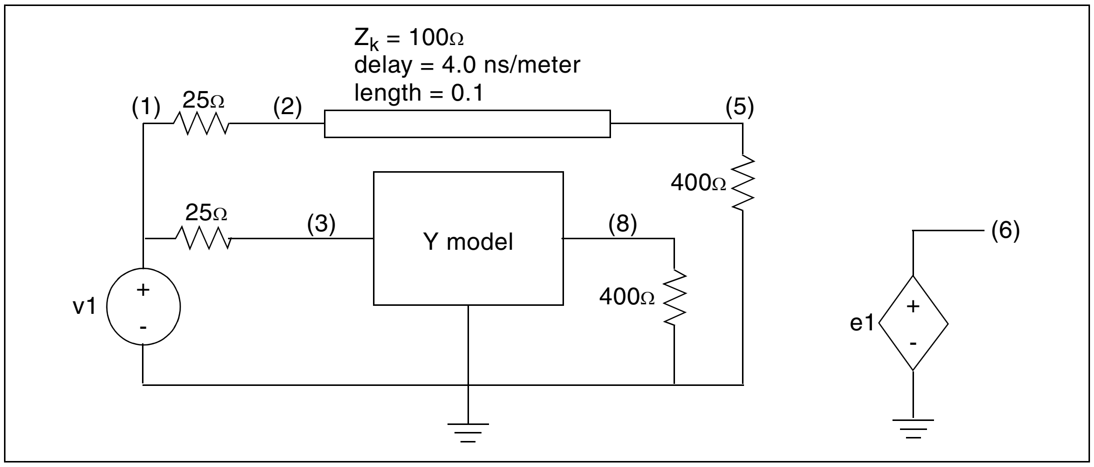


Figure 99 Line and Y-parameter Modeling

Figure 100 shows the voltage and current definitions for a Y-parameter model.

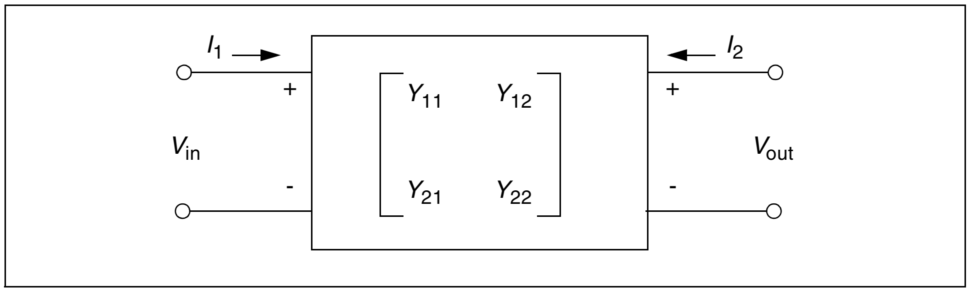


Figure 100 Y Matrix for the Two-Port Network

The following equations describe the general network in Figure 100, which you can translate into G Elements:

$$I_1 = Y_{11}V_{in} + Y_{12}V_{out}$$

$$I_2 = Y_{21}V_{in} + Y_{22}V_{out}$$

Figure 101 shows a schematic for a set of two-port Y-parameters. The circuit consists mostly of G Elements.

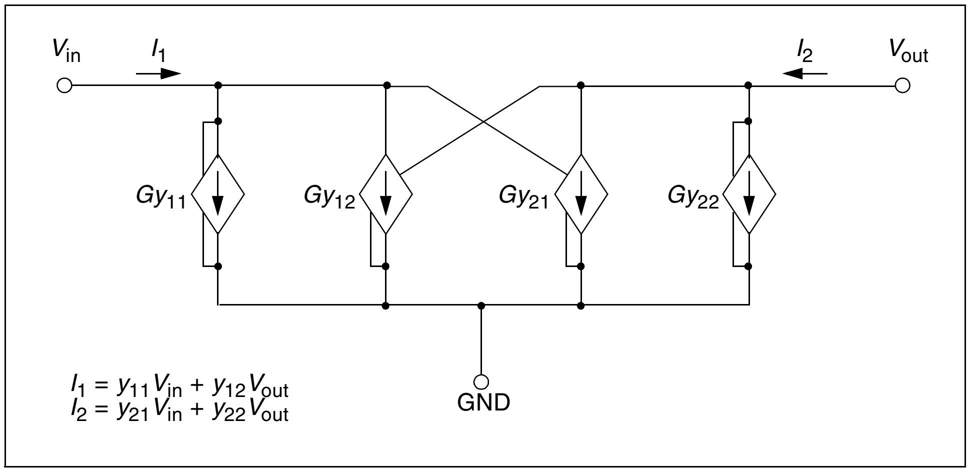

V_{in}
I_1
I_2
V_{out}
Gy_{11}
Gy_{12}
Gy_{21}
Gy_{22}
$I_1 = y_{11}V_{in} + y_{12}V_{out}$
$I_2 = y_{21}V_{in} + y_{22}V_{out}$
GND

Figure 101 Schematic for the Y-parameter Network

A Pade expansion of the Y-parameters for a transmission line, determines the Laplace parameters for the Y-parameter model, as shown in matrix form in the following equation:

$$Y = \frac{1}{Z_o} \cdot \begin{bmatrix} \coth(p) & -\mathrm{csch}(p) \\ -\mathrm{csch}(p) & \coth(p) \end{bmatrix}$$

where *p* is the product of the propagation constant, times the line length.

