Contents

Related Documents ................................................................. 6
Typographic and Syntax Conventions ......................................... 6
References .................................................................................. 7

1 Introducing the Virtuoso Spectre Circuit Simulator .................. 8
   Improvements over SPICE ........................................................ 9
      Improved Capacity ............................................................. 9
      Improved Accuracy ......................................................... 9
      Improved Speed ............................................................ 10
      Improved Reliability ....................................................... 11
      Improved Models .......................................................... 12
   Spectre Usability Features and Customer Service ..................... 12
   Analog HDLs ........................................................................ 13
   RF Capabilities ..................................................................... 13
   Mixed-Signal Simulation ....................................................... 15
   Environments ........................................................................ 15

2 Command Options ................................................................. 17
   Default Values ...................................................................... 22
   Default Parameter Values ..................................................... 22

3 Analysis Statements .............................................................. 24
   AC Analysis (ac) .................................................................... 26
   Alter a Circuit, Component, or Netlist Parameter (alter) ............ 29
   Alter Group (altergroup) ....................................................... 29
   Check Parameter Values (check) ............................................ 31
   Checklimit Analysis (checklimit) ........................................... 32

November 2004 1
DC Analysis (dc) ................................................................. 33
DC Device Matching Analysis (dcmatch) ................................. 37
Envelope Following Analysis (envlp) ........................................ 41
Circuit Information (info) .................................................. 46
Monte Carlo Analysis (montecarlo) ......................................... 48
Noise Analysis (noise) ....................................................... 60
Immediate Set Options (options) ........................................... 64
Periodic AC Analysis (pac) .................................................. 74
Periodic Distortion Analysis (pdisto) ....................................... 78
Periodic Noise Analysis (pnoise) ........................................... 86
Periodic S-Parameter Analysis (psp) ....................................... 92
Periodic Steady-State Analysis (pss) ...................................... 98
Periodic Transfer Function Analysis (pxf) ............................... 112
PZ Analysis (pz) ............................................................... 117
Quasi-Periodic AC Analysis (qpac) ......................................... 122
Quasi-Periodic Noise Analysis (qpnoise) .................................. 125
Quasi-Periodic S-Parameter Analysis (qpssp) .......................... 131
Quasi-Periodic Steady State Analysis (qpss) ............................ 137
Quasi-Periodic Transfer Function Analysis (qpxf) .................... 145
Deferred Set Options (set) ................................................... 149
Shell Command (shell) ...................................................... 153
S-Parameter Analysis (sp) .................................................... 154
Stability Analysis (stb) ....................................................... 158
Sweep Analysis (sweep) ...................................................... 164
Time-Domain Reflectometer Analysis (tdr) ............................. 166
Transient Analysis (tran) .................................................... 168
Transfer Function Analysis (xf) ............................................. 177

4

Syntax ................................................................................. 182
Using analogmodel for Model Passing (analogmodel) .................. 184
Behavioural Source Use Model (bsource) ................................. 186
Checkpoint - Restart (checkpoint) ........................................... 191
Configuring CMI Shared Objects (cmiconfig) ......................... 192
Built-in Mathematical and Physical Constants (constants) ........... 193
## Spectre Circuit Simulator Reference

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convergence Difficulties (convergence)</td>
<td>195</td>
</tr>
<tr>
<td>Export (export)</td>
<td>196</td>
</tr>
<tr>
<td>Expressions (expressions)</td>
<td>197</td>
</tr>
<tr>
<td>User Defined Functions (functions)</td>
<td>201</td>
</tr>
<tr>
<td>Global Nodes (global)</td>
<td>202</td>
</tr>
<tr>
<td>Initial Conditions (ic)</td>
<td>203</td>
</tr>
<tr>
<td>The Structural if-statement (if)</td>
<td>203</td>
</tr>
<tr>
<td>Include File (include)</td>
<td>205</td>
</tr>
<tr>
<td>Spectre Netlist Keywords (keywords)</td>
<td>206</td>
</tr>
<tr>
<td>Library - Sectional Include (library)</td>
<td>210</td>
</tr>
<tr>
<td>Node Sets (nodeset)</td>
<td>213</td>
</tr>
<tr>
<td>Parameter Soft Limits (param_limits)</td>
<td>213</td>
</tr>
<tr>
<td>Netlist Parameters (parameters)</td>
<td>216</td>
</tr>
<tr>
<td>Parameter Set - Block of Data (paramset)</td>
<td>218</td>
</tr>
<tr>
<td>Output Selections (save)</td>
<td>219</td>
</tr>
<tr>
<td>Sensitivity Analyses (sens)</td>
<td>220</td>
</tr>
<tr>
<td>SpectreHDL Usage and Language Summary (spectrehdl)</td>
<td>221</td>
</tr>
<tr>
<td>SpectreRF Summary (spectrerf)</td>
<td>230</td>
</tr>
<tr>
<td>Subcircuit Definitions (subckt)</td>
<td>231</td>
</tr>
<tr>
<td>Verilog-A Usage and Language Summary (veriloga)</td>
<td>235</td>
</tr>
<tr>
<td></td>
<td>244</td>
</tr>
</tbody>
</table>

## A

### References

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>References</td>
<td>245</td>
</tr>
</tbody>
</table>
Preface

This manual assumes that you are familiar with the development, design, and simulation of integrated circuits and that you have some familiarity with SPICE simulation. It contains information about the Virtuoso® Spectre® circuit simulator.

Spectre is an advanced circuit simulator that simulates analog and digital circuits at the differential equation level. The simulator uses improved algorithms that offer increased simulation speed and greatly improved convergence characteristics over SPICE. Besides the basic capabilities, the Spectre circuit simulator provides significant additional capabilities over SPICE. SpectreHDL (Spectre High-Level Description Language) and Verilog®-A use functional description text files (modules) to model the behavior of electrical circuits and other systems. Virtuoso® SpectreRF Simulation Option adds several new analyses that support the efficient calculation of the operating point, transfer function, noise, and distortion of common RF and communication circuits, such as mixers, oscillators, sample holds, and switched-capacitor filters.

This preface discusses the following topics:

- Related Documents on page Preface-6
- Typographic and Syntax Conventions on page Preface-6
- References on page Preface-7
Related Documents

The following can give you more information about the Spectre circuit simulator and related products:

- To learn more about the equations used in the Spectre circuit simulator, consult the Virtuoso Spectre Circuit Simulator Device Model Equations manual.

- The Spectre circuit simulator is often run within the Cadence® analog circuit design environment, under the Cadence® design framework II. To see how the Spectre circuit simulator is run under the analog circuit design environment, read the Virtuoso Analog Design Environment User Guide.

- For more information about using the Spectre circuit simulator with Verilog-A, see the Verilog-A Language Reference manual.

- If you want to see how SpectreRF is run under the analog circuit design environment, read Virtuoso SpectreRF Simulation Option User Guide.

- For more information about RF theory, see Virtuoso SpectreRF Simulation Option Theory.

- For more information about how you work with the design framework II interface, see Design Framework II Help.

- For more information about specific applications of Spectre analyses, see The Designer’s Guide to SPICE & Spectre.

Typographic and Syntax Conventions

This list describes the syntax conventions used for the Spectre circuit simulator.

**literal** Nonitalic words indicate keywords that you must enter literally. These keywords represent command (function, routine) or option names, file names and paths, and any other sort of type-in commands.

**argument** Words in italics indicate user-defined arguments for which you must substitute a name or a value. (The characters before the underscore (_) in the word indicate the data types that this argument can take. Names are case sensitive.

---

|Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character.

[] Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices. You can choose one argument from the list.

{} Braces are used with OR-bars and enclose a list of choices. You must choose one argument from the list.

... Three dots (...) indicate that you can repeat the previous argument. If you use them with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more.

⚠️ Important

The language requires many characters not included in the preceding list. You must enter required characters exactly as shown.

References

Text within brackets ([ ]) are references. See Appendix A, “References,” for more detailed information.
Introducing the Virtuoso Spectre Circuit Simulator

This chapter discusses the following:

- **Improvements over SPICE** on page 9
- **Analog HDLs** on page 13
- **RF Capabilities** on page 13
- **Mixed-Signal Simulation** on page 15
- **Environments** on page 15

The Virtuoso® Spectre® circuit simulator is a modern circuit simulator that uses direct methods to simulate analog and digital circuits at the differential equation level. The basic capabilities of the Spectre circuit simulator are similar in function and application to SPICE, but the Spectre circuit simulator is not descended from SPICE. The Spectre and SPICE simulators use the same basic algorithms—such as implicit integration methods, Newton-Raphson, and direct matrix solution—but every algorithm is newly implemented. Spectre algorithms, the best currently available, give you an improved simulator that is faster, more accurate, more reliable, and more flexible than previous SPICE-like simulators.
Improvements over SPICE

The Spectre circuit simulator has many improvements over SPICE.

Improved Capacity

The Spectre circuit simulator can simulate larger circuits than other simulators because its convergence algorithms are effective with large circuits, because it is fast, and because it is frugal with memory and uses dynamic memory allocation. For large circuits, the Spectre circuit simulator typically uses less than half as much memory as SPICE.

Improved Accuracy

Improved component models and core simulator algorithms make the Spectre circuit simulator more accurate than other simulators. These features improve Spectre accuracy:

- Advanced metal oxide semiconductor (MOS) and bipolar models
  - The Spectre BSIM 3v3 is a physics-based metal-oxide semiconductor field effect transistor (MOSFET) model for simulating analog circuits.
  - The Spectre models include the MOS0 model, which is even simpler and faster than MOS1 for simulating noncritical MOS transistors in logic circuits and behavioral models, MOS 9, EKV, BTA-HVMOS, BTA-SOI, VBIC95, TOM2, and HBT.

- Charge-conserving models
  - The capacitance-based nonlinear MOS capacitor models used in many SPICE derivatives can create or destroy small amounts of charge on every time step. The Spectre circuit simulator avoids this problem because all Spectre models are charge-conserving.

- Improved Fourier analyzer
  - The Spectre circuit simulator includes a two-channel Fourier analyzer that is similar in application to the SPICE .FOURIER statement but is more accurate. The Spectre simulator’s Fourier analyzer has greater resolution for measuring small distortion products on a large sinusoidal signal. Resolution is normally greater than 120 dB. Furthermore, the Spectre simulator’s Fourier analyzer is not subject to aliasing, a common error in Fourier analysis. As a result, the Spectre simulator can accurately compute the Fourier coefficients of highly discontinuous waveforms.

- Better control of numerical error
Many algorithms in the Spectre circuit simulator are superior to their SPICE counterparts in avoiding known sources of numerical error. The Spectre circuit simulator improves the control of local truncation error in the transient analysis by controlling error in the voltage rather than the charge.

In addition, the Spectre circuit simulator directly checks Kirchhoff’s Current Law (also known as Kirchhoff’s Flow Law) at each time step, improves the charge-conservation accuracy of the Spectre circuit simulator, and eliminates the possibility of false convergence.

- Superior time-step control algorithm

  The Spectre circuit simulator provides an adaptive time-step control algorithm that reliably follows rapid changes in the solution waveforms. It does so without limiting assumptions about the type of circuit or the magnitude of the signals.

- More accurate simulation techniques

  Techniques that reduce reliability or accuracy, such as device bypass, simplified models, or relaxation methods, are not used in the Spectre circuit simulator.

- User control of accuracy tolerances

  For some simulations, you might want to sacrifice some degree of accuracy to improve the simulation speed. For other simulations, you might accept a slower simulation to achieve greater accuracy. With the Spectre circuit simulator, you can make such adjustments easily by setting a single parameter.

### Improved Speed

The Spectre circuit simulator is designed to improve simulation speed. The Spectre circuit simulator improves speed by increasing the efficiency of the simulator rather than by sacrificing accuracy.

- Faster simulation of small circuits

  The average Spectre simulation time for small circuits is typically two to three times faster than SPICE. The Spectre circuit simulator can be over 10 times faster than SPICE when SPICE is hampered by discontinuity in the models or problems in the code. Occasionally, the Spectre circuit simulator is slower when it finds ringing or oscillation that goes unnoticed by SPICE. This can be improved by setting the `macromodels` option to `yes`.

- Faster simulation for large circuits
The Spectre circuit simulator is generally two to five times faster than SPICE with large circuits because it has fewer convergence difficulties and because it rapidly factors and solves large sparse matrices.

**Improved Reliability**

The Spectre circuit simulator offers you the following improvements in reliability:

- **Improved convergence**
  Spectre proprietary algorithms ensure convergence of the Newton-Raphson algorithm in the DC analysis. The Spectre circuit simulator virtually eliminates the convergence problems that earlier simulators had with transient simulation.

- **Helpful error and warning messages**
  The Spectre circuit simulator detects and notifies you of many conditions that are likely to be errors. For example, the Spectre circuit simulator warns of models used in forbidden operating regions, of incorrectly wired circuits, and of erroneous component parameter values. By identifying such common errors, the Spectre circuit simulator saves you the time required to find these errors with other simulators.

  The Spectre circuit simulator lets you define soft parameter limits and sends you warnings if parameters exceed these limits.

- **Thorough testing**
  Automated tests, which include over 1,000 test circuits, are constantly run on all hardware platforms to ensure that the Spectre circuit simulator is consistently reliable and accurate.

- **Benchmark suite**
  There is an independent collection of SPICE netlists that are difficult to simulate. You can obtain these circuits from the Microelectronics Center of North Carolina (MCNC) if you have File Transfer Protocol (FTP) access on the Internet. You can also get information about the performance of several simulators with these circuits.

  The Spectre circuit simulator has successfully simulated all of these circuits. Sometimes the netlists required minor syntax corrections, such as inserting balance parentheses, but circuits were never altered, and options were never changed to affect convergence.
Improved Models

The Spectre circuit simulator has MOSFET Level 0–3, BSIM1, BSIM2, BSIM3, BSIM 3v3, EKV, MOS9, JFET, TOM2, GaAs MESFET, BJT, VBIC, HBT, diode, and many other models. It also includes the temperature effects, noise, and MOSFET intrinsic capacitance models.

The Spectre Compiled Model Interface (CMI) option lets you integrate new devices into the Spectre simulator using a very powerful, efficient, and flexible C language interface. This CMI option, the same one used by Spectre developers, lets you install proprietary models.

Spectre Usability Features and Customer Service

The following features and services help you use the Spectre circuit simulator easily and efficiently:

- You can use Spectre soft limits to catch errors created by typing mistakes.
- Spectre diagnosis mode, available as an options statement parameter, gives you information to help diagnose convergence problems.
- You can run the Spectre circuit simulator standalone or run it under the Cadence analog design environment. To see how the Spectre circuit simulator is run under the analog design environment, read the Virtuoso Analog Design Environment User Guide. You can also run the Spectre circuit simulator in the Composer-to-Spectre direct simulation environment. The environment provides a graphical user interface for running the simulation.
- The Spectre circuit simulator gives you an online help system. With this system, you can find information about any parameter associated with any Spectre component or analysis. You can also find articles on other topics that are important to use the Spectre circuit simulator effectively.
- The Spectre circuit simulator also includes a waveform display tool, WaveScan, to use to display simulation results. For more information about AWD, see the WaveScan User Guide.
- If you experience a stubborn convergence or accuracy problem, you can send the circuit to Customer Support to get help with the simulation. For current phone numbers and e-mail addresses, see the following web site: http://sourcelink.cadence.com/supportcontacts.html.
Analog HDLs

The Spectre circuit simulator works with two analog high-level description languages (AHDLs): SpectreHDL and Verilog®-A. These languages are part of the Spectre Verilog-A Simulation option. SpectreHDL is proprietary to Cadence and is provided for backward compatibility. The Verilog-A language is an open standard, which was based upon SpectreHDL. The Verilog-A language is preferred because it is upward compatible with Verilog-AMS, a powerful and industry-standard mixed-signal language.