A Pade approximation contains polynomials, in both the numerator and the denominator. A Pade approximation can model both poles and zeros, and coth and csch functions also contain both poles and zeros, so a Pade approximation provides a better low-order model, than a series approximation does.

The following equation calculates the Pade expansion of coth(p) and csch(p), with a second-order numerator and a third-order denominator:

$$coth(p) \rightarrow \frac{\left(1 + \frac{2}{5} \cdot p^2\right)}{\left(p + \frac{1}{15} \cdot p^3\right)} \qquad csch(p) \rightarrow \frac{\left(1 - \frac{1}{20} \cdot p^2\right)}{\left(p + \frac{7}{60} \cdot p^3\right)}$$

When you substitute $(s \cdot length \cdot \sqrt{LC})$ for p, HSPICE generates polynomial expressions for each G Element. When you substitute 400 nH for L, 40 pF for C, 0.1 meter for length, and 100 for Z_o, $(Z_o = \sqrt{L/C})$ in the matrix equation above, you can use the resulting values in a circuit file.

The circuit file shown in the following sections uses all of the equation substitutions. The Pade approximations have different denominators for csch and coth, but the circuit file contains identical denominators. Although the actual denominators for csch and coth are only slightly different, using them can cause oscillations in the HSPICE response. To avoid this problem, use the same denominator in the coth and csch functions in the example. The simulation results might vary, depending on which denominator you use as the common denominator because the coefficient of the third-order term changes (but by less than a factor of 2).

This sample LC Line circuit file is located in the following directory:
$installdir/demo/hspice/filters/lcline.sp

Figure 102 compares the output of the Y-parameter model, with that of a full transmission line simulation, and with that obtained for a single-pole transfer function. In the latter case, the gain for the load impedance is incorrect, so the function produces an incorrect final voltage level. As expected, the Y parameter model provides the correct final voltage level. Although the Y parameter model provides a good approximation of the circuit delay, it contains too few poles to model all of the transient details. However, the Y-parameter model provides excellent agreement with the overshoot and settling times.

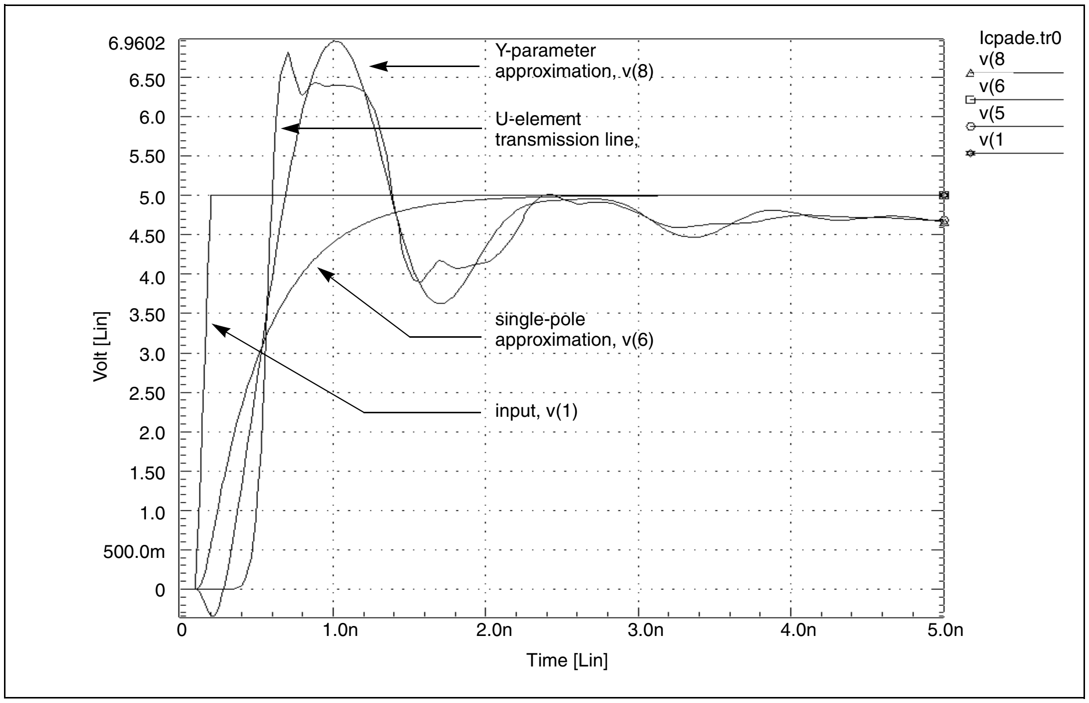


Figure 102 Transient Response of the Y-parameter Line Model

Comparison of Circuit and Pole/Zero Models

This example simulates a ninth-order, low-pass filter circuit, and compares the results with its equivalent pole/zero description, using an E Element. The results are identical, but the pole/zero model runs about 40% faster.The example shown in Simulation Time Summary on page 349 shows the total CPU times for the two methods. For larger circuits, the computation time saving can be much higher.

Figure 103 on page 350 and Figure 104 on page 351 display the transient and frequency response comparisons, resulting from the two modeling methods.