Both languages use functional description text files (modules) to model the behavior of electrical circuits and other systems. Each programming language allows you to create your own models by simply writing down the equations. The AHDL lets you describe models in a simple and natural manner. This is a higher level modeling language than previous modeling languages, and you can use it without being concerned about the complexities of the simulator or the simulator algorithms. In addition, you can combine AHDL components with Spectre built-in primitives.

Both languages let designers of analog systems and integrated circuits create and use modules that encapsulate high-level behavioral descriptions of systems and components. The behavior of each module is described mathematically in terms of its terminals and external parameters applied to the module. Designers can use these behavioral descriptions in many disciplines (electrical, mechanical, optical, and so on).

Both languages borrow many constructs from Verilog and the C programming language. These features are combined with a minimum number of special constructs for behavioral simulation. These high-level constructs make it easier for designers to use a high-level description language for the first time.

RF Capabilities

Virtuoso® SpectreRF Simulation Option adds several new analyses that support the efficient calculation of the operating point, transfer function, noise, and distortion of common analog and RF communication circuits, such as mixers, oscillators, sample and holds, and switched-capacitor filters.

SpectreRF adds four types of analyses to the Spectre simulator. The first is periodic steady-state (PSS) analysis, a large-signal analysis that directly computes the periodic steady-state response of a circuit. With PSS, simulation times are independent of the time constants of the circuit, so PSS can quickly compute the steady-state response of circuits with long time constants, such as high-Q filters and oscillators.
You can also embed a PSS analysis in a sweep loop (referred to as an SPSS analysis in the Cadence analog design environment), which allows you to easily determine harmonic levels as a function of input level or frequency, making it easy to measure compression points, intercept points, and voltage-controlled oscillator (VCO) linearity.

The second new type of analysis is the periodic small-signal analysis. After completing a PSS analysis, SpectreRF can predict the small-signal transfer functions and noise of frequency translation circuits, such as mixers or periodically driven circuits such as oscillators or switched-capacitor or switched-current filters. The periodic small-signal analyses—periodic AC (PAC) analysis, periodic transfer function (PXF) analysis, and periodic noise (Pnoise) analysis—are similar to Spectre's AC, XF, and Noise analyses, but the traditional small-signal analyses are limited to circuits with DC operating points. The periodic small-signal analyses can be applied to circuits with periodic operating points, such as the following:

- Mixers
- VCOs
- Switched-current filters
- Phase/frequency detectors
- Frequency multipliers
- Chopper-stabilized amplifiers
- Oscillators
- Switched-capacitor filters
- Sample and holds
- Frequency dividers
- Narrow-band active circuits

The third SpectreRF addition to Spectre functionality is periodic distortion (PDISTO) analysis. PDISTO analysis directly computes the steady-state response of a circuit driven with a large periodic signal, such as an LO (local oscillation) or a clock, and one or more tones with moderate level. With PDISTO, you can model periodic distortion and include harmonic effects. PDISTO computes both a large signal, the periodic steady-state response of the circuit, and also the distortion effects of a specified number of moderate signals, including the distortion effects of the number of harmonics that you choose. This is a common scenario when trying to predict the intermodulation distortion of a mixer, amplifier, or a narrow-band filter. In this analysis, the tones can be large enough to create significant distortion, but not so large as to cause the circuit to switch or clip. The frequencies of the tones need not be periodically related to each other or to the large signal LO or clock. Thus, you can make the
tone frequencies very close to each other without penalty, which allows efficient computation of intermodulation distortion of even very narrow band circuits.

The fourth analysis that SpectreRF adds to the Spectre circuit simulator is the envelope-following analysis. This analysis computes the envelope response of a circuit. The simulator automatically determines the clock period by looking through all the sources with the specified name. Envelope-following analysis is most efficient for circuits where the modulation bandwidth is orders of magnitude lower than the clock frequency. This is typically the case, for example, in circuits where the clock is the only fast varying signal and other input signals have a spectrum whose frequency range is orders of magnitude lower than the clock frequency. For another example, the down conversion of two closely placed frequencies can also generate a slow-varying modulation envelope. The analysis generates two types of output files, a voltage versus time (td) file, and an amplitude/phase versus time (fd) file for each specified harmonic of the clock fundamental.

In summary, with periodic small-signal analyses, you apply a small signal at a frequency that might not be harmonically related (noncommensurate) to the periodic response of the undriven system, the clock. This small signal is assumed to be small enough so that the circuit is unaffected by its presence.

With PDISTO, you can apply one or two additional signals at frequencies not harmonically related to the large signal, and these signals can be large enough to drive the circuit to behave nonlinearly.

For complex nonlinear circuits, hand calculation of noise or transfer function is virtually impossible. Without SpectreRF, these circuits must be breadboarded to determine their performances. The SpectreRF simulator eliminates unnecessary breadboarding, saving time.

**Mixed-Signal Simulation**

You can use the Spectre circuit simulator coupled with the Verilog®-XL simulator in the Cadence analog design environment to simulate mixed analog and digital circuits efficiently. This mixed-signal simulation solution can easily handle complex designs with tens of thousands of transistors and tens of thousands of gates. The digital Verilog data can come from the digital designer as either an RTL block or gates out of synthesis.

**Environments**

The Spectre circuit simulator is fully integrated into the Cadence® design framework II for the Cadence analog design environment and also into the Cadence analog workbench design
You can also use the Spectre circuit simulator by itself with several different output format options.

Assura® interactive verification, Dracula® distributed multi-CPU option, and Assura hierarchical physical verification produce a netlist that can be read into the Spectre circuit simulator. However, only interactive verification when used with the Cadence analog design environment automatically attaches the stimulus file. All other situations require a stimulus file as well as device models.
Command Options

This chapter lists the options you can use with the `spectre` command and gives a brief description of each. It also discusses the following topics:

- Default Values on page 22
- Default Parameter Values on page 22

The `spectre` command takes the following syntax at the command line:

```
spectre options inputfile
```

**Note:** The Virtuoso® Spectre® circuit simulator reads default values for all the command line arguments marked with a dagger (†) from the UNIX environment variable `%S_DEFAULTS`.

- `-help` Lists command options and available components and analyses. You can use `-h` as an abbreviation of `-help`.

- `-help name` Gives a synopsis of the device or analysis `name`. If `name` is `all`, the synopses for all components and analyses are given. You can use `-h` as an abbreviation of `-help`.

- `-helpsort name` Gives a synopsis of the device or analysis `name` and sorts all the parameters by name. You can use `-hs` as an abbreviation of `-helpsort`.

- `-helpfull name` Gives a full synopsis of the component or analysis `name`, including parameter types and range limits. You can use `-hf` as an abbreviation of `-helpfull`.

- `-helpsortfull name` Gives a full synopsis of component or analysis `name`, including parameter types and range limits. Sorts all parameters by name. You can use `-hsf` as an abbreviation of `-helpsortfull`.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-param†</code></td>
<td>Does not read the file containing the suggested parameter range limits. You can use <code>-p</code> as an abbreviation of <code>-param</code>.</td>
</tr>
<tr>
<td><code>+param file†</code></td>
<td>Reads <code>file</code> for the suggested parameter range limits. You can use <code>+p</code> as an abbreviation of <code>+param</code>.</td>
</tr>
<tr>
<td><code>-log†</code></td>
<td>Does not copy all messages to a file. You can use <code>-l</code> as an abbreviation of <code>-log</code>.</td>
</tr>
<tr>
<td><code>+log file†</code></td>
<td>Copies all messages to <code>file</code>. You can use <code>+l</code> as an abbreviation of <code>+log</code>.</td>
</tr>
<tr>
<td><code>=log file†</code></td>
<td>Sends all messages to <code>file</code>. You can use <code>=l</code> as an abbreviation of <code>=log</code>.</td>
</tr>
<tr>
<td><code>-raw raw†</code></td>
<td>Puts results in a file or directory named <code>raw</code>. In <code>raw</code>, <code>%C</code> is replaced by a circuit name. You can use <code>-r</code> as an abbreviation of <code>-raw</code>.</td>
</tr>
<tr>
<td><code>-format fmt†</code></td>
<td>Produces raw data in the format <code>fmt</code>. You can use <code>-f</code> as an abbreviation of <code>-format</code>. Possible values for <code>fmt</code> are <code>nutbin</code>, <code>nutascii</code>, <code>wsfbin</code>, <code>wsfascii</code>, <code>psfbin</code>, <code>psfascii</code>, or <code>awb</code>.</td>
</tr>
<tr>
<td><code>+checkpoint†</code></td>
<td>Turns on the checkpoint capability. You can use <code>+cp</code> as an abbreviation of <code>+checkpoint</code>.</td>
</tr>
<tr>
<td><code>-checkpoint†</code></td>
<td>Turns off the checkpoint capability. You can use <code>-cp</code> as an abbreviation of <code>-checkpoint</code>.</td>
</tr>
<tr>
<td><code>-recover†</code></td>
<td>Does not restart the simulation, even if a checkpoint file exists. You can use <code>-rec</code> as an abbreviation of <code>-recover</code>.</td>
</tr>
<tr>
<td><code>+recover†</code></td>
<td>Restarts the simulation from a checkpoint file, if it exists. You can use <code>+rec</code> as an abbreviation of <code>+recover</code>.</td>
</tr>
<tr>
<td><code>-cols N†</code></td>
<td>Sets screen width in characters to <code>N</code>. You can use <code>-c</code> as an abbreviation of <code>-cols</code>. If not set, the Spectre simulator determines the screen width automatically.</td>
</tr>
<tr>
<td><code>-env env</code></td>
<td>Calls the Spectre simulator from the <code>env</code> simulation environment. Possible values for <code>env</code> are <code>artist2</code>, <code>artist4</code>, <code>awb</code>, <code>edge</code>, <code>opus</code>, or <code>solo</code>.</td>
</tr>
</tbody>
</table>
-%X  In quoted strings within the netlist, replaces \%X with nothing where \(X\) is any uppercase or lowercase letter.

+%X string†  In quoted strings within the netlist, replaces \%X with \texttt{string}, where \(X\) is an uppercase or lowercase letter. You can modify the string by using the :\(X\) operators.

+error†  Prints error messages.

-error†  Does not print error messages.

+varedefnerror  Prints error messages if Verilog-A modules are redefined.

+warn†  Prints warning messages.

-warn†  Does not print warning messages.

+note  Prints notices.

-note  Does not print notices.

+info†  Prints informational messages.

-info†  Does not print informational messages.

+debug†  Prints debugging messages.

-debug†  Does not print debugging messages.

-slave <cmd>  Starts the attached simulator using the command \texttt{cmd}.

-slvhost <hostname>  Runs the attached simulator on machine \texttt{hostname}. Defaults to local machine.

-V  Prints version information.

-W  Prints subversion information.

-cmiversion  Prints CMI version information.

-cmiconfig file  Reads \texttt{file} for information to modify the existing CMI configuration.

-alias <name>†  Gives \texttt{name} to the license manager as the name of the simulator.

-E†  Runs the C preprocessor on an input file. In SPICE mode, the first line in the file must be a comment.
## Spectre Circuit Simulator Reference
### Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-D&lt;x&gt;</code></td>
<td>Defines string ( x ) and runs the C preprocessor.</td>
</tr>
<tr>
<td><code>-D&lt;x&gt;=y&gt;</code></td>
<td>Defines string ( x ) to be ( y ) and runs the C preprocessor.</td>
</tr>
<tr>
<td><code>-U&lt;x&gt;</code></td>
<td>Undefines string ( x ) and runs the C preprocessor.</td>
</tr>
<tr>
<td><code>-I&lt;dir&gt;</code></td>
<td>Runs the C preprocessor and searches the directory ( dir ) for include files.</td>
</tr>
<tr>
<td><code>-spp</code></td>
<td>Do not run the Spice netlist reader on the input file.</td>
</tr>
<tr>
<td><code>+spp</code></td>
<td>Run the Spice netlist reader on the input file. Use <code>+spp -sppbin</code> on the command line option to read other spp binaries.</td>
</tr>
<tr>
<td><code>-sppbin file</code></td>
<td>Specify the path to nondefault spp binary. Default provided.</td>
</tr>
<tr>
<td><code>+sensdata &lt;file&gt;</code></td>
<td>Sends the sensitivity analyses data to ( file ).</td>
</tr>
<tr>
<td><code>+multithread</code></td>
<td>Turns on multithread capability. Spectre automatically detects the number of processors and selects the proper number of threads to use. You can use <code>+mt</code> as an abbreviation of <code>+multithread</code>.</td>
</tr>
<tr>
<td><code>+multithread=N</code></td>
<td>Turns on multithread capability with ( N ) threads. You can use <code>+mt</code> as an abbreviation of <code>+multithread</code>.</td>
</tr>
<tr>
<td><code>-multithread</code></td>
<td>Turns off multithread capability. By default, multithreading is turned off. You can use <code>+mt</code> as an abbreviation of <code>+multithread</code>.</td>
</tr>
<tr>
<td><code>-interactive</code></td>
<td>Run in the non-interactive mode, that is, process the input file and then return. You can use <code>+inter</code> as an abbreviation of <code>+interactive</code>.</td>
</tr>
<tr>
<td><code>+interactive</code></td>
<td>Run in the default interactive mode. You can use <code>+inter</code> as an abbreviation of <code>+interactive</code>.</td>
</tr>
<tr>
<td><code>+interactive=type</code></td>
<td>Run in the interactive mode of the type specified. Possible values for ( type ) are <code>skill</code> or <code>mpsc</code>.</td>
</tr>
<tr>
<td><code>+mpssession=sessionName</code></td>
<td>The <code>sessionName</code> for an interactive session using multiprocess SKILL (MPS). This option is necessary for <code>+interactive=mpsc</code> and implies <code>+interactive=mpsc</code>.</td>
</tr>
</tbody>
</table>
+mpshost=sessionHost

The sessionHost for an interactive session using MPS.

-mdlcontrol

Do not run with the MDL control file. You can use -mdl as an abbreviation of -mdlcontrol.

+mdlcontrol

Runs with the default MDL control file. You can use -mdl as an abbreviation of -mdlcontrol.

=mdlcontrol file

Specifies the location of the MDL control file to run. You can use -mdl as an abbreviation of -mdlcontrol.

-checklimitfile file

Writes assert violations to file. In file, %C is replaced by the circuit name. You can use -cl as an abbreviation of -checklimitfile.

-docheklimit

Turns off the checklimit capability. You can use -docl as an abbreviation of -docheklimit.

+docheklimit

Turns on the checklimit capability. You can use +docl as an abbreviation of +docheklimit.

+lqtimeout value

Turns on license queuing. value specifies how many seconds to wait for a license. Specifying 0 means wait until license is available. You can use +lqt as an abbreviation of +lqtimeout.

+lqsleep value

Sleep time between two tries to check out a license when queuing. Setting the value to a positive number will override the default sleep time of 30 seconds. You can use +lqs as an abbreviation of +lqsleep.

If you do not specify an input file, the Spectre simulator reads from standard input. When +/- pairs of spectre command options are available, the default is the first value given in the previous list. For further information about the percent code options, +% and -%, see Chapter 11, “Managing Files,” in the Spectre Circuit Simulator User Guide.

Note: To remain consistent with the C preprocessor, there is no space between the preprocessor flags (D, U, I) and their arguments. The C preprocessor is available on UNIX systems only and requires that the first line of the file (the SPICE title line) begin with a comment character (* or //).
**Default Values**

The Spectre simulator reads default values for all the command line arguments marked with a dagger (†) from the UNIX environment variable %S_DEFAULTS. The name of the simulator as called replaces %S. Typically, this name is spectre, and the Spectre simulator looks for spectre_DEFAULTS. However, the name can be different if you move the executable to a file with a different name or if you call the Spectre simulator through a symbolic or hard link with a different name. This feature lets you set different default values for each name you use to call the Spectre simulator.

If the variable %S_DEFAULTS does not exist, SPECTRE_DEFAULTS is used instead. The command line arguments always override any specifications from the options statement in the circuit file. The options statement specifications, in turn, override any specifications in the environment variable.

**Default Parameter Values**

Many Spectre parameters have default values, and sometimes you will need to know them so you can determine whether they are acceptable for your simulation. You can find the default values for component, analysis, and control statement parameters by consulting the documentation for the statement in Spectre online help (spectre -h). Values given for parameters in the online help are the default values.

The following examples show you some defaults for different types of parameters from the Spectre online help:

```
nf=1.0         Forward emission coefficient
etchc=etchm    Narrowing due to etching for capacitances
homotopy=all   Method used when there is no convergence on initial attempt of DC analysis; possible values are none, gmin, source, dptran, ptran, or all
rawfile="%C:r.raw"   Output raw data filename
```

In this example, the default values for nf, etchc, homotopy, and rawfile are a real number (1.0), the value of a different parameter (etch), an enumerated type (all), and a character string with a percent code and a colon modifier that gives Spectre instructions for creating the output filename ("%C:r.raw").
For more information about percent codes and colon modifiers, see “Description of Spectre Predefined Percent Codes,” “Customizing Percent Codes,” and “Creating Filenames from Parts of Input Filenames” in the Spectre Circuit Simulator User Guide.
Analysis Statements

This chapter discusses the following topics:

- **AC Analysis (ac)** on page 26
- **Alter a Circuit, Component, or Netlist Parameter (alter)** on page 29
- **Alter Group (altergroup)** on page 29
- **Check Parameter Values (check)** on page 31
- **Checklimit Analysis (checklimit)** on page 32
- **DC Analysis (dc)** on page 33
- **DC Device Matching Analysis (dcmatch)** on page 37
- **Envelope Following Analysis (enlvp)** on page 41
- **Circuit Information (info)** on page 46
- **Monte Carlo Analysis (montecarlo)** on page 48
- **Noise Analysis (noise)** on page 60
- **Immediate Set Options (options)** on page 64
- **Periodic AC Analysis (pac)** on page 74
- **Periodic Distortion Analysis (pdisto)** on page 78
- **Periodic Noise Analysis (pnoise)** on page 86
- **Periodic S-Parameter Analysis (psp)** on page 92
- **Periodic Steady-State Analysis (pss)** on page 98
- **Periodic Transfer Function Analysis (pxf)** on page 112
- **PZ Analysis (pz)** on page 117
- **Quasi-Periodic AC Analysis (qpac)** on page 122
Spectre Circuit Simulator Reference
Analysis Statements

- Quasi-Periodic Noise Analysis (qpnoise) on page 125
- Quasi-Periodic S-Parameter Analysis (qpsp) on page 131
- Quasi-Periodic Steady State Analysis (qpss) on page 137
- Quasi-Periodic Transfer Function Analysis (qpxf) on page 145
- Deferred Set Options (set) on page 149
- Shell Command (shell) on page 153
- S-Parameter Analysis (sp) on page 154
- Stability Analysis (stb) on page 158
- Sweep Analysis (sweep) on page 164
- Time-Domain Reflectometer Analysis (tdr) on page 166
- Transient Analysis (tran) on page 168
- Transfer Function Analysis (xf) on page 177
AC Analysis (ac)

Description
The AC analysis linearizes the circuit about the DC operating point and computes the response to a given small sinusoidal stimulus.

Spectre can perform the analysis while sweeping a parameter. The parameter can be frequency, temperature, component instance parameter, component model parameter, or netlist parameter. If changing a parameter affects the DC operating point, the operating point is recomputed on each step. You can sweep the circuit temperature by giving the parameter name as temp with no dev or mod parameter. You can sweep a netlist parameter by giving the parameter name with no dev, or mod parameter. After the analysis has completed, the modified parameter returns to its original value.

Definition
Name ac parameter=value ...

Parameters
1 prevoppoint=no Use operating point computed on the previous analysis. Possible values are no or yes.

Sweep interval parameters
2 start=0 Start sweep limit.
3 stop Stop sweep limit.
4 center Center of sweep.
5 span=0 Sweep limit span.
6 step Step size, linear sweep.
7 lin=50 Number of steps, linear sweep.
8 dec Points per decade.
9 log=50 Number of steps, log sweep.
Analysis Statements

10 values=[...]  Array of sweep values.

Sweep variable parameters

11 dev  Device instance whose parameter value is to be swept.
12 mod  Model whose parameter value is to be swept.
13 param Name of parameter to sweep.
14 freq (Hz) Frequency when parameter other than frequency is being swept.

State-file parameters

15 readns  File that contains estimate of DC solution (nodeset).

Output parameters

16 save  Signals to output.
Possible values are all, lvl, allpub, lvlpub, selected, or none.
17 nestlvl  Levels of subcircuits to output.
18 oppoint=no  Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if operating point is computed in the previous analysis and is unchanged.
Possible values are no, screen, logfile, or rawfile.

Convergence parameters

19 restart=yes  Restart the DC solution from scratch if any condition has changed. If not, use the previous solution as initial guess.
Possible values are no or yes.

Annotation parameters

20 annotate=sweep  Degree of annotation.
Possible values are no, title, sweep, status, or steps.
21 stats=no
   Analysis statistics.
   Possible values are no or yes.

22 title
   Analysis title.

You can specify sweep limits by giving the end points or by providing the center value and the
span of the sweep. Steps can be linear or logarithmic, and you can specify the number of
steps or the size of each step. You can give a step size parameter (step, lin, log, dec) to
determine whether the sweep is linear or logarithmic. If you do not give a step size parameter,
the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when
this ratio is 10 or greater. All frequencies are in Hertz.

The small-signal analysis begins by linearizing the circuit about an operating-point. By
default this analysis computes the operating-point if it is not known, or recomputes it if any
significant component or circuit parameter has changed. However, if a previous analysis
computed an operating point, you can set prevoppoint=yes to avoid recomputing it. For
example, if you use this option when the previous analysis was a transient analysis, the
operating point is the state of the circuit on the final time point.

Parameter Index

In the following index, the number following each parameter name indicates where to find the
description of that parameter.

annotate 20         log 9               readns 15         step 6
center 4            mod 12              restart 19        stop 3
dec 8               nestlvl 17          save 16           title 22
dev 11              oppoint 18          span 5            values 10
freq 14             param 13            start 2
lin 7               prevoppoint 1       stats 21
Alter a Circuit, Component, or Netlist Parameter (alter)

Description

The `alter` statement changes the value of any modifiable component or netlist parameter for any analyses that follow. The parameter to be altered can be circuit temperature, a device instance parameter, a device model parameter, a netlist parameter, or a subcircuit parameter for a particular subcircuit instance. You can alter the circuit temperature by giving the parameter name as `param=temp` with no `dev`, `mod` or `sub` parameter. You can alter a top-level netlist parameter by giving the parameter name with no `dev`, `mod` or `sub` parameter. You can alter a subcircuit parameter for a particular subcircuit instance by specifying the subcircuit instance name with the `sub` parameter, and the subcircuit parameter name with the `param` parameter. Each `alter` statement can change only one parameter.

Definition

Name alter parameter=value ...

Parameters

1. `mod` Device model.
2. `dev` Device instance.
3. `sub` Subcircuit instance.
4. `param` Name of parameter to be altered.
5. `value` New value for parameter.
6. `annotate` Degree of annotation. Possible values are `no` or `title`.

Alter Group (altergroup)

Description

The `altergroup` statement changes the values of any modifiable model, instance or netlist parameter for any analyses that follow. Within an alter group, you can specify model statements, instance statements and parameter statements. These statements should be
bound within braces. The opening brace is required at the end of the line defining the alter
group. Alter groups cannot be nested or specified within subcircuits. The following statements
are not allowed within altergroups (analyses, export, ic, nodeset, paramset, save, and
sens).

Within an alter group, each device (instance or model) is first defaulted and then the device
parameters are updated. For netlist parameters, the expressions are updated and evaluated.

For subckt within altergroup, all instances of the subckts are modified during the altergroup.
There are strict checks that do not allow changes to topology.

You can include files into the alter group and can use the simulator lang=spice directive.
See spectre -h include for more details. A model defined in the netlist, has to have the
same model name and primitive type (bsim2, bsim3, bjt) in the alter group. An instance
defined in the netlist, has to have the same instance name, terminal connections and primitive
type. For model groups you can change the number of models in the group. There is a
restriction that you cannot change from a model to a model group and vice versa. See
spectre -h bsim3v3 for details on model groups.

Definition
Name altergroup parameter=value ... 

Parameters

1 annotate Degree of annotation.
Possible values are no or title.

Example:

FastCorner altergroup {

parameters p2=1 p3=p1+2

model myres resistor r1=1e3 af=1

model mybsim bsim3v3 lmax=p1 lmin=3.5e-7

m1 (n1 n2 n3 n4) mybsim w=0.3u l=1.2u

}

The list of public devices supported by altergroup:
Check Parameter Values (check)

Description

The check analysis checks the values of component parameters to assure they are reasonable. This analysis reduces the cost of data entry errors. Various filters specify which parameters are checked. You can perform checks on input, output, or operating-point parameters. Use this analysis in conjunction with the +param command line argument, which specifies a file that contains component parameter soft limits.

Definition

Name check parameter=value ...
Parameters

1 what=all What parameters should be checked. Possible values are none, inst, models, input, output, all, or oppoint.

Checklimit Analysis (checklimit)

Description

A checklimit analysis allows the enabling or disabling of individual or group of asserts specified in the netlist. Use this analysis in conjunction with assert statements in the netlist to perform checks on parameters of device instances, models, subcircuits or expressions.

Multiple checklimit analyses maybe defined in the netlist. The enabled checks will be applied to all subsequent analyses until the next checklimit analysis is encountered.

Definition

Name checklimit parameter=value ...

Parameters

1 enable=[...] Array of checks to be enabled. Default is all.
2 disable=[...] Array of checks to be disabled. Default is none.
3 start=0 s Start(time) of the checks.
4 stop (s) Stop(time) of the checks.
5 severity Severity of the checks. Possible values are none, notice, warning, or error.
6 title Analysis title.
7 checkallasserts=yes If all checks should be enabled or disabled. CheckAllAsserts=no will disable all checks. Possible values are no or yes.
DC Analysis (dc)

Description

The DC analysis finds the DC operating-point or DC transfer curves of the circuit. To generate transfer curves, specify a parameter and a sweep range. The swept parameter can be circuit temperature, a device instance parameter, a device model parameter, a netlist parameter, or a subcircuit parameter for a particular subcircuit instance. You can sweep the circuit temperature by giving the parameter name as `param=temp` with no `dev`, `mod` or `sub` parameter. You can sweep a top-level netlist parameter by giving the parameter name with no `dev`, `mod` or `sub` parameter. You can sweep a subcircuit parameter for a particular subcircuit instance by specifying the subcircuit instance name with the `sub` parameter, and the subcircuit parameter name with the `param` parameter. After the analysis has completed, the modified parameter returns to its original value.

Definition

Name dc parameter=value ...

Parameters

Sweep interval parameters

1. `start=0`  
   Start sweep limit.
2. `stop`  
   Stop sweep limit.
3. `center`  
   Center of sweep.
4. `span=0`  
   Sweep limit span.
5. `step`  
   Step size, linear sweep.
6. `lin=50`  
   Number of steps, linear sweep.
7. `dec`  
   Points per decade.
8. `log=50`  
   Number of steps, log sweep.
9. `values=[... ]`  
   Array of sweep values.


Sweep variable parameters

10 dev 
Device instance whose parameter value is to be swept.

11 mod 
Model whose parameter value is to be swept.

12 param 
Name of parameter to sweep.

State-file parameters

13 force=none 
What should be used to force values for DC. Uses the values from the device and node ICs. Possible values are none, node, dev, or all.

14 readns 
File that contains estimate of DC solution (nodeset).

15 readforce 
File that contains force values.

16 write 
File to which solution at first step in sweep is written.

17 writefinal 
File to which solution at last step in sweep is written.

Output parameters

18 save 
Signals to output. Possible values are all, lvl, allpub, lvlpub, selected, or none.

19 nestlvl 
Levels of subcircuits to output.

20 print=no 
Print node voltages. Possible values are no or yes.

21 oppoint=no 
Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if sweep parameter param is set. Possible values are no, screen, logfile, or rawfile.

22 check=yes 
Check operating point parameters against soft limits. Possible values are no or yes.
Convergence parameters

23 homotopy=all  Method used when no convergence on initial attempt of DC analysis.
Possible values are none, gmin, source, dptran, ptran, arclength, or all.

24 restart=yes  Restart from scratch if any condition has changed. If not, use the previous solution as initial guess.
Possible values are no or yes.

25 maxiters=150  Maximum number of iterations.

26 maxsteps=10000 Maximum number of steps used in homotopy method.

Annotation parameters

27 annotate=sweep  Degree of annotation.
Possible values are no, title, sweep, status, or steps.

28 title  Analysis title.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, dec) and determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. If you specify the oppoint parameter, Spectre computes and outputs the linearized model for each nonlinear component.

Nodesets help find the DC or initial transient solution. You can supply them in the circuit description file with nodeset statements, or in a separate file using the readns parameter. When nodesets are given, Spectre computes an initial guess of the solution by performing a DC analysis while forcing the specified values onto nodes by using a voltage source in series with a resistor whose resistance is rforce. Spectre then removes these voltage sources and resistors and computes the true solution from this initial guess.

Nodesets have two important uses. First, if a circuit has two or more solutions, nodesets can bias the simulator towards computing the desired one. Second, they are a convergence aid. By estimating the solution of the largest possible number of nodes, you might be able to eliminate a convergence problem or dramatically speed convergence.
When you simulate the same circuit many times, we suggest that you use both the write and readns parameters and give the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.

You may specify values to force for the DC analysis by setting the parameter force. The values used to force signals are specified by using the force file, the ic statement, or the ic parameter on the capacitors and inductors. The force parameter controls the interaction of various methods of setting the force values. The effects of individual settings are:

force=none: Any initial condition specifiers are ignored.

force=node: The ic statements are used, and the ic parameter on the capacitors and inductors are ignored.

force=dev: The ic parameters on the capacitors and inductors are used, and the ic statements are ignored.

force=all: Both the ic statements and the ic parameters are used, with the ic parameters overriding the ic statements.

If you specify a force file with the readforce parameter, force values read from the file are used, and any ic statements are ignored.

Once you specify the force conditions, Spectre computes the DC analysis with the specified nodes forced to the given value by using a voltage source in series with a resistor whose resistance is rforce (see options).