Simulation Time Summary

Circuit model simulation times:

```
analysis   time   # points   # iter   conv.iter
op point   0.23   1   3
ac analysis   0.47   151   151
transient   0.75   201   226   113  rev=0
readin   0.22
errchk   0.13
setup   0.10
output   0.00
total cpu time 1.98 seconds
```

Pole/zero model simulation times:

```
analysis   time   # points   #
 iter   conv.iter
op point   0.12   1   3
ac analysis   0.22   151   151
transient   0.40   201   222   111  rev=0
readin   0.23
errchk   0.13
setup   0.02
output   0.00
total cpu time 1.23 seconds
```

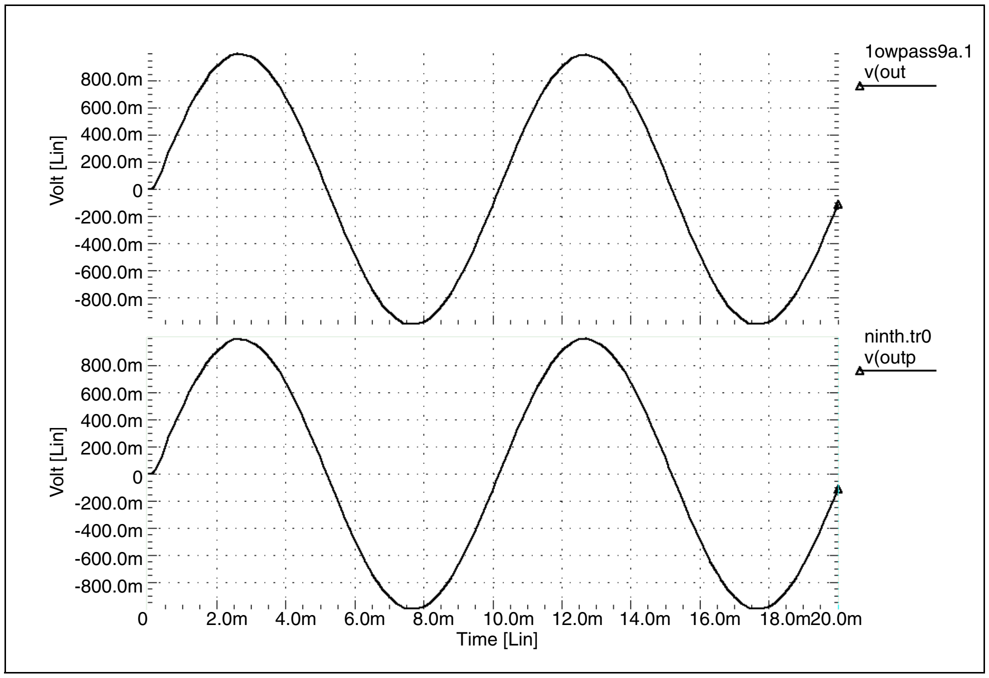


Figure 103 Transient Responses of the Circuit and Pole/Zero Models

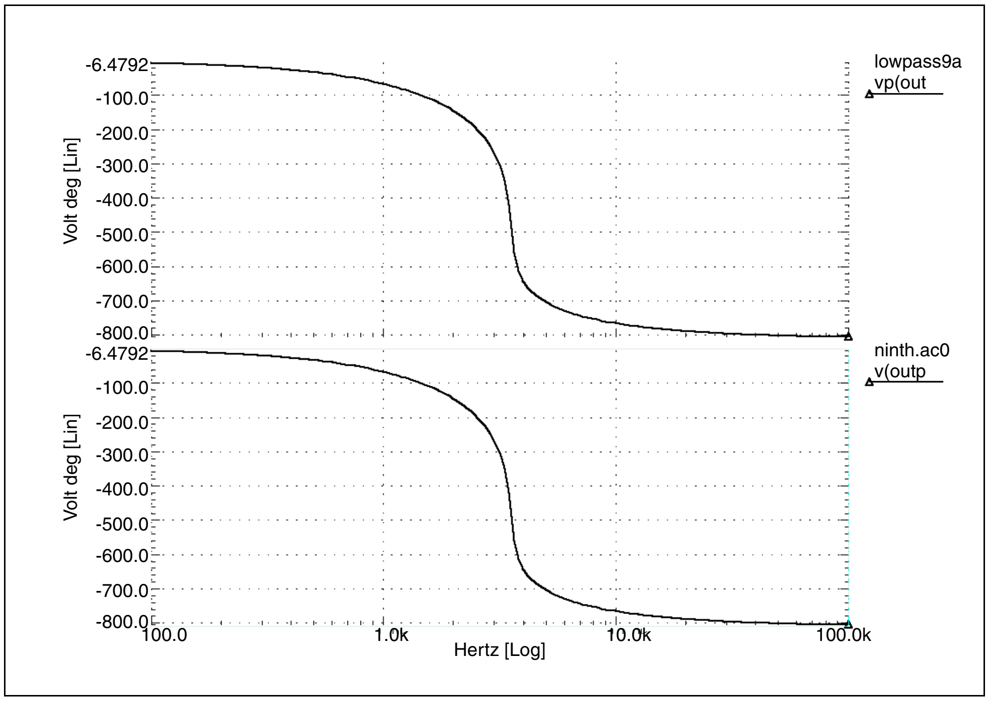


Figure 104 AC Analysis Responses of the Circuit and Pole/Zero Models

Modeling Switched Capacitor Filters

The following sections discuss these topics:

- Switched Capacitor Network
- Switched Capacitor Filter Example
- Input File for Switched Capacitor Filter

Switched Capacitor Network

You can model a resistor as a capacitor and switch combination. The value of the equivalent is proportional to the frequency of the switch, divided by capacitance.

Construct a filter from MOSFETs and capacitors, where the filter characteristics are a function of the switching frequency of the MOSFETs.

To quickly determine the filter characteristics, use ideal switches (voltage controlled resistors), instead of MOSFETs. The resulting simulation speed-up can be as great as 7 to 10 times faster than a circuit using MOSFETs.

To construct an RC network, the model uses a resistor and a capacitor, along with a switched-capacitor equivalent network. The `RCOUT` node is the resistor/capacitor output, and `VCROUT` is the switched-capacitor output.

The `GVCR1` and `GVCR2` switches, and the `C3` capacitance, model the resistor. The following equation calculates the resistor value:

$$Res = \frac{Tswitch}{C3}$$

where Tswitch is the period of the PHI1 and PHI2 pulses.

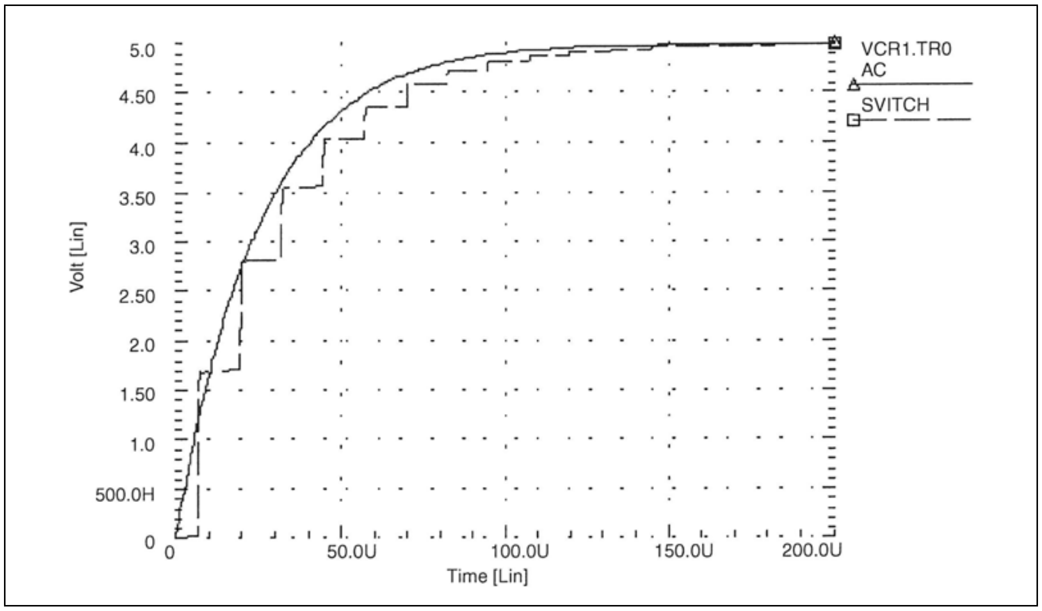


Figure 105 VCR1.SP Switched Capacitor RC Circuit

Switched Capacitor Filter Example

This example is a fifth-order elliptic, switched-capacitor filter. The passband is 0-1 kHz, with loss less than 0.05 dB. This results from cascading models of the switches. The resistance is1 ohm when the switch is closed, and 100 Megohm when it is open. The E Element models op-amps as an ideal op-amp. This example provides the transient response of the filter, for 1 kHz and 2 kHz sinusoidal input signals.