**Parameter Index**

In the following index, the number following each parameter name indicates where to find the description of that parameter.
DC Device Matching Analysis (dcmatch)

Description
The DCMATCH analysis performs DC device mis-matching analysis for a given output. It computes the deviation in the DC operating point of the circuit caused by mismatch in the devices. Users need to specify mismatch parameters in their model cards for each device contributing to the deviation. The analysis uses the device mismatch models to construct equivalent mismatch current sources to all the devices that have mismatch modeled. These current sources will have zero mean and some variance. The variance of the current sources are computed according to mismatch models. It then computes the 3-sigma variance of dc voltages or current at user specified outputs due to the mismatch current sources. The simulation results displays the devices rank ordered by their contributions to the outputs. In addition, for mosfet devices, it displays threshold voltage mismatch, current factor mismatch, gate voltage mismatch, and drain current mismatch. For bipolar devices, it displays base-emitter junction voltage mismatch. For resistors, it displays resistor mismatches.

The analysis replaces multiple simulation runs by circuit designers for accuracy vs. size analysis. It automatically identifies the set of critical matched components during circuit design. For example, when there are matched pairs in the circuit, the contribution of two matched transistors will be equal in magnitude and opposite in sign. Typical usage are to simulate the output offset voltage of operational amplifiers, estimate the variation in bandgap voltages, and predict the accuracy of current steering DACS.

Definition
Name ... dcmatch parameter=value ...

Parameters
1 mth Relative mismatch contribution threshold value.
2 where=screen Where DC-Mismatch analysis results should be printed. Possible values are screen, logfile, file, or rawfile.
3 `file` File name for results to be printed if where=file is used.

**Probe parameters**

4 `oprobe` Compute mismatch at the output defined by this component.

**Port parameters**

5 `portv` Voltage across this probe port is output of the analysis.

6 `porti` Current through this probe port is output of the analysis.

**Sweep interval parameters**

7 `start=0` Start sweep limit.

8 `stop` Stop sweep limit.

9 `center` Center of sweep.

10 `span=0` Sweep limit span.

11 `step` Step size, linear sweep.

12 `lin=50` Number of steps, linear sweep.

13 `dec` Points per decade.

14 `log=50` Number of steps, log sweep.

15 `values=[...]` Array of sweep values.

**Sweep variable parameters**

16 `dev` Device instance whose parameter value is to be swept.

17 `mod` Model whose parameter value is to be swept.

18 `param` Name of parameter to sweep.
State-file parameters

19 readns
File that contains estimate of DC solution (nodeset).

Output parameters

20 save
Signals to output.
Possible values are all, lvl, allpub, lvlpub, selected, or none.

21 nestlvl
Levels of subcircuits to output.

22 oppoint=no
Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if (1) operating point is computed in the previous analysis and is unchanged, or (2) sweep parameter param is set.
Possible values are no, screen, logfile, or rawfile.

Convergence parameters

23 prevoppoint=no
Use operating point computed on the previous analysis.
Possible values are no or yes.

24 restart=yes
Restart the DC solution from scratch if any condition has changed. If not, use the previous solution as initial guess.
Possible values are no or yes.

Annotation parameters

25 annotate=sweep
Degree of annotation.
Possible values are no, title, sweep, status, or steps.

26 stats=no
Analysis statistics.
Possible values are no or yes.

27 title
Analysis title.

The dcmatch analysis will find a dc operating point first. If the dc analysis fails, then the dcmatch analysis will fail also. The parameter mth is a threshold value relative to maximum
contribution. Any device contribution less than \((mth \times \text{maximum})\) will not be reported. Where maximum is the maximum contribution among all the devices of a given type.

Examples:

- \textit{dcmm1} \textit{dcmatch mth=1e-3 oprobe=vd porti=1}
- \textit{dcmm2} \textit{dcmatch mth=1e-3 oprobe=r3 portv=1}
- \textit{dcmm3 n1 n2 dcmatch mth=1e-3 where=rawfile stats=yes}
- \textit{dcmm4 n3 0 dcmatch mth=1e-3 where=file file="%C:r.info.what"}
- \textit{sweep1 sweep dev=mp6 param=w start=80e-6 stop=90e-6 step=2e-6}
- \{ \textit{dcmm5} \textit{dcmatch oprobe=vd mth=1e-3 where=rawfile} \}
- \textit{dcmm6 n3 0 dcmatch mth=0.01 dev=x1.mp2 param=w start=15e-6 stop=20e-6 step=1e-6}
- \textit{dcmm7 n3 0 dcmatch mth=0.01 param=temp start=25 stop=100 step=25}

Note:

porti allows users to select a current associated with a specific device given in oprobe as an output. This device, however, has to have its terminal currents as network variables, i.e. the device has to be an inductor, a vsource, a switch, a tline, a controlled voltage source, an iprobe, or other type of device which has current solution. Further, for inductor, vsource, switch, controlled voltage source and iprobe, porti can only be set to one, since these devices are two terminal devices (one port); and for tline porti can be set to one or two, since it is a four terminal device (two ports).

**Parameter Index**

In the following index, the number following each parameter name indicates where to find the description of that parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotate</td>
<td>25</td>
</tr>
<tr>
<td>mod</td>
<td>17</td>
</tr>
<tr>
<td>portv</td>
<td>5</td>
</tr>
<tr>
<td>stats</td>
<td>26</td>
</tr>
<tr>
<td>center</td>
<td>9</td>
</tr>
<tr>
<td>mth</td>
<td>1</td>
</tr>
<tr>
<td>prevoppoint</td>
<td>23</td>
</tr>
<tr>
<td>step</td>
<td>11</td>
</tr>
<tr>
<td>dec</td>
<td>13</td>
</tr>
<tr>
<td>nestlvl</td>
<td>21</td>
</tr>
<tr>
<td>readns</td>
<td>19</td>
</tr>
<tr>
<td>stop</td>
<td>8</td>
</tr>
<tr>
<td>dev</td>
<td>16</td>
</tr>
<tr>
<td>oppoint</td>
<td>22</td>
</tr>
<tr>
<td>restart</td>
<td>24</td>
</tr>
<tr>
<td>title</td>
<td>27</td>
</tr>
</tbody>
</table>
Envelope Following Analysis (envlp)

Description
This analysis computes the envelope response of a circuit. The user specifies the analysis clockname. The simulator automatically determines the clock period by looking through all the sources with the specified name. The envelope response is computed over the interval from start to stop. If the interval is not a multiple of the clock period, it is rounded off to the nearest multiple before the stop time. The initial condition is taken to be the DC steady-state solution if not otherwise given.

Envelope following analysis is most efficient for circuits where the modulation bandwidth is orders of magnitude lower than the clock frequency. This is typically the case, for example, in circuits where the clock is the only fast varying signal and other input signals have a spectrum whose frequency range is orders of magnitude lower than the clock frequency. For another example, the down conversion of two closely placed frequencies can also generate a slow-varying modulation envelope.

The analysis generates two types of output files, a voltage versus time (td) file, and an amplitude/phase versus time (fd) file for each of specified harmonic of the clock fundamental.

Definition
Name envlp parameter=value ...

Parameters

Envelope fundamental parameters

1 clockname  Name of the clock fundamental.

2 modulationbw (Hz) Modulation bandwidth.
Resolution bandwidth, if set, will overwrite the StopTime to be at least \(1/\text{resolutionbw}\).

### Simulation interval parameters

4. `stop (s)`
   - Stop time.

5. `start=0 s`
   - Start time.

6. `tstab=0 s`
   - Initial stabilization time, can be used to change the phase that `envlp` starts shooting.

7. `outputstart=start s`
   - Output is saved only after this time is reached.

### Time-step parameters

8. `maxstep (s)`
   - Maximum time step for inner transient integration. Default is 50 steps per Clock period.

9. `envmaxstep (s)`
   - Maximum outer envelope step size. Default derived from `errpreset`.

### Initial-condition parameters

10. `ic=all`
    - What should be used to set initial condition. Possible values are `dc`, `node`, `dev`, or `all`.

11. `skipdc=no`
    - If yes, there will be no dc analysis for initial transient. Possible values are `no` or `yes`.

12. `readic`
    - File that contains initial transient condition.

### Convergence parameters

13. `readns`
    - File that contains estimate of initial DC solution.

14. `cmin=0 F`
    - Minimum capacitance from each node to ground.

State-file parameters
Spectre Circuit Simulator Reference
Analysis Statements

15 write
File to which initial transient solution is to be written.

16 writefinal
File to which final transient solution is to be written.

17 swapfile
Temporary file that holds the matrix information used by Newtons method. Tells Spectre to use a regular file rather than virtual memory to hold the matrix information. Use this option if Spectre complains about not having enough memory to complete this analysis.

Envelope Integration method parameters

18 envmethod=gear2only
Envelop Integration method.
Possible values are euler, trap, traponly, gear2, gear2only, or trapgear2.

Integration method parameters

19 method=gear2only
Iner transient integration method.
Possible values are euler, trap, traponly, gear2, gear2only, or trapgear2.

Accuracy parameters

20 errpreset=moderate
Selects a reasonable collection of parameter settings.
Possible values are liberal, moderate or conservative.

21 relref
Reference used for the relative convergence criteria. Default derived from errpreset.
Possible values are pointlocal, alllocal, sigglobal, or allglobal.

22 lteratio
Ratio used to compute LTE tolerances from Newton tolerance. Default derived from errpreset.

23 steadyratio
Ratio used to compute steady state tolerances from LTE tolerance. Default derived from errpreset.
24 envltetatio

Ratio used to compute envelope LTE tolerances. Default derived from errpreset.

**Annotation parameters**

25 stats=no

Analysis statistics.
Possible values are no or yes.

26 annotate=sweep

Degree of annotation.
Possible values are no, title, sweep, status, or steps.

27 title

Analysis title.

**Output parameters**

28 harms=1

Number of clock harmonics to output.

29 harmsvec= [...] 

Array of desired clock harmonics. Alternate form of harms that allows selection of specific harmonics.

30 outputtype=both

Output type.
Possible values are both, envelope or spectrum.

31 save

Signals to output.
Possible values are all, lvl, allpub, lvlpub, selected, or none.

32 nestlvl

Levels of subcircuits to output.

33 compression=no

Do data compression on output. See full description bellow.
Possible values are no or yes.

34 strobeperiod (s)

The output strobe interval (in seconds of envelope following time). The actual strobe interval is rounded to an integer multiple of the clock period.

**Newton parameters**

35 maxiters=5

Maximum number of Newton iterations per transient integration time step.
36 envmaxiters=3 Maximum number of Newton iterations per envelope step.

37 restart=yes Restart the DC solution from scratch if any condition has changed. If not, use the previous solution as initial guess. Possible values are no or yes.

**Circuit age**

38 circuitage (Years) Stress Time. Age of the circuit used to simulate hot-electron degradation of MOSFET and BSIM circuits.

The simulator examines all the sources whose name matches the clock name specified in the analysis line by the `clockname` parameter to determine the clock frequency. If more than one frequencies are found, the greatest common factor of these frequencies is used as the clock frequency.

The maximum envelope step size is affected by many parameters. It can be directly limited by `envmaxstep`. It is also limited by `modulationbw`. The user gives an estimate of the modulation bandwidth. The simulator will put at least eight points within the modulation period. The user is recommended to use `strobeperiod` to get equally spaced envelope points, to improve the noise floor in power spectrum density computation.

The `harms` and `harmsvec` parameters affect the simulation time in a insignificant way. The spectrum is calculated for all the specified harmonics for all sampled integration cycles as the envelope following analysis marches on. For each harmonic, a file is generated. Typically, `harms` is set to 1 or 2. High order harmonics will not be accurate.

Most parameters of this analysis are inherited from either transient or PSS analysis and their meanings are consistent. However, a few of them need to be clarified. The effect of `errpreset` on some particular envelope following analysis parameters is shown in the following table.

**Parameter defaults as a function of `errpreset`**

<table>
<thead>
<tr>
<th>errpreset</th>
<th>envmaxstep</th>
<th>reltol</th>
<th>relref</th>
<th>steadyratio</th>
<th>envlteratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>liberal</td>
<td>Interval/10</td>
<td>0.01</td>
<td>sigglobal</td>
<td>0.1</td>
<td>0.35</td>
</tr>
<tr>
<td>moderate</td>
<td>Interval/25</td>
<td>0.001</td>
<td>sigglobal</td>
<td>0.1</td>
<td>3.5</td>
</tr>
<tr>
<td>conservative</td>
<td>Interval/50</td>
<td>0.0001</td>
<td>sigglobal</td>
<td>1.0</td>
<td>35.0</td>
</tr>
</tbody>
</table>

The default value for `compression` is no. The output file stores data for every signal at every timepoint for which Spectre calculates a solution. Spectre saves the x axis data only once,
since every signal has the same x value. If `compression=yes`, Spectre writes data to the output file only if the signal value changes by at least 2*the convergence criteria. In order to save data for each signal independently, x axis information corresponding to each signal must be saved. If the signals stay at constant values for large periods of the simulation time, setting `compression=yes` results in a smaller output data file. If the signals in your circuit move around a lot, setting `compression=yes` results in a larger output data file.

### Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

- annotate 26
- harms 28
- outputtype 30
- steadyratio 23
- circuitage 38
- harmsvec 29
- readic 12
- stop 4
- clockname 1
- ic 10
- readns 13
- strobeperiod 34
- cmin 14
- lteratio 22
- relref 21
- swapfile 17
- compression 33
- maxiters 35
- resolutionbw 3
- title 27
- envlteratio 24
- maxstep 8
- restart 37
- tstab 6
- envmaxiters 36
- method 19
- save 31
- write 15
- envmaxstep 9
- modulationbw 2
- skipdc 11
- writefinal 16
- envmethod 18
- nestlvl 32
- start 5
- errpreset 20
- outputstart 7
- stats 25

### Circuit Information (info)

#### Description

The circuit information analysis outputs several kinds of information about the circuit and its components. You can use various filters to specify what information is output. You can create a listing of model, instance, temperature-dependent, input, output, and operating point
parameters. You can also generate a summary of the minimum and maximum parameter values (by using `extremes=yes` or `only`). Finally, you can request that Spectre provide a node-to-terminal map (by using `what=terminals`) or a terminal-to-node map (by using `what=nodes`).

The following are brief descriptions of the types of parameters you can request with the `info` statement:

Input parameters: Parameters that you specify in the netlist, such as the given length of a MOSFET or the saturation current of a bipolar transistor (use `what=inst, models, input, or all`)

Output parameters: Parameters that are computed by Spectre, such as temperature dependent parameters and the effective length of a MOSFET after scaling (use `what=output or all`)

Operating-point parameters: Parameters that depend on the actual solution computed (use `what=oppoint`)

**Definition**

Name info parameter=value ...

**Parameters**

1. `what=oppoint` What parameters should be printed. Possible values are `none, inst, models, input, output, nodes, all, terminals, oppoint, captab, parameters, primitives, subckts, assert, or allparameters`

2. `where=logfile` Where parameters should be printed. Possible values are `nowhere, screen, file, logfile, or rawfile`

3. `file="%C:r.info.what"` File name when `where=file`

4. `save` Signals to output. Possible values are `all, lvl, allpub, lvpub, selected, or none`

5. `nestlvl` Levels of subcircuits to output
6 extremes=yes
   Print minimum and maximum values.
   Possible values are no, yes or only.

7 title
   Analysis title.

Captab parameters

8 detail=node
   How detailed should the capacitance table be.
   Possible values are node, nodetoground or nodetonode.

9 sort=name
   How to sort the capacitance table.
   Possible values are name or value.

10 threshold=0 F
   Threshold value for printing capacitances (ignore capacitances
   smaller than this value).

Parameter Index

In the following index, the number following each parameter name indicates where to find the
description of that parameter.

detail  8     nestlvl  5     threshold  10     where  2
extremes  6     save  4     title  7
file  3     sort  9     what  1

Monte Carlo Analysis (montecarlo)

Description

The montecarlo analysis is a swept analysis with associated child analyses similar to the
sweep analysis (see spectre -h sweep.) The Monte Carlo analysis refers to "statistics
blocks" where statistical distributions and correlations of netlist parameters are specified.
(Detailed information on statistics blocks is given below.) For each iteration of the Monte Carlo
analysis, new pseudo-random values are generated for the specified netlist parameters
(according to their specified distributions) and the list of child analyses are then executed.
Expressions are associated with the child analyses. These expressions, which are constructed as scalar calculator expressions by the user during Monte Carlo analysis set up, can be used to measure circuit metrics, such as the slew-rate of an op-amp. During a Monte Carlo analysis, these expression results will vary as the netlist parameters vary for each Monte Carlo iteration. The Monte Carlo analysis therefore becomes a tool that allows you to examine and predict circuit performance variations, which affect yield.

The statistics blocks allow you to specify batch-to-batch (process) and per-instance (mismatch) variations for netlist parameters. These statistically-varying netlist parameters can be referenced by models or instances in the main netlist and may represent IC manufacturing process variation, or component variations for board-level designs for example. The following description gives a simplified example of the Monte Carlo analysis flow:

perform nominal run if requested

if any errors in nominal run then stop

foreach Monte Carlo iteration {
    if process variations specified then
        apply process variation to parameters
    if mismatch variations specified then
        foreach subcircuit instance {
            apply mismatch variation to parameters
        }
    foreach child analysis {
        run child analysis
        evaluate expressions
    }
}

**Definition**

Name montecarlo parameter=value ...
Parameters

**Analysis parameters**

1. `numruns=100`  
   Number of Monte Carlo iterations to perform (not including nominal).

2. `seed`  
   Optional starting seed for random number generator.

3. `scalarfile`  
   Output file that will contain output scalar data.

4. `paramfile`  
   Output file that will contain output scalar data labels.

**Saving Process Parameters**

5. `saveprocessparams`  
   Whether or not to save scalar data for statistically varying process parameters which are subject to process variation. Possible values are `no` or `yes`.

6. `processscalarfile`  
   Output file that will contain process parameter scalar data.

7. `processparamfile`  
   Output file that will contain process parameter scalar data labels.

8. `saveprocessvec=[... ]`  
   Array of statistically varying process parameters (which are subject to process variation) to save as scalar data in `processscalarfile`.

9. `firstrun=1`  
   Starting iteration number.

10. `variations=process`  
    Level of statistical variation to apply. Possible values are `process`, `mismatch` or `all`.

**Flags**

11. `donominal=yes`  
    Whether or not to perform nominal run. Possible values are `no` or `yes`. 
12 `appendsd=no` Whether or not to append scalar data. Possible values are `no` or `yes`.

13 `savefamilyplots=no` Whether or not to save data for family plots. If yes, this could require a lot of disk space. Possible values are `no` or `yes`.

**Annotation parameters**

14 `annotate=sweep` Degree of annotation. Possible values are `no`, `title`, `sweep`, or `status`.

15 `title` Analysis title.

**Detailed Description and Examples**

- **numruns:**(default=100)
  
  The number of Monte Carlo iterations to perform. The simulator will perform a loop, running the specified child analyses and evaluating any expressions `numruns` times.

- **seed:**(no default)
  
  Seed for the random number generator. By always specifying the same seed, you can reproduce a previous experiment. If you do not specify a seed, then each time you run the analysis, you will get different results i.e. a different stream of pseudo-random numbers will be generated.

- **scalarfile="filename"**
  
  This parameter allows an ASCII file to be specified in which scalar data (results of expressions that resolve to scalar values) will be written. The data from this file can be read and plotted in histograms by Artist. For each iteration of each Monte Carlo child analyses, Spectre (through Artil) will write a line to this ASCII file which contains scalar data (one scalar expression per column e.g. slewrate or bandwidth.) The default name for this file will be of the form name.mcddata, where name is the name of the Monte Carlo analysis instance. This file contains only the matrix of numeric values. Artist Monte Carlo users will be more familiar with the term `mcddata` file for the scalar file. Additionally, when the Analog Artist Monte Carlo tool is used to generate the spectre netlist file, Spectre will merge the values of the statistically varying process parameters into this file containing the scalar data (results of expressions). This means that Analog Artist can later read the data, and create scatterplots of the statistically varying process parameters against each other, or against the results of the
expressions. In this way, the user can see correlations between process parameter variations and circuit performances variations. This data merging will take place whenever the scalarfile and processscalarfile (see below) are written in the same directory.

**paramfile="filename"**

This file contains the titles, sweep variable values and the full expression for each of the columns in the scalarfile. Artist Monte Carlo users will be more familiar with the term `mcparam` file for the paramfile. This file will be created in the psf directory by default, unless you specify some path information in the filename.

**processscalarfile="filename"**

If `saveprocessparams` is set to yes, then the process (batch-to-batch) values of all statistically varying parameters are saved to this scalar data file. You can use the `saveprocessvec` to filter out a subset of parameters in which case Spectre will save only the parameters specified in `saveprocessvec` to the processscalarfile. The processscalarfile is equivalent to the scalarfile, except that the data in the scalarfile contains the values of the scalar expressions, whereas the data in the processscalarfile contains the corresponding process parameter values. The default name for this file will be of the form `instname.process.mcdata`, where `instname` is the name of the Monte Carlo analysis instance. This file will be created in the psf directory by default, unless you specify some path information in the filename. You can load the processscalar file and processparamfile into the Artist statistical postprocessing environment to plot/verify the process parameter distributions. If you later merge the processparamfile with the data in the scalarfile, you can then plot scalar expressions values against the corresponding process parameters by loading this merged file into the Artist statistical postprocessing environment.

**processparamfile="filename"**

This file contains the titles, sweep variable values for each of the columns in the processscalarfile. These titles will be the names of the process parameters.

The processparamfile is equivalent to the paramfile, except that the paramfile contains the name of the expressions, whereas the processparamfile contains the names of the process parameters. The default name for this file will be of the form `instname.process.mcpparam`, where `instname` is the name of the Monte Carlo analysis instance. This file will be created in the psf directory by default, unless you specify some path information in the filename.

**firstrun:(default=1)**

index of first iteration. If the first iteration is specified as some number n greater than one, then the beginning n-1 iterations are skipped, i.e. the Monte Carlo analysis behaves as if the first n-1 iterations were run, but without actually performing the child analyses for these iterations. The subsequent stream of random numbers generated for the remaining iterations will be the
same as if the first n-1 iterations were actually run. By specifying the first iteration number and the same value for seed, you can reproduce a particular run or sequence of runs from a previous experiment (for example to examine an outlier case in more detail.)

variations={process,mismatch,all} (defaults to process).

Whether to apply process (batch-to-batch) variations only, or mismatch (per-instance) variations only, or both together. This assumes that you have specified appropriate statistical distributions in the statistics block. You cannot request that mismatch variations be applied unless you have specified mismatch statistics in the statistics block. You cannot request that process variations be applied unless you have specified process statistics in the statistics block. More details on statistics blocks are given below.

saveprocessvec=[rshsp TOX ...]

If saveprocessparams is specified as yes, then save the process (batch-to-batch) values of only those parameters listed in saveprocessvec in the processparamfile. This acts as a filter so that you do not save all process parameters to the file. If you do not want to filter the list of process parameters, then do not specify this parameter.

donominal={yes,no}(defaults to yes).

This parameter controls whether or not Spectre should perform a nominal run before starting the main Monte Carlo loop of iterations. If any errors are encountered during the nominal run (e.g. convergence problems, incorrect expressions, etc.) then Spectre will issue an appropriate error message and immediately abandon the Monte Carlo analysis.

If donominal is set to no then Spectre will run the Monte Carlo iterations only, and will not perform a nominal analysis. If any errors are encountered during the Monte Carlo iterations, Spectre will issue a warning and continue with the next iteration of the Monte Carlo loop.

appendsd={yes,no}(defaults to no).

Specifies whether to append scalar data to an existing scalarfile, or to overwrite the existing scalarfile. This flag applies to both the scalar file and the processscalarfile.

savefamilyplots={yes,no}.

If yes, a data file (e.g. psf) is saved for each analysis for each Monte Carlo iteration, in addition to the expressions scalar results which are saved to the ASCII scalar data file at the end of each iteration. Saving the full data files between runs enables the cloud plotting feature (overlaid waveforms) in Artist. It also enables the user to define/evaluate new calculator measurements after the simulation has been run using the Artist calculator. This feature could result in a huge amount of data being stored to disk, and it is advised that you use this feature with care. If you do decide to use this feature, it is advisable to keep the number of saved
quantities to a minimum. If this parameter is set to no, then data files are overwritten by each Monte Carlo iteration.

annotate={no,title,sweep,status}

Degree of annotation. Use the maximum value of status to print a summary of which runs did not converge or had problems evaluating expressions, etc.

Examples:

// do a Monte Carlo analysis, with process variations only
// useful for looking at absolute performance spreads
mc1 montecarlo variations=process seed=1234 numruns=200 {
    dcop1 dc  // a child analysis
    tran1 tran start=0 stop=1u  // another child analysis
    // expression calculations are sent to the scalardata file
    export slewrate=oceanEval("slewRate(v("vout"),10n,t,30n,t,10,90 )")
}

// do a Monte Carlo analysis, with mismatch variations only
// useful for detecting spreads in differential circuit
// applications, etc. Do not perform a nominal run.
mc2 montecarlo donominal=no variations=mismatch seed=1234 numruns=200 {
    dcop2 dc
    tran2 tran start=0 stop=1u
    export slewrate=oceanEval("slewRate(v("vout"),10n,t,30n,t,10,90 )")
}

// do both together...
mc3 montecarlo saveprocessparams=yes variations=all numruns=200 {
    dcop3 dc
tran3 tran start=0 stop=1u

export slewrate=oceaneval("slewRate(v("vout"),10n,t,30n,t,10,90 )")

Specifying Parameter Distributions using Statistics Blocks:

The statistics blocks are used to specify the input statistical variations for a Monte Carlo analysis. A statistics block may contain one or more process blocks (which represents batch-to-batch type variations), and/or one or more mismatch blocks (which represents on-chip or device mismatch variations), in which the distributions for parameters are specified. Statistics blocks may also contain one or more correlation statements to specify the correlations between specified process parameters, and/or to specify correlated device instances (for example matched pairs). Statistics blocks may also contain a truncate statement which may be used for generating truncated distributions. The distributions specified in the process block will be sampled once per Monte Carlo iteration, and are typically used to represent batch-to-batch, or process variations, whereas the distributions specified in the mismatch block are sampled on a per subcircuit instance basis and are typically used to represent device-to-device mismatch for devices on the same chip. In the case where the same parameter is subject to both process and mismatch variations, then the sampled process value becomes the mean for the mismatch random number generator for that particular parameter.

NOTE: Multiple statistics blocks may exist, in which case they accumulate or overlay. Typically, process variations, mismatch variations and correlations between process parameters will be specified in one statistics block. A second statistics block would be specified where actual device instance correlations are specified (i.e. specification of matched pairs).

Statistics blocks can be specified using combinations of the Spectre keywords statistics, process, mismatch, vary, truncate and correlate. Braces {} are used to delimit blocks.

The following example shows some sample statistics blocks, which are discussed below along with syntax requirements.

// define some netlist parameters to represent process parameters

// such as sheet resistance and mismatch factors

parameters rshsp=200 rshpi=5k rshpi_std=0.4K xisn=1 xisp=1 xxx=20000 uuu=200

// define statistical variations, to be used
// with a MonteCarlo analysis.

statistics {

    process {  // process: generate random number once per MC run
        vary rshsp dist=gauss std=12 percent=yes
        vary rshpi dist=gauss std=rshpi_std // rshpi_std is a parameter
        vary xxx dist=lnorm std=12
        vary uuu dist=unif N=10 percent=yes

        ...
    }

    mismatch {  // mismatch: generate a random number per instance
        vary rshsp dist=gauss std=2
        vary xisn dist=gauss std=0.5
        vary xisp dist=gauss std=0.5
    }

    // some process parameters are correlated
    correlate param=[rshsp rshpi] cc=0.6

    // specify a global distribution truncation factor
    truncate tr=6.0  // +/- 6 sigma

}

// a separate statistics block to specify correlated (i.e. matched) components
// where m1 and m2 are subckt instances.

statistics {

    correlate dev=[m1 m2] param=[xisn xisp] cc=0.8

}

// some process parameters are correlated

 correlate param=[rshsp rshpi] cc=0.6

 // specify a global distribution truncation factor

 truncate tr=6.0  // +/- 6 sigma
Specifying Distributions

Parameter variations are specified using the following syntax:

\[
\text{vary PAR\_NAME dist}=<\text{type}> \{\text{std}=<\text{value}> | \text{N}=<\text{value}>\} \{\text{percent}=\text{yes|no}\}
\]

Three types of parameter distributions are available: gaussian, lognormal and uniform, corresponding to the <\text{type}> keywords gauss, lnorm and unif respectively. For both the gauss and the lnorm distributions, you specify a standard deviation using the std keyword.

Gaussian Distribution:

For the gaussian distribution, the mean value is taken as the current value of the parameter being varied, giving a distribution denoted by Normal (mean,std). Using the example above, parameter rshpi is varied with a distribution of Normal (5k,0.4k)

Lognormal Distribution:

The lognormal distribution is denoted by

\[
\log(x) = \text{Normal}(\log(\text{mean}), \text{std})
\]

where \(x\) is the parameter being specified as having a lognormal distribution.

(Note: \(\log()\) is the natural logarithm function.) For parameter xxx in the example above, the process variation is according to

\[
\log(xxx) = \text{Normal}(\log(20000), 12)
\]

Uniform Distribution:

The uniform distribution for parameter \(x\) is generated according to

\[
x = \text{unif}(\text{mean}-\text{N}, \text{mean}+\text{N})
\]

such that the mean value is the nominal value of the parameter \(x\), and the parameter is varied about the mean with a range of +/- \(N\). The standard deviation is not specified for the uniform distribution, but its value can be calculated from the formula: \(\text{std}=\text{N}/\sqrt{3}\).

Values as percentages:

The \text{percent} flag indicates whether the standard deviation \text{std} or uniform range \text{N} are specified in absolute terms (\text{percent}=\text{no}) or as a percentage of the mean value (\text{percent}=\text{yes}). For parameter uuu in the example above, the mean value is 200, and the variation is 200 +/- 10%*(200) i.e. 200 +/- 20. For parameter rshsp, the process variation is
given by Normal (200, 12%*(200)) i.e. Normal (200, 24). It is not advised that you use percent=yes with the lognormal distribution.

Process and Mismatch Variations:

The statistics specified in a process block are applied at global scope, and the distributions are sampled once per Monte Carlo iteration. The statistics specified in a mismatch block are applied on a per-subcircuit instance basis, and are sampled once per subcircuit instance. If you place model cards and/or device instances in subcircuits, and add a mismatch block to your statistics block you can effectively model device-to-device mismatch for these devices/models.

Correlation Statements:

There are two types of correlation statements that you can use: process parameter correlation statements, and instance correlation statements.