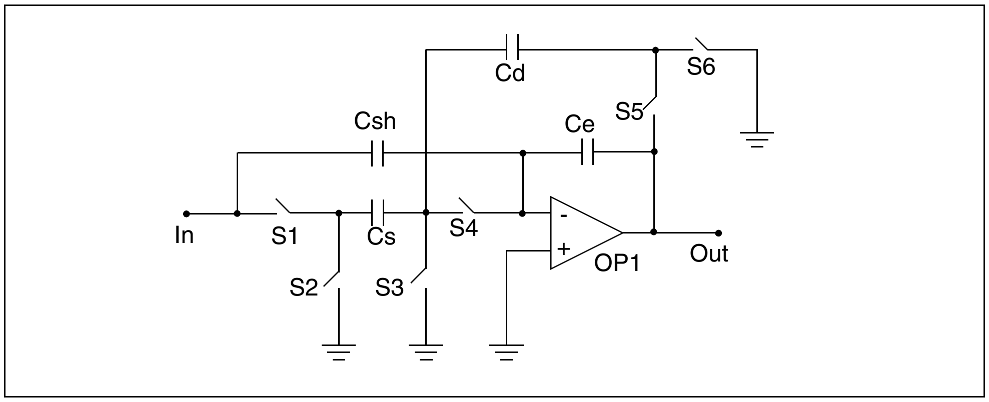


Figure 106 Linear Section

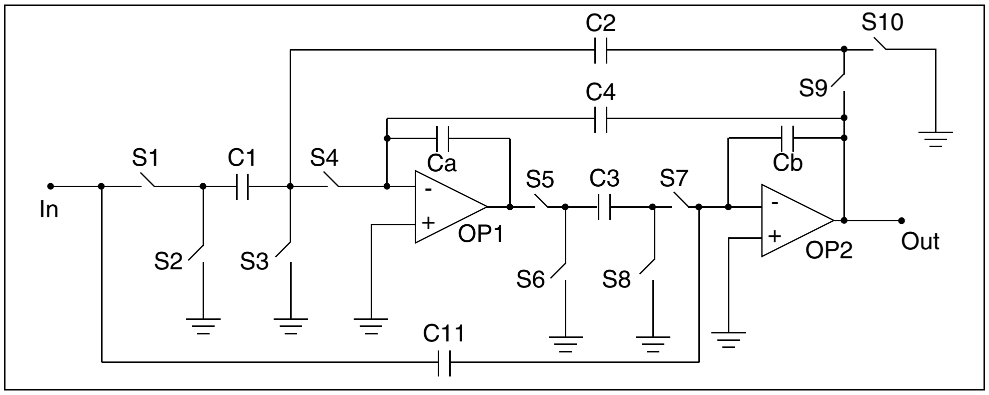


Figure 107 High_Q Biquad Section

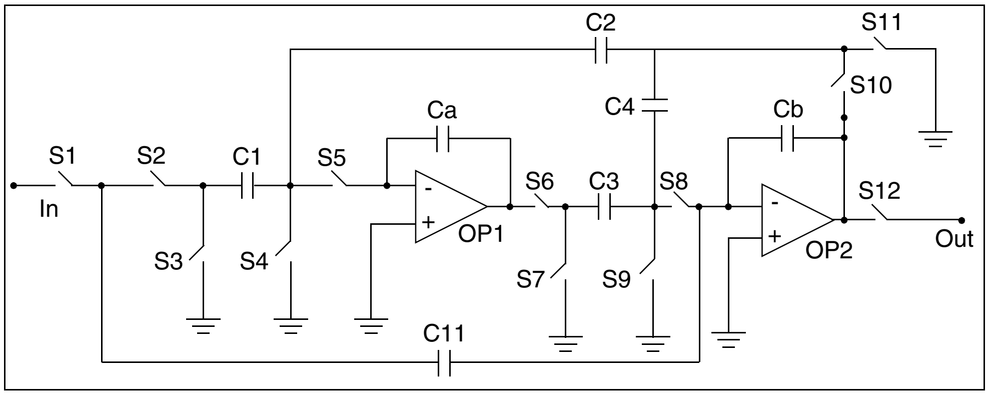


Figure 108 Low_Q Biquad Section

Input File for Switched Capacitor Filter

This sample input file for a switched capacitor filter is located in the following directory:

$installdir/demo/hspice/behave/swcap5.sp

This file also contains the following examples:

- Sample and Hold
- Linear Section
- High_Q Biquad Section
- Low_Q Biquad Section