Process Parameter Correlation:

The syntax of the process parameter correlation statement is:

\[
\text{correlate param=}[\text{list of parameters}] \ cc=<\text{value}>
\]

This allows you to specify a correlation coefficient between multiple process parameters. You can specify multiple process parameter correlation statements in a statistics block, to build a matrix of process parameter correlations. During a Monte Carlo analysis, process parameter values will be randomly generated according to the specified distributions and correlations.

Mismatch Correlation (Matched Devices):

The syntax of the instance or mismatch correlation statement is:

\[
\text{correlate dev=}[\text{list of subcircuit instances}] \ {\text{param=}[\text{list of parameters}]} \ cc=<\text{value}>
\]

where the device or subcircuit instances to be matched are listed in the list of subcircuit instances, and the list of parameters specifies exactly which parameters with mismatch variations are to be correlated.

The instance mismatch correlation statement is used to specify correlations for particular subcircuit instances. If a subcircuit contains a device, you can effectively use the instance correlation statements to specify that certain devices are correlated (i.e. matched) and give the correlation coefficient. You can optionally specify exactly which parameters are to be correlated by giving a list of parameters (each of which must have had distributions specified for it in a mismatch block), or specify no parameter list, in which case all parameters with mismatch statistics specified are correlated with the given correlation coefficient. The
correlation coefficients are specified in the <value> field and must be between +/- 1.0, not including 1.0 or -1.0.

NOTE: correlation coefficients can be constants or expressions, as can std and N when specifying distributions.

Truncation Factor:

The default truncation factor for gaussian distributions (and for the gaussian distribution underlying the lognormal distribution) is 4.0 sigma. Randomly generated values which are outside the range of mean +/- 4.0 sigma are automatically rejected and regenerated until they fall inside the range. You can change the truncation factor using the truncate statement. The syntax is:

truncate tr=<value>.

Note: The value of the truncation factor can be a constant or an expression.

Note: Parameter correlations can be affected by using small truncation factors.

Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate 14 numruns 1 savefamilyplots 13
appendsd 12 paramfile 4 saveprocessparams 5
donominal 11 processparamfile 7 saveprocessvec 8 variations 10
firstrun 9 processscalarfile 6 scalarfile 3
Noise Analysis (noise)

Description

The noise analysis linearizes the circuit about the operating point and computes the noise spectral density at the output. If you identify an input source, the transfer function and the input-referred noise for an equivalent noise-free network is computed. In addition, if the input source is noisy, then the noise figure is computed.

The noise is computed at the output of the circuit. The output is specified with either a pair of nodes or a probe component. To specify the output of a circuit with a probe, specify it with the oprobe parameter. If the output is voltage (or potential), choose a resistor or a port as the output probe. If the output is current (or flow), choose a vsource or iprobe as the output probe.

If the input-referred noise is desired, specify the input source using the iprobe parameter. Currently, only a vsource, an isource, or a port may be used as an input probe. If the input source is noisy, as is a port, the noise analysis will compute the noise factor (F) and noise figure (NF). To match the IEEE definition of noise figure, the input probe must be a port with no excess noise and its noisetemp must be set to 16.85°C (290K). In addition, the output load must be a resistor or port and must be identified as the oprobe.

If port is specified as the input probe, then both input-referred noise and gain are referred back to the equivalent voltage source inside the port. S-parameter analysis calculates those values in traditional sense.

The noise analysis always computes the total noise at the output, which includes contributions from the input source and the output load. The amount of the output noise that is attributable to each noise source in the circuit is also computed and output individually. If the input source is identified, the input-referred noise is computed, which includes the noise from the input source itself. Finally, if the input source is identified and is noisy, the noise factor and noise figure are computed. Thus if

\[
\begin{align*}
No &= \text{total output noise} \\
Ns &= \text{noise at the output due to the input probe (the source)} \\
NI &= \text{noise at the output due to the output probe (the load)} \\
IRN &= \text{input referred noise} \\
G &= \text{gain of the circuit} \\
F &= \text{noise factor}
\end{align*}
\]
NF = noise figure

then,

IRN = sqrt(No^2 / G^2)

F = (No^2 - NI^2)/Ns^2

NF = 10*log10(F)

When the results are output, No is named out, IRN is named in, G is named gain, F is named F, and NF is named NF.

Spectre can perform the analysis while sweeping a parameter. The parameter can be frequency, temperature, component instance parameter, component model parameter, or netlist parameter. If changing a parameter affects the DC operating point, the operating point is recomputed on each step. You can sweep the circuit temperature by giving the parameter name as temp with no dev or mod parameter. You can sweep a netlist parameter by giving the parameter name with no dev, or mod parameter. After the analysis has completed, the modified parameter returns to its original value.

Definition

Name [p] [n] noise parameter=value ...

The optional terminals (p and n) specify the output of the circuit. If you do not give the terminals, then you must specify the output with a probe component.

Parameters

1 prevoppoint=no Use operating point computed on the previous analysis. Possible values are no or yes.

Sweep interval parameters

2 start=0 Start sweep limit.
3 stop Stop sweep limit.
4 center Center of sweep.
5 span=0 Sweep limit span.
Spectre Circuit Simulator Reference
Analysis Statements

6  step  Step size, linear sweep.
7  lin=50  Number of steps, linear sweep.
8  dec  Points per decade.
9  log=50  Number of steps, log sweep.
10  values= [...]  Array of sweep values.

**Sweep variable parameters**
11  dev  Device instance whose parameter value is to be swept.
12  mod  Model whose parameter value is to be swept.
13  param  Name of parameter to sweep.
14  freq (Hz)  Frequency when parameter other than frequency is being swept.

**Probe parameters**
15  oprobe  Compute total noise at the output defined by this component.
16  iprobe  Input probe. Refer the output noise to this component.

**State-file parameters**
17  readns  File that contains estimate of DC solution (nodeset).

**Output parameters**
18  save  Signals to output.
Possible values are all, lvl, allpub, lvlpub, selected, or none.
19  nestlvl  Levels of subcircuits to output.
20  oppoint=no  Should operating point information be computed, and if so, where should it be sent. Operating point information would not
be output if operating point is computed in the previous analysis and is unchanged.
Possible values are no, screen, logfile, or rawfile.

Convergence parameters

21 restart=yes Restart the DC solution from scratch if any condition has changed. If not, use the previous solution as initial guess. Possible values are no or yes.

Annotation parameters

22 annotate=sweep Degree of annotation.
Possible values are no, title, sweep, status, or steps.

23 stats=no Analysis statistics.
Possible values are no or yes.

24 title Analysis title.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, dec) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. All frequencies are in Hertz.

The small-signal analysis begins by linearizing the circuit about an operating-point. By default this analysis computes the operating-point if it is not known, or recomputes it if any significant component or circuit parameter has changed. However, if a previous analysis computed an operating point, you can set prevoppoint=yes to avoid recomputing it. For example, if you use this option when the previous analysis was a transient analysis, the operating point is the state of the circuit on the final time point.

Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate 22    lin 7    param 13    start 2
Immediate Set Options (options)

Description

The immediate set options statement sets or changes various program control options. These options take effect immediately and are set while the circuit is read. For further options, see the individual analyses.

Note: Options that are dependent on netlist parameter values, do not maintain their dependencies on those netlist parameters.

Definition

Name options parameter=value ...

Parameters

Tolerance parameters

1 reltol=0.001 Relative convergence criterion.
2 residualtol=1.0 Tolerance ratio for residual (multiplies reltol).
3 vabstol=1e-06 V Voltage absolute tolerance convergence criterion.
4 iabstol=1e-12 A Current absolute tolerance convergence criterion.
Temperature parameters

5  temp=27 C  Temperature.

6  tnom=27 C  Default component parameter measurement temperature.

7  tempeffects=all  Temperature effect selector. If tempeffect = vt, only thermal voltage varies with temperature; if tempeffect = tc, parameters that start with tc are active and thermal voltage is dependent on temperature; and if tempeffect = all, all built-in temperature models are enabled. Possible values are vt, tc or all.

Output parameters

8  save=selected  Signals to output. Possible values are all, lvl, allpub, lvlpub, selected, or none.

9  nestlvl=∞  Levels of subcircuits to output.

10  subcktprobelvl=0  Level up to which subcircuit terminal current probes are to be set up.

11  currents=selected  Terminal currents to output. (See important note below about saving currents by using probes). Possible values are all, nonlinear or selected.

12  useprobes=no  Use current probes when measuring terminal currents. (See important note below about saving currents by using probes). Possible values are no or yes.

13  useterms=index  Output terminal currents by specified option. Possible values are name or index.

14  redundant_currents=no  If yes, save both currents through two terminal devices. Possible values are no or yes.

15  pwr=none  Power signals to create. Possible values are all, subckts, devices, total, or none.
Spectre Circuit Simulator Reference
Analysis Statements

16  saveahdlvars=selected
    AHDL variables to output.
    Possible values are all or selected.

17  rawfmt=psfbin
    Output raw data file format.
    Possible values are nutbin, nutascii, wsfbin, wsfascii, psfbin, psfascii, psfbinf, awb, or sst2.

18  rawfile="%C\:r.\raw"
    Output raw data file name.

Convergence parameters

19  homotopy=all
    Method used when no convergence on initial attempt of DC analysis.
    Possible values are none, gmin, source, dptran, ptran, arclength, or all.

20  limit=dev
    Limiting algorithms to aid DC convergence.
    Possible values are delta, log or dev.

Multithreading parameters

21  multithread=off
    This option turns on/off multithread capability. When multithreading is turned on but the number of threads (nThreads) is not specified, Spectre will automatically detect the number of processors and select the proper number of threads to use. (See important note below about using multithreading).
    Possible values are off or on.

22  nthreads
    Specifies the number of threads for multithreading.

Component parameters

23  scalem=1
    Model scaling factor.

24  scale=1
    Device instance scaling factor.

25  compatible=spectre
    Encourage device equations to be compatible with a foreign simulator. This option does not affect input syntax.
Possible values are spectre, spice2, spice3, cdsspice, hspice, or spiceplus.

26 approx=no Use approximate models. Difference between approximate and exact models is generally very small. Possible values are no or yes.

27 macromodels=no Circuit contains macromodels; sometimes helps performance. Possible values are no or yes.

28 mos_method=s Global Mos Table Model selection flag. Possible values are s or a.

29 maxrsd=0.0 Ω Use approximation for drain/source parasitic resistors which are less then maxrsd. Applies to bsim3v3, bsim4 mosfet models.

30 mos_vres=0.05 V Voltage increment for mosfet table model interpolation grid. Smaller value will reduce interpolation error, but may increase memory consumption. 20mV is good for those analog circuits, which are extremely sensitive to small model parameter variations, subthreshold and substrate currents.

Error-checking parameters

31 topcheck=full Check circuit topology for errors. Possible values are no, min, full, or fixall.

32 ignshorts=no Silently ignore shorted components. Possible values are no or yes.

33 diagnose=no Print additional information that might help diagnose accuracy and convergence problems. Possible values are no or yes.

34 checklimitfile File to which assert violations will be written to.

35 dochecklimit=yes Check asserts in the netlist. Possible values are no or yes.

36 checklimitdest=file Destination(s) where violations will be written to. Possible values are file, psf or both.
opptcheck=yes
Check operating point parameters against soft limits.
Possible values are no or yes.

Resistance parameters

\texttt{gmin=1e-12 \, S}
Minimum conductance across each nonlinear device.

\texttt{gmin\_check=max\_v\_only}
Specifies that effect of gmin should be reported if significant.
Possible values are no, max\_v\_only, max\_only, or all.

\texttt{rforce=1 \, \Omega}
Resistance used when forcing nodesets and node-based initial conditions.

Quantity parameters

\texttt{value="V"}
Default value quantity.

\texttt{flow="I"}
Default flow quantity.

\texttt{quantities=no}
Print quantities.
Possible values are no, min or full.

Annotation parameters

\texttt{audit=detailed}
Print time required by various parts of the simulator.
Possible values are no, brief, detailed, or full.

\texttt{inventory=detailed}
Print summary of components used.
Possible values are no, brief or detailed.

\texttt{narrate=yes}
Narrate the simulation.
Possible values are no or yes.

\texttt{debug=no}
Give debugging messages.
Possible values are no or yes.

\texttt{info=yes}
Give informational messages.
Possible values are no or yes.
49 note=yes Give notice messages. Possible values are no or yes.

50 maxnotes=5 Maximum number of times any notice will be issued per analysis.

51 warn=yes Give warning messages. Possible values are no or yes.

52 maxwarns=5 Maximum number of times any warning message will be issued per analysis.

53 maxwarnstologfile=5 Maximum number of times any warning message will be printed to the log file per analysis.

54 maxnotestologfile=5 Maximum number of times any notice message will be printed to the log file per analysis.

55 error=yes Give error messages. Possible values are no or yes.

56 digits=5 Number of digits used when printing numbers.

57 notation=eng When printing real numbers to the screen, what notation should be used. Possible values are eng, sci or float.

58 cols=80 Width of screen in characters.

59 title Circuit title.

**Matrix parameters**

60 pivotdc=no Use numeric pivoting on every iteration of DC analysis. Possible values are no or yes.

61 pivrel=0.001 Relative pivot threshold.

62 pivabs=0 Absolute pivot threshold.
**Miscellaneous parameters**

63 `ckptclock=1800 s` Clock time checkpoint period.

**Sensitivity parameters**

64 `sensfile` Output sensitivity data file name.

65 `sensformat=tabular` Format of sensitivity data.
Possible values are `tabular` or `list`.

66 `senstype=partial` Type of sensitivity being calculated.
Possible values are `partial` or `normalized`.

67 `paramrangefile` Parameter range file.

**Performance parameters**

68 `speed` Speeddial setting for transient simulation speed/accuracy control. Possible values are integers 1 to 6.

**Important note about `currents` and `useprobes` options:**

Adding probes to circuits that are sensitive to numerical noise might affect the solution. In such cases accurate solution may be obtained by reducing `reltol`.

The following devices will always use probes to save currents (even with `useprobes=no`): port, delay, switch, hbt, transformer, core, winding, fourier, d2a, a2d, a2ao, a2ai.

**senstype parameter**

When `senstype` is set to `partial`, the sensitivity being calculated is the partial derivative of a differentiable output variable F with respect to a design parameter p:

\[
\frac{dF}{dp} = D(F \text{ w.r.t. } p) = --
\]

\[
\frac{dF}{dp}
\]
This definition is not scale free. When \texttt{senstype} is set to \texttt{normalized}, the sensitivity being calculated is the normalized sensitivity

\[
S (F \text{ w.r.t. } p) = \frac{d \ln F}{d \ln p} \cdot \frac{dF}{dp} \cdot p
\]

When either \( F \) or \( p \) takes a zero value, the above normalized definition no longer provides a useful measure, the following two seminormalized sensitivities are used instead:

\[
S (F \text{ w.r.t. } p) = \frac{dF}{dp} \quad \text{if } F = 0
\]

\[
S (F \text{ w.r.t. } p) = \frac{d \ln F}{dp} \cdot \frac{1}{F} \quad \text{if } p = 0
\]

When both \( F \) and \( p \) are zero, the partial sensitivity is used.

\textbf{topcheck parameter}

When \texttt{topcheck=full}, the topology check is performed and \texttt{gmin} is inserted between isolated nodes and ground. A heuristic topology check is also performed to find nodes that may be isolated due to the numerical nature of the circuit. For example, nodes isolated by reverse biased diodes in MOSFETS.

Use \texttt{topcheck=fixall} to attach \texttt{gmin} to all types of isolated nodes. Including the ones found by the heuristic topology check.