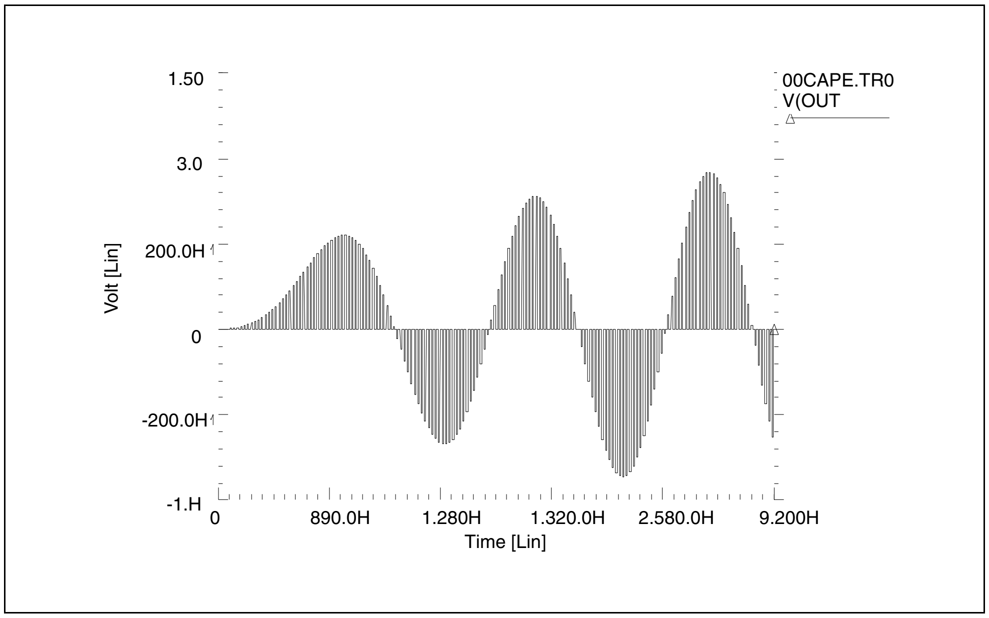


Figure 109 Response to 1-kHz Sinusoidal Input

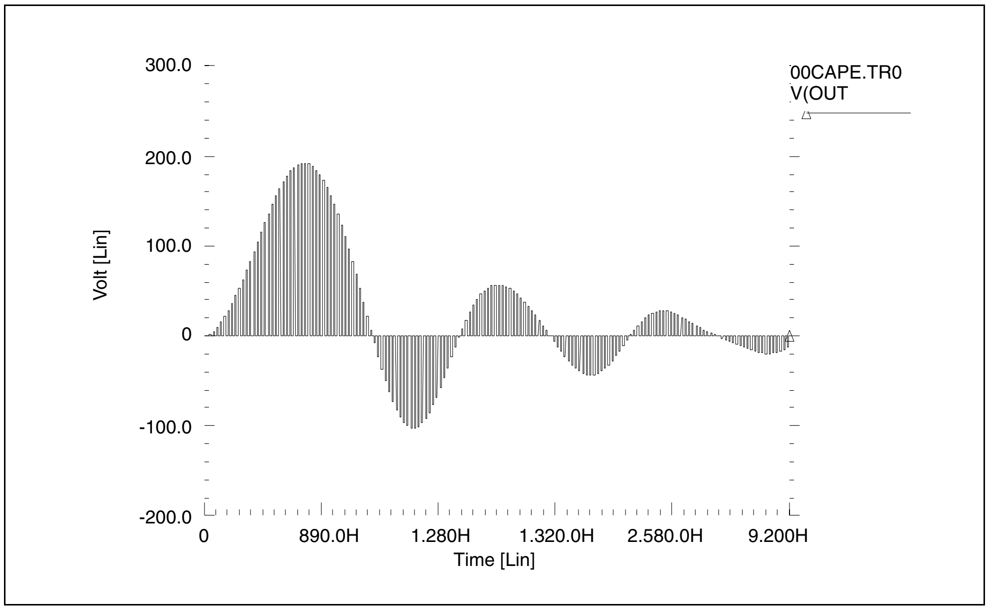


Figure 110 Response to 2-kHz Sinusoidal Input

References

[1] Williams, Arthur B., and Taylor, Fred J. *Electronic Filter Design Handbook*. New York: McGraw-Hill, 1988, pp. 6-20 to 6-23.

[2] Nillson, James W. *Electric Circuits*, 4th Edition. Reading, Massachusetts: Addison-Wesley, 1993.

[3] Edminister, Joseph A. *Electric Circuits*. New York: McGraw-Hill, 1965.

[4] Ghausi, Kelly, and M.S. “On the Effective Dominant Pole of the Distributed RC Networks”, *Jour. Franklin Inst.*, June 1965, pp. 417- 429.

[5] Elmore, W.C. and Sands, M. *Electronics, National Nuclear Energy Series*, New York: McGraw-Hill, 1949.

[6] Pillage, L.T. and Rohrer, R.A. “Asymptotic Waveform Evaluation for Timing Analysis”, *IEEE Trans. CAD*, Apr. 1990, pp. 352 - 366.

[7] Kuo, F. F. *Network Analysis and Synthesis*. John Wiley and Sons, 1966.

[8] Gregorian, Roubik & Temes, Gabor C. *Analog MOS Integrated Circuits*. J. Wiley, 1986, page 354.

Index

A

AC analysis
  circuit and pole/zero models 351
accuracy
  element parameter 321
algorithm
  linear acceleration 231
  nonlinear perturbation 105
  periodic AC 105
AM
  behavioral 299
amplifier 45
amplifier, IP3 48
amplitude modulation (AM) 111
AM-PM separation 111
analog
  elements 295
  modeling 282
analysis
  circuit model 351
  phase noise 101
  pole/zero
    model 351
  time domain steady-state 61
  transfer function (.TF) 293
asymptotic waveform evaluation 318, 344
autocorrelation function 237
autocorrelation function, jitter measurement 252
AWE *See* asymptotic waveform evaluation

B

behavioral
  741 op-amp 301
  amplitude modulator 299
  AND and NAND gates 283
  BJTs 311
  CMOS inverter 292
  comparator 304
  components 288
  controlled sources 282
  data sampler 300
  differentiator 297
  D-Latch 284
  double-edge triggered flip-flop 286
  elements 280, 295
  flip-flop 286
  gates 283
  LC oscillator 307
  look-up tables 288
  n-channel MOSFETs 285, 290
  p-channel MOSFETs 285
  phase detector model 310
  phased locked loop 310
  ring oscillator 294
  transformer 298
  VCO model 305
  VVCAP model 308
behavioral modeling 279
broadband phasenoise 106
broadband phasenoise algorithm 106

C

capacitance
  voltage variable 308
capacitor
  switched 351
.CHECK EDGE statement 16
.CHECK FALL statement 16
.CHECK GLOBAL_LEVEL statement 15
.CHECK HOLD statement 17
.CHECK IRDROP statement 18
.CHECK RISE statement 15
.CHECK SETUP statement 17
.CHECK SLEW statement 15
circuits
  AC analysis 351
  model 349
  RC line 342
  transient responses 350
clock source, random jitter 122
CMOS GPS VCO 93
coefficients

Laplace 334
transfer function 331
Colpitts oscillator 90
.command
.PRINT ENV 201
command
.PROBE ENV 201
commands
PTDNOISE 166
comparators 301, 304
comparing results 139
complex poles and zeros 339
component noise models 269
config file
hspicerf 8
configuration file 8
example 10
configuration options
flush_waveform 8
ground_floating_ node 8
hier_delimiter 8
html 9
integer_node 9
max_waveform_size 9
negative_td 9
port_element_ voltage_ matchload 9
rcxt_divider 9
unit_atto 9
v_supply 9
wildcard_left_range 10
wildcard_match_all 10
wildcard_match_one 10
wildcard_right_range 10
controlled sources 282, 283
critical frequency 320
current
controlled
elements 283
Custom WaveView 7

D

data
sampler, behavioral 300
DDL 282
delays
Elmore 342
DELF element parameter 321
DELTA
element parameter 321
demo files
22
demo files, RF 22
demonstration files, RF 22
demonstration input files 22
design
stability 318
time, reducing 318
Detailed Standard Parasitic Format *See* DSPF
device model cards 55
differentiator, behavioral 297
digital
files 283
DSPF
expansion 215
file structure 210

E

E Elements 283, 336–341
integrator application 297
Laplace transform 319
NAND gate 283
parameters 322
SCALE parameter 345
transformer application 298
edge condition 16
element
analog behavioral 295
current controlled 283
voltage-controlled 283
element noise models 269
element parameters
G and E Elements 321–322
Elmore delay 342
.ENV statement 198
Envelope Analysis (ENV) 197
envelope simulation 197
.ENVFFT 200
.ENVFFT statement 200
.ENVOSC 199
.ENVOSC statement 199
EPSMIN option 331
ergodic simulations 242
example
configuration file 10