\textbf{Important note about using multithreading}

Currently, multithreading is only available for devices evaluation for BSIM3v3, and BSIM4. Multithreading does not work with table model. If there is an instance of a primitive using table model, multithreading would not be applied to all instances of that primitive.
Multithreading can be turned on/off by command line option, or by the multithread parameter in the options statement from the input file. If both options are specified, command line option will make the final determination on the number of threads to use.

Using multithreading on circuits that are sensitive to numerical noise might affect the solution. The solution should still be within acceptable tolerance specified by the tolerance parameters in the Spectre input file. Due to the order of evaluation of devices is different for each multithreading run of the same simulation, this could lead to different round off error in the computation. It is possible that same exact result may not be reproducible when multithreading is used.

Multithreading would work best when the following capabilities are not used: useprobles=yes, save-current/ SOA/ alarm for multithreaded devices.

**Note about using speed option**

Speed option can be set to a number from 1 to 6, as an easy way to setup multiple accuracy-performance trade-off parameters at once. Speed option sets reltol, vabstol, iabstol, errpreset, and maxrsd parameters implicitly according to the speed number. Users can explicitly specify these parameters to different values, however, to override the setting.

Speed=1 is the most accurate, whereas speed=6 is the fastest:

```
speed       1     2     3     4     5     6
errpreset   con   mod   mod   mod   mod   mod
maxrsd      0     0     10    50    50    50
reltol      1e-3  1e-3  2e-3  5e-3  1e-2  2e-2
vabstol     1e-6  1e-6  1e-6  1e-6  1e-6  1e-6
iabstol     1e-12 1e-12 1e-12 1e-12 1e-12 1e-12
```

Notice that the maxrsd setting in speed table depends on whether there are small-signal analyses or RF analyses exist in the netlist. If there is any, then maxrsd is set to 0, unless maxrsd is explicitly specified. This is to ensure accurate results be obtained for those maxrsd-sensitive analyses.
Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

approx 26  iabstol 4  note 49  scale 24
audit 44  ignshorts 32  nthreads 22  scalem 23
checklimitdest 36  info 48  opptcheck 37  sensfile 64
checklimitfile 34  inventory 45  paramrangefile 67  sensformat 65
ckptclock 63  limit 20  pivabs 62  senstype 66
cols 58  macromodels 27  pivotdc 60  speed 68
compatible 25  maxnotes 50  pivrel 61  subcktprobelvl 10
currents 11  maxnotestologfile 54  pwr 15  temp 5
debug 47  maxrsd 29  quantities 43  tempeffects 7
diagnose 33  maxwarns 52  rawfile 18  title 59
digits 56  maxwarnstologfile 53  rawfmt 17  tnom 6
dochecklimit 35  mos_method 28  redundant_currents 14  topcheck 31
derror 55  mos_vres 30  reltol 1  useprobes 12
flow 42  multithread 21  residualtol 2  useterms 13
gmin 38  narrate 46  rforce 40  vabstol 3
gmin_check 39  nestlvl 9  save 8  value 41
homotopy 19  notation 57  saveahdlvars 16  warn 51
Periodic AC Analysis (pac)

Description

The periodic AC (PAC) analysis is used to compute transfer functions for circuits that exhibit frequency translation. Such circuits include mixers, switched-capacitor filters, samplers, phase-locked loops, and the like. It is a small-signal analysis like AC analysis, except the circuit is first linearized about a periodically varying operating point as opposed to a simple DC operating point. Linearizing about a periodically time-varying operating point allows transfer-functions that include frequency translation, whereas simply linearizing about a DC operating point could not because linear time-invariant circuits do not exhibit frequency translation. Also, the frequency of the sinusoidal stimulus is not constrained by the period of the large periodic solution.

Computing the small-signal response of a periodically varying circuit is a two step process. First, the small stimulus is ignored and the periodic steady-state response of the circuit to possibly large periodic stimulus is computed using PSS analysis. As a normal part of the PSS analysis, the periodically time-varying representation of the circuit is computed and saved for later use. The second step is applying the small stimulus to the periodically varying linear representation to compute the small signal response. This is done using the PAC analysis. A PAC analysis cannot be used alone, it must follow a PSS analysis. However, any number of periodic small-signal analyses such as PAC, PSP, PXF, PNoise, can follow a PSS analysis.

Modulated small signal measurements are possible using the Analog Artist(ADE) environment. The modulated option for PAC and other modulated parameters are set by Artist. PAC analyses with this option will produce results which could have limited use outside such environment. Direct Plot is configured to analyze these results and combine several wave forms to measure AM and PM response due to single sideband or modulated stimuli. For details, please see the SpectreRF User Guide.

Unlike AC analysis, PAC analysis can output the time-domain simulation results, by specifying the outputperiod parameter.

Unlike other analyses in Spectre, this analysis can only sweep frequency.

Definition

Name pac parameter=value ...
Parameters

Sweep interval parameters

1 start=0 Start sweep limit.
2 stop Stop sweep limit.
3 center Center of sweep.
4 span=0 Sweep limit span.
5 step Step size, linear sweep.
6 lin=50 Number of steps, linear sweep.
7 dec Points per decade.
8 log=50 Number of steps, log sweep.
9 values=[...] Array of sweep values.
10 sweeptype Specifies if the sweep frequency range is absolute frequency of input or if it is relative to the port harmonics. Possible values are absolute, relative or wrong-sweep-type.
11 relharmnum=1 Harmonic to which relative frequency sweep should be referenced.

Output parameters

12 sidebands=[...] Array of relevant sidebands for the analysis.
13 maxsideband=0 An alternative to the sidebands array specification, which automatically generates the array: [-maxsideband ... 0 ... +maxsideband ].
14 freqaxis Specifies whether the results should be output versus the input frequency, the output frequency, or the absolute value of the output frequency. Default is in for logarithmic frequency sweeps.
and absout otherwise. Possible values are absout, out or in.

15 save Signals to output. Possible values are all, lvl, allpub, lvlpub, selected, or none.

16 nestlvl Levels of subcircuits to output.

17 outputperiod=0.0 (no output) Time-domain output period. The time-domain small-signal response is computed for the period specified, rounded to the nearest integer multiple of the pss period.

Convergence parameters

18 tolerance=1e-9 Relative tolerance for linear solver.

19 gear_order=2 Gear order used for small-signal integration.

20 solver=turbo Solver type. Possible values are std or turbo.

21 oscsolver=turbo Oscillator solver type. Possible values are std or turbo.

Annotation parameters

22 annotate=sweep Degree of annotation. Possible values are no, title, sweep, status, or steps.

23 stats=no Analysis statistics. Possible values are no or yes.

24 title Analysis title.

Modulation conversion parameters

25 modulated=no Compute transfer functions/conversion between modulated sources and outputs. Possible values are single, first, second, or no.
You can select the set of periodic small-signal output frequencies of interest by setting either the `maxsideband` or the `sidebands` parameters. For a given set of n integer numbers representing the sidebands K1, K2, ... Kn, the output frequency at each sideband is computed as f(out) = f(in) + K1 * fund(pss), where f(in) represent the (possibly swept) input frequency, and fund(pss) represents the fundamental frequency used in the corresponding PSS analysis. Thus, when analyzing a down-converting mixer, while sweeping the RF input frequency, the most relevant sideband for IF output is K1= -1. When simulating an up-converting mixer, while sweeping IF input frequency, the most relevant sideband for RF output is K1= 1. By setting the `maxsideband` value to Kmax, all 2 * Kmax + 1 sidebands from -Kmax to +Kmax are generated.

The number of requested sidebands does not change substantially the simulation time. However, the `maxacfreq` of the corresponding PSS analysis should be set to guarantee that | max{f(out)} | is less than `maxacfreq`, otherwise the computed solution might be contaminated by aliasing effects. The PAC simulation is not executed for | f(in) | greater than `maxacfreq`. Diagnostic messages are printed for those extreme cases, indicating how `maxacfreq` should be set in the PSS analysis. In the majority of the simulations, however, this is not an issue, because `maxacfreq` is never allowed to be smaller than 40x the PSS fundamental.

With PAC the frequency of the stimulus and of the response are usually different (this is an important way in which PAC differs from AC). The `freqaxis` parameter is used to specify whether the results should be output versus the input frequency (`in`), the output frequency (`out`), or the absolute value of the output frequency (`absout`).

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (`step, lin, log, dec`) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. Alternatively, you may specify the particular values that the sweep parameter should take using the `values` parameter. If you give both a specific set of values and a set specified using a sweep range, the two sets are merged and collated before being used. All frequencies are in Hertz.
Parameter Index

In the following index, the number following each parameter name indicates where to find the
description of that parameter.

annotate 22  maxsideband 13  relharmnum 11  stop 2
center 3  modsource 29  save 15  sweeptype 10
dec 7  modulated 25  sidebands 12  title 24
dfreqaxis 14  moduppersideband 28  solver 20  tolerance 18
gear_order 19  nestlvl 16  span 4  values 9
inmodharmnum 26  oscsolver 21  start 1
lin 6  outmodharmvec 27  stats 23
log 8  outputperiod 17  step 5

Periodic Distortion Analysis (pdisto)

Description

This analysis computes the quasi-periodic steady-state (QPSS) response of a circuit that
operates on multiple time scales. A quasi-periodic signal has dynamics in multiple
fundamental frequencies. Closely spaced or incommensurate fundamentals cannot be
resolved by PSS efficiently. QPSS allows you to compute a circuit responses to several
moderately large input signals in addition to a strangely nonlinear tone which represents a LO
or clock signal. A typical example is the intermodulation distortion measurements of a mixer
with two closely spaced moderate input signals. QPSS treats one particular input signal
(usually the one that causes the most nonlinearity or the largest response) as the large signal,
and the others as moderate signals.

An initial transient analysis is carried out by first suppressing all moderate input signals. Then,
a number of (at least 2) stabilizing iterations with all signals activated is run. They are followed
by the shooting Newton method. QPSS employs the Mixed Frequency Time (MFT) algorithm
extended to multiple fundamental frequencies. For details of MFT algorithm, please see
Like PSS, QPSS uses the shooting Newton method as its backbone. However, instead of doing a single transient integration, each Newton iteration does a number of transient integrations of one large signal period.

Each of the integrations differs by a phase-shift in each moderate input signal. The number of integrations is determined by the numbers of harmonics of moderate fundamentals specified by the user. Given maxharms=[k1 k2 ... kn], the total number of integrations is $(2^*k2+1)*(2^*k3+1)*...*(2^*kn+1)$. As one consequence, the efficiency of the algorithm depends significantly on the number of harmonics required to model the responses of moderate fundamentals. As another consequence, the number of harmonics of the large fundamental does not significantly affect the efficiency of the shooting algorithm. The boundary conditions of a shooting interval are such that the time domain integrations are consistent with a frequency domain transformation with a shift of one large signal period.

QPSS inherits a majority of PSS parameters. A few new parameters are added. The most important ones are funds and maxharms. They replace PSS parameters, fund (or period) and harms, respectively. The funds parameter accepts a list of names of fundamentals that are present in the sources. These names are specified in the sources by parameter fundname. The first fundamental is considered as the large signal. A few heuristics can be used for picking the large fundamental.

1. Pick the one which is not a sinusoidal.
2. Pick the one which causes the most nonlinearity.
3. Pick the one which causes the largest response.

The maxharms parameter accepts a list of numbers of harmonics that are required to sufficiently model responses due to different fundamentals.

Definition

Name pdisto parameter=value ...

Parameters

QPSS fundamental parameters

1 funds=[...] Array of fundamental frequency names for fundamentals to use in analysis.
maxharms= [...] Array of number of harmonics of each fundamental to consider for each fundamental.

**Simulation interval parameters**

- **tstab=0.0 s** Extra stabilization time after the onset of periodicity for independent sources.
- **stabcycles=2** Stabilization cycles with both large and moderate sources enabled.
- **tstart=0.0 s** Initial transient analysis start time.

**Time-step parameters**

- **maxstep (s)** Maximum time step. Default derived from errpreset.
- **step=0.001 period s** Minimum time step that would be used solely to maintain the aesthetics of the results.

**Initial-condition parameters**

- **ic=all** What should be used to set initial condition. Possible values are dc, node, dev, or all.
- **skipdc=no** If yes, there will be no dc analysis for transient. Possible values are no, yes or sigrampup.
- **readic** File that contains initial condition.

**Convergence parameters**

- **readns** File that contains estimate of initial transient solution.
- **cmin=0 F** Minimum capacitance from each node to ground.
**Output parameters**

13 **save**
   Signals to output.
   **Possible values are** all, lvl, allpub, lvlpub, selected, or none.

14 **nestlvl**
   Levels of subcircuits to output.

15 **oppoint=no**
   Should operating point information be computed for initial timestep, and if so, where should it be sent.
   **Possible values are** no, screen, logfile, or rawfile.

16 **skipstart=starttime s**
   The time to start skipping output data.

17 **skipstop=stoptime s**
   The time to stop skipping output data.

18 **skipcount**
   Save only one of every skipcount points.

19 **strobeperiod (s)**
   The output strobe interval (in seconds of transient time).

20 **strobedelay=0 s**
   The delay (phase shift) between the skipstart time and the first strobe point.

21 **compression=no**
   Do data compression on output. See full description below.
   **Possible values are** no or yes.

22 **saveinit=no**
   If set, the waveforms for the initial transient before steady state are saved.
   **Possible values are** no or yes.

**State-file parameters**

23 **write**
   File to which initial transient solution (before steady-state) is to be written.

24 **writefinal**
   File to which final transient solution in steady-state is to be written.

25 **swapfile**
   Temporary file that holds steady-state information. Tells Spectre to use a regular file rather than virtual memory to hold the
periodic operating point. Use this option if Spectre complains about not having enough memory to complete this analysis.

**Integration method parameters**

26 method Integration method. Default derived from errpreset. Possible values are euler, trap, traponly, gear2, or gear2only.

**Accuracy parameters**

27 errpreset Selects a reasonable collection of parameter settings. Possible values are liberal, moderate or conservative.

28 relref Reference used for the relative convergence criteria. Default derived from errpreset. Possible values are pointlocal, alllocal, sigglobal, or allglobal.

29 lteratio Ratio used to compute LTE tolerances from Newton tolerance. Default derived from errpreset.

30 steadyratio Ratio used to compute steady state tolerances from LTE tolerance. Default derived from errpreset.

31 maxperiods Maximum number of simulated periods to reach steady-state.

32 itres=1e-4 Relative tolerance for linear solver.

33 fimitediff Options for finite difference method refinement after quasi-periodic shooting method. fimitediff is changed from no to samegrid automatically when readqpss and writeqpss are used to re-use QPSS results. Possible values are no, yes or refine.