F

F Elements 283
fall time
  verification 16
files
  .hl# 190
  hspicerf 8
  .ls# 196
  .p2d# 196
  .printhl# 190
  .printls# 196
  .printss# 196
  .ss# 196
files, output 5
filters
  30 degree phase shift 328
  band reject 324
  low-pass 325
  transfer functions 340
flags 8
flicker noise 265
flush_waveform configuration option 8
format
  output
    DSPF 209
Fourier
  integral 320
  transfer function H(f) 320
  transform 320
FREQ
  element parameter 322
  keyword 320
frequency
  analysis 322
  complex 319, 330
  critical 320
  domain to time domain 317
  domain, transfer function 331
  maximum 321
    frequency analysis 322
  resolution 321
  response 319
    analysis 318
    table 321, 322
  table 318
functions
  LAPLACE 318
  OPTIMIZE 281
  POLE 318

G

G Elements 283, 336–340
  AND gate 283
  Laplace transform 319
  multiply parameter 322
  parameter value multiplier 322
generating output 5
ground_floating_ node configuration option 8

H

H Elements 283
Harmonic Balance (HB) 27
  equations 29
  errors 34
  output 31
  warnings 34
.HB
  for HBLIN 186
HB analysis
  IP3 amplifier 48
  power amplifier 45
HB_GIBBS option 39
HBAC 131, 138
  errors 136, 145, 255
  example 138
  output 133, 201
  output data files 135
  syntax 133
  warnings 136, 145, 255
.HBLIN 184, 187
  limitations 186
  output syntax 189
.HBLSP 191
  example 194
  input syntax 193
  limitations 193
  output data files 190, 196
  output syntax 195
HBOSC analysis
  Colpitts oscillator 90
  VCO 93
HBXF
  command 173
hier_delimiter configuration option 8

.hl# file 190
hold time verification 17
hspicerf file 8
hspicerf test 9
html configuration option 9

I

ideal transformer 298
impulse response h(t) 320
input files demonstration 22
input files, demo examples 22
integer_node configuration option 9
inverse Laplace transform 342
inverter lookup table 290
IR drop
  checking 18

J

jitter
  random, with clock source 122
jitter measurements, transient noise 252
jitter, random, clock source 122

K

keywords
  FREQ 320

L

Laplace
  band-reject filter 324
  element parameter 322
  function 318, 320, 330, 334–337
  low-pass filter 325
  parameters 321
  transfer function 318, 330
  transform 317, 332
    frequency 322
    function call 319
    inverse 342
    modeling 330
    POLE (pole/zero) function 338
large-signal S parameter extraction 191
LC oscillator model 307
LEVEL parameter 322
linear
  acceleration 230
  matrix reduction 230
.ls# file 196

M

M element parameter 322
mA741 op-amp 301
MAX parameter 322
max_waveform_size configuration option 9
MAXF parameter 321, 322
.measure 202
.MEASURE ENV command 202
mixed mode simulation 282
mixed-signal simulation
  See mixed mode
model cards 55
models
  behavioral 279
Monte Carlo, TRANNOISE method 237
multiplier
  G and E Element values 322

N

negative_td configuration option 9
network, switched capacitor 351
noise
  component models 269
  element models 269
  flicker 265
  .HBNOISE 151, 159
  shot 268
  simulation 269, 270
  sources 262
  thermal 264
  types 263
noise parameter extraction
  small-signal 191
noise simulation 257
nonlinear elements 318
nonlinear perturbation algorithm 105
Nyquist critical frequency 320, 322

O

op-amps
  behavioral models 301
  mA741 model 301

operators 332
optimization
  behavioral models 292, 294
  syntax 12
OPTIMIZE keyword 281
.OPTION
  SIM_DSPF 208
  SIM_DSPF_ACTIVE 208, 212
  SIM_DSPF_INSERROR 214
  SIM_DSPF_LUMPCAPS 213
  SIM_DSPF_MAX_ITER 213
  SIM_DSPF_RAIL 213
  SIM_DSPF_SCALEC 213
  SIM_DSPF_SCALER 213
  SIM_DSPF_VTOL 212
  SIM_LA 209, 230, 233
  SIM_LA_FREQ 233
  SIM_LA_MAXR 233
  SIM_LA_MINC 233
  SIM_LA_MINMODE 233
  SIM_LA_TIME 233
  SIM_LA_TOL 233
  SIM_POWERDC_ACCURACY 19
  SIM_POWERDC_HSPICE 19
  SIM_POWERPOST 21
  SIM_POWERSTART 21
  SIM_SPEF 208
  SIM_SPEF_ACTIVE 212
  SIM_SPEF_INSERROR 214
  SIM_SPEF_LUMPCAPS 213
  SIM_SPEF_MAX_ITER 213
  SIM_SPEF_PARVALUE 214
  SIM_SPEF_RAIL 213
  SIM_SPEF_SCALEC 213
  SIM_SPEF_SCALER 213
  SIM_SPEF_VTOL 212
.OPTION HB_GIBBS 39
options
  EPSMIN 331
oscillator
  phase noise 101
oscillator example 90
oscillators
  behavioral models 301
  LC 307
  VCO 305
output
  files 5
  format
    DSPF 216
  generating 5
  restricting 11

P

.p2d# file 196
Pade approximation 347
parameters
  frequency response table 321
  Laplace 321
  pole/zero 321
periodic AC algorithm 105
periodic pime-dependent noise analysis 166
phase
  detector model 310
  locked loop, BJT model 311
phase modulation (PM) 111
phase noise 101
phase noise analysis 101
PHASENOISE 101
PHASENOISE algorithms 105
PI (linear acceleration) algorithm 232
PLL *See* phase locked loop
PLL, jitter measurements 115
POLE
  element parameter 322
  function 318, 338–341
    call 338
    highpass filter 340
    low-pass filter 341
pole/zero
  analysis 318
  complex 339
  models 349, 350, 351
  parameters 321
  transfer function 338
  transient modeling 318
port_element_voltage_matchload configuration
    option 9
power amplifier 45
power amplifier IP3 48
.POWER statement 20
.POWERDC statement 18
power-dependent S parameter extraction 191
.PRINT ENV command 201
.printhl# file 190
.printls# file 196

.printss# file 196
.PROBE command 201
PTDNOISE
  input syntax 168
  .MEASURE 170
  output file format 170
  output syntax 169
  overview 166
  syntax 168
PTDNOISE command 166
.PZ statement 318

R

random signals, characteristcs 258
RC
  line modeling 342
rcxt_divider configuration option 9
restricting output 11
results 139
reusing simulation output 18, 20
RF
  demo files 22
RFdemo files 22
rise time
  example 16
  verify 15

S

S parameter extraction
  large-signal 191
  power-dependent 183
  small-signal 191
SCALE parameter 322, 345
SDE 237, 247
s-domain 330, 335
separating AM-PM noise 111
SETUP time verification 17
Shooting Newton
  driven phase frequency circuit example 66
  overview 61
  ring oscillator example 73
shot noise 268
SIM_ACTIVE option 208, 212, 213, 214
SIM_DSPF option 208, 209
SIM_DSPF_ACTIVE option 208, 212
SIM_DSPF_INSERROR option 214
SIM_DSPF_LUMPCAPS option 213
SIM_DSPF_MAX_ITER option 213
SIM_DSPF_RAIL option 213
SIM_DSPF_SCALEC option 213
SIM_DSPF_SCALER option 213
SIM_DSPF_VTOL option 212
SIM_LA option 209, 230, 233
SIM_LA_FREQ option 233
SIM_LA_MAXR option 233
SIM_LA_MINC option 233
SIM_LA_MINMODE option 233
SIM_LA_TIME option 233
SIM_LA_TOL option 233
SIM_POWERDC_ACCURACY option 19
SIM_POWERED_HSPICE option 19
SIM_POWERPOST option 21
SIM_POWERSTART option 21
SIM_SPEF option 208
SIM_SPEF_ACTIVE option 212
SIM_SPEF_INSERROR option 214
SIM_SPEF_LUMPCAPS option 213
SIM_SPEF_MAX_ITER option 213
SIM_SPEF_PARVALUE option 214
SIM_SPEF_RAIL option 213
SIM_SPEF_SCALEC option 213
SIM_SPEF_SCALER option 213
SIM_SPEF_VTOL option 212
simulation
  reducing time 352
skip_nrd_nrs configuration option
  configuration options
    skip_nrd_nrs 9
slew rate
  example 15
  verification 15
small-signal noise parameter extraction 191
small-signal S parameter extraction 191
SN steady-state time domain analysis 61
SNAC
  input syntax 141
  output data files 144
SNFT 64
SNNOISE 159
  input syntax 160
  output data files 163
SNOSC 100

SNXF 177
source
  controlled 282, 283
  element types 283
spur (spurious) signals 104
.ss# file 196
statement 199, 200
  .ENV 198
statements
  .CHECK EDGE 16
  .CHECK FALL 16
  .CHECK GLOBAL_LEVEL 15
  .CHECK HOLD 17
  .CHECK IRDROP 18
  .CHECK RISE 15
  .CHECK SETUP 17
  .CHECK SLEW 15
  .HBXF 173
  .POWER 20
  .POWERDC 18
  .PTDNOISE 166
  .SURGE 21
  .TRAN 20
steady state time domain analysis, Shooting Newton 61
Stochastic Differential Equation, TRANNOISE method 237
strobejitter 170
.SURGE statement 21
switched capacitor 351, 352

T

TC1, TC2 element parameters 322
temperature
  coefficients 322
.TF statement 293
thermal noise 264
TIE 237
TIE, jitter mreasurement 252
time
  domain
    to frequency domain 317
  maximum 321
  resolution 320
time domain steady state analysis 61
time interval error (TIE) 237
.TRAN statement 20
TRANNOISE
  jitter measurements 237
  SDE method 247
transfer function
  algebraic 281
  analysis 293
  coefficients 340
  filters 340
  frequency domain 331
  general form 330, 338
  inverse Laplace transform 342
  reduced form 339
  voltage gain 334
transformer, behavioral 298
transforms, Fourier 320
transient
  modeling 318
transient modeling 317
transient noise jitter measurements 252
two-tone HB 137

U

unit_atto configuration option 9

V

v_supply configuration option 9
VCO 93
VCO *See* voltage controlled oscillator
Verilog-A support 4
voltage
  variable capacitance model 308
voltage-controlled
  elements 283
  oscillator 305
  voltage source 304
VVCAP model 308

W

warnings
  MAXF ignored 324
waveform viewing 7
WaveView 7
wildcard uses 10
wildcard_left_range configuration option 10
wildcard_match_all configuration option 10
wildcard_match_one configuration option 10

wildcard_right_range configuration option 10

Y

Y parameter
  line model 349
  modeling 346
  network 347

Z

z transform function 322