**Annotation parameters**

34 stats=no Analysis statistics. Possible values are no or yes.
### Spectre Circuit Simulator Reference
#### Analysis Statements

| 35 | annotate=sweep | Degree of annotation. Possible values are no, title, sweep, status, or steps. |
| 36 | title | Analysis title. |

**Newton parameters**

| 37 | maxiters=5 | Maximum number of iterations per time step. |
| 38 | restart=yes | Restart the DC/PSS/QPSS solution from scratch if set to yes, if set to no, reuse the previous solution as initial guess. Possible values are no or yes. |

**Circuit age**

| 39 | circuitage (Years) | Stress Time. Age of the circuit used to simulate hot-electron degradation of MOSFET and BSIM circuits. |
| 40 | writeqpss | File to which final quasi-periodic steady-state solution is to be written. Small signal analyses such as qpac, qpxf and qpnoise can read in the steady-state solution from this file directly instead of running the qpss analysis again. |
| 41 | readqpss | File from which final quasi-periodic steady-state solution is to be read. Small signal analyses such as qpac, qpxf and qpnoise can read in the steady-state solution from this file directly instead of running the qpss analysis again. |

Most of QPSS analysis parameters are inherited from PSS analysis, and their meanings remain essentially unchanged. Two new important parameters are funds and maxharms. They replace and extend the role of fund and harms parameters of PSS analysis. One important difference is that funds accepts a list of fundamental names instead of actual frequencies. The frequencies associated with fundamentals are figured out automatically by the simulator. An important feature is that each input signal can be a composition of more than one sources. However, these sources must have the same fundamental name. For each fundamental name, its fundamental frequency is the greatest common factor of all frequencies associated with the name. Missing or not listing all fundamental names using the parameter funds will result in an amputation of the current simulation. However if maxharms is not given, a warning message will be issued, and the number of harmonics is defaulted to 1 for each of the fundamentals.
For QPSS analyses, the role of some PSS parameters is extended. The parameter `maxperiods` that controls the maximum number of shooting iterations for PSS analysis also controls the number of the maximum number of shooting iterations for QPSS analysis. Its default value is set to 50.

The `tstab` parameter controls both the length of the initial transient integration with only the clock tone activated and the number of stable iterations with moderate tones activated. The stable iterations are run before Newton iterations.

The `errpreset` parameter lets you adjust several simulator parameters to fit your needs. In most cases, `errpreset` should be the only parameter you need to adjust. If you want a fast simulation with reasonable accuracy, you might set `errpreset` to `liberal`. If you have some concern for accuracy, you might set `errpreset` to `moderate`. If accuracy is your main interest, you might set `errpreset` to `conservative`.

If users do not specify `steadyratio`, it is always 1.0, and it is not affected by `errpreset`. The following table shows the effect of `errpreset` on other parameters.

<table>
<thead>
<tr>
<th>Parameter defaults as a function of <code>errpreset</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>errpreset</code></td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>liberal</td>
</tr>
<tr>
<td>moderate</td>
</tr>
<tr>
<td>conservative</td>
</tr>
</tbody>
</table>

* : `Iteratio`=10.0 for conservative `errpreset` by default. Only if user specified `reltol` <= 1e-4*10.0/3.5, it will change to `Iteratio`=3.5.

The new `errpreset` settings include a new default `reltol` which is actually an enforced upper limit for appropriate setting. An increase of `reltol` above default will be ignored by the simulator. User can decrease this value in the options statement. The only way to increase `reltol` is to relax `errpreset`. Spectre sets the value of `maxstep` so that it is no larger than the value given in the table. Except for `reltol` and `maxstep`, `errpreset` does not change the value of any parameters you have explicitly set. The actual values used for the QPSS analysis are given in the log file.
If `errpreset` is not specified in the netlist, liberal settings will be used.

The default value for `compression` is no. The output file stores data for every signal at every time point for which Spectre calculates a solution. Spectre saves the x axis data only once, since every signal has the same x value. If `compression=yes`, Spectre writes data to the output file only if the signal value changes by at least 2*the convergence criteria. In order to save data for each signal independently, x axis information corresponding to each signal must be saved. If the signals stay at constant values for large periods of the simulation time, setting `compression=yes` results in a smaller output data file. If the signals in your circuit move around a lot, setting `compression=yes` results in a larger output data file.

**Parameter Index**

In the following index, the number following each parameter name indicates where to find the description of that parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotate</td>
<td>35</td>
</tr>
<tr>
<td>circuitage</td>
<td>39</td>
</tr>
<tr>
<td>cmin</td>
<td>12</td>
</tr>
<tr>
<td>compression</td>
<td>21</td>
</tr>
<tr>
<td>errpreset</td>
<td>27</td>
</tr>
<tr>
<td>finitediff</td>
<td>33</td>
</tr>
<tr>
<td>funds</td>
<td>1</td>
</tr>
<tr>
<td>ic</td>
<td>8</td>
</tr>
<tr>
<td>itres</td>
<td>32</td>
</tr>
<tr>
<td>lteratio</td>
<td>29</td>
</tr>
<tr>
<td>maxharms</td>
<td>2</td>
</tr>
<tr>
<td>maxiters</td>
<td>37</td>
</tr>
<tr>
<td>maxperiods</td>
<td>31</td>
</tr>
<tr>
<td>method</td>
<td>26</td>
</tr>
<tr>
<td>nestlvl</td>
<td>14</td>
</tr>
<tr>
<td>oppoint</td>
<td>15</td>
</tr>
<tr>
<td>readic</td>
<td>10</td>
</tr>
<tr>
<td>readns</td>
<td>11</td>
</tr>
<tr>
<td>readqpss</td>
<td>41</td>
</tr>
<tr>
<td>relref</td>
<td>28</td>
</tr>
<tr>
<td>restart</td>
<td>38</td>
</tr>
<tr>
<td>save</td>
<td>13</td>
</tr>
<tr>
<td>saveinit</td>
<td>22</td>
</tr>
<tr>
<td>skipcount</td>
<td>18</td>
</tr>
<tr>
<td>skipdc</td>
<td>9</td>
</tr>
<tr>
<td>skipstart</td>
<td>16</td>
</tr>
<tr>
<td>skipstop</td>
<td>17</td>
</tr>
<tr>
<td>tstart</td>
<td>5</td>
</tr>
<tr>
<td>tstab</td>
<td>3</td>
</tr>
<tr>
<td>tstart</td>
<td>5</td>
</tr>
<tr>
<td>steadyratio</td>
<td>30</td>
</tr>
<tr>
<td>stats</td>
<td>34</td>
</tr>
<tr>
<td>write</td>
<td>23</td>
</tr>
<tr>
<td>writefinal</td>
<td>24</td>
</tr>
<tr>
<td>writeqpss</td>
<td>40</td>
</tr>
<tr>
<td>strobedelay</td>
<td>20</td>
</tr>
<tr>
<td>strobeperiod</td>
<td>19</td>
</tr>
<tr>
<td>swapfile</td>
<td>25</td>
</tr>
<tr>
<td>title</td>
<td>36</td>
</tr>
</tbody>
</table>
Periodic Noise Analysis (pnoise)

Description

The Periodic Noise, or PNoise analysis is similar to the conventional noise analysis, except that it includes frequency conversion effects. Hence it is useful for predicting the noise behavior of mixers, switched-capacitor filters, and other periodically driven circuits. It is particularly useful for predicting the phase noise of autonomous circuits, such as oscillators.

PNoise analysis linearizes the circuit about the periodic operating point computed in the prerequisite PSS analysis. It is the periodically time-varying nature of the linearized circuit that accounts for the frequency conversion. In addition, the affect of a periodically time-varying bias point on the noise generated by the various components in the circuit is also included.

The time-average of the noise at the output of the circuit is computed in the form of a spectral density versus frequency. The output of the circuit is specified with either a pair of nodes or a probe component. To specify the output of a circuit with a probe, specify it using the oprobe parameter. If the output is voltage (or potential), choose a resistor or a port as the output probe. If the output is current (or flow), choose a vsource or iprobe as the output probe.

If the input-referred noise or noise figure is desired, specify the input source using the iprobe parameter. For input-referred noise, use either a vsource or isource as the input probe; for noise figure, use a port as the probe. Currently, only a vsource, an isource, or a port may be used as an input probe. If the input source is noisy, as is a port, the noise analysis will compute the noise factor (F) and noise figure (NF). To match the IEEE definition of noise figure, the input probe must be a port with no excess noise and its noisetemp must be set to 16.85C (290K). In addition, the output load must be a resistor or port and must be identified as the oprobe.

If port is specified as the input probe, then both input-referred noise and gain are referred back to the equivalent voltage source inside the port. S-parameter analysis calculates those values in traditional sense.

The reference sideband (refsideband) specifies which conversion gain is used when computing input-referred noise, noise factor, and noise figure. The reference sideband specifies the input frequency relative to the output frequency with:

\[ |f(\text{input})| = |f(\text{out}) + \text{refsideband} \times \text{fund(pss)}| \]

Use refsideband=0 when the input and output of the circuit are at the same frequency (such as with amplifiers and filters). When refsideband differs from 0, the single side-band noise figure is computed.
The noise analysis always computes the total noise at the output, which includes contributions from the input source and the output load. The amount of the output noise that is attributable to each noise source in the circuit is also computed and output individually. If the input source is identified (using iprobe) and is a vsource or isourec, the input-referred noise is computed, which includes the noise from the input source itself. Finally, if the input source is identified (using iprobe) and is noisy, as is the case with ports, the noise factor and noise figure are computed. Thus if

\[ \text{No} = \text{total output noise} \]
\[ \text{Ns} = \text{noise at the output due to the input probe (the source)} \]
\[ \text{Nsi} = \text{noise at the output due to the image harmonic at the source} \]
\[ \text{Nso} = \text{noise at the output due to harmonics other than input at the source} \]
\[ \text{Nl} = \text{noise at the output due to the output probe (the load)} \]
\[ \text{IRN} = \text{input referred noise} \]
\[ \text{G} = \text{gain of the circuit} \]
\[ \text{F} = \text{noise factor} \]
\[ \text{NF} = \text{noise figure} \]
\[ \text{Fdsb} = \text{double sideband noise factor} \]
\[ \text{NFdsb} = \text{double sideband noise figure} \]
\[ \text{Fieee} = \text{IEEE single sideband noise factor} \]
\[ \text{NFieee} = \text{IEEE single sideband noise figure} \]

then,

\[ \text{IRN} = \sqrt{\text{No}^2 / \text{G}^2} \]
\[ \text{F} = (\text{No}^2 - \text{Nl}^2) / \text{Ns}^2 \]
\[ \text{NF} = 10 \log_{10} \text{F} \]
\[ \text{Fdsb} = (\text{No}^2 - \text{Nl}^2) / (\text{Ns}^2 + \text{Nsi}^2) \]
\[ \text{NFdsb} = 10 \log_{10} \text{Fdsb} \]
Fieee = (No^2 - NI^2 - Nso^2)/Ns^2

NFieee = 10*log10(Fieee).

When the results are output, No is named out, IRN is named in, G is named gain, F, NF, Fdsb, NFdsb, Fieee, and NFieee are named F, NF, Fdsb, NFdsb, Fieee, and NFieee respectively.

Unlike other analyses in Spectre, this analysis can only sweep frequency.

**Definition**

Name [p] [n] pnoise parameter=value ...

The optional terminals (p and n) specify the output of the circuit. If you do not give the terminals, then you must specify the output with a probe component.

**Parameters**

**Sweep interval parameters**

1. start=0 Start sweep limit.
2. stop Stop sweep limit.
3. center Center of sweep.
4. span=0 Sweep limit span.
5. step Step size, linear sweep.
6. lin=50 Number of steps, linear sweep.
7. dec Points per decade.
8. log=50 Number of steps, log sweep.
9. values=[...] Array of sweep values.
10. sweeptype Specifies if the sweep frequency range is absolute frequency of input or if it is relative to the port harmonics. Possible values are absolute, relative or wrong-sweep-type.
Probe parameters

11 relharmnum=1 Harmonic to which relative frequency sweep should be referenced.

Probes

12 oprobe Compute total noise at the output defined by this component.

13 iprobe Refer the output noise to this component.

14 refsideband Conversion gain associated with this sideband is used when computing input-referred noise or noise figure.

Jitter parameters

15 thresholdvalue=0 PM Jitter is measured when the signal crosses this value.

16 crossingdirection=all Specifies for which transitions to measure the jitter. Possible values are all, rise or fall.

Output parameters

17 noisetype=sources Specifies if the pnoise analysis should output cross-power densities or noise source information. Possible values are sources, correlations, timedomain, or pmjitter.

18 maxsideband=7 Maximum sideband included when computing noise either up-converted or down-converted to the output by the periodic drive signal.

19 sidebands=[...] Array of relevant sidebands for the analysis.

20 save Signals to output. Possible values are all, lvl, allpub, lvlpub, selected, or none.

21 nestlvl Levels of subcircuits to output.
Spectre Circuit Simulator Reference
Analysis Statements

22 maxcycles=0 Maximum cycle correlation frequency included when computing noise either up-converted or down-converted to the output by the periodic drive signal.

23 cycles=[...] Array of relevant cycle frequencies. Valid only if noisetype=correlations.

24 noisesskipcount=-1 Calculate time-domain noise on only one of every noisesskipcount time points. When < 0, the parameter is ignored. When >=0, Simulator uses this parameter and ignores numberofpoints.

25 noisetimepoints=[...] Additional time points for time-domain noise analysis.

26 numberofpoints=5 Number of time points of interest in the period where to calculate time domain PSD. Simulator divides the period evenly into N segments (N=numberofpoints) and calculates time domain PSD on the starting time point of each segment. When < 0, the parameter is ignored.

27 saveallsidebands=no Save noise contributors by sideband. Possible values are no or yes.

Convergence parameters

28 tolerance=1e-9 Relative tolerance for linear solver.

29 gear_order=2 Gear order used for small-signal integration.

30 solver=turbo Solver type. Possible values are std or turbo.

31 oscsolver=turbo Oscillator solver type. Possible values are std or turbo.

Annotation parameters

32 annotate=sweep Degree of annotation. Possible values are no, title, sweep, status, or steps.
33 stats=no
   Analysis statistics.
   Possible values are no or yes.

34 title
   Analysis title.

In practice, noise can mix with each of the harmonics of the periodic drive signal applied in
the PSS analysis and end up at the output frequency. However, the PNoise analysis only
includes the noise that mixes with a finite set of harmonics that are typically specified using
the maxsideband parameter, but in special circumstances may be specified with the
sidebands parameter. If Ki represents sideband i, then

\[ f(\text{noise_source}) = f(\text{out}) + K_i \cdot f\text{(fund(pss))} \]

The maxsideband parameter specifies the maximum \(|K_i|\) included in the PNoise calculation.
Thus, noise at frequencies less than \(f(\text{out}) - \text{maxsideband} \cdot f\text{(fund(pss))}\) and greater than
\(f(\text{out}) + \text{maxsideband} \cdot f\text{(fund(pss))}\) are ignored. If selected sidebands are specified using the
sidebands parameter, then only those are included in the calculation. Care should be taken
when specifying the sidebands because the results will be in error if you do not include a
sideband that contributes significant noise to the output.

The number of requested sidebands does not change substantially the simulation time.
However, the maxacfreq of the corresponding PSS analysis should be set to guarantee that
\(|\text{max}\{f(\text{noise_source})\}|\) is less than maxacfreq, otherwise the computed solution might be
contaminated by aliasing effects. The PNoise simulation is not executed for \(|f(\text{out})|\) greater
than maxacfreq. Diagnostic messages are printed for those extreme cases, indicating
which maxacfreq should be set in the PSS analysis. In the majority of the simulations,
however, this is not an issue, because maxacfreq is never allowed to be smaller than 40x
the PSS fundamental.

Phase Noise measurements are possible using the Analog Artist(ADE) environment. Two
pnoise analyses are preconfigured for this simulation and most of the parameters are set by
Artist. First pnoise analysis named mod1 is a regular noise analysis and can be used
independently. The second pnoise correlation analysis called mod2 has a limited use outside
of the Artist environment. Direct Plot is configured to analyze these results and combine
several wave forms to measure AM and PM components of the output noise. For details,
please see the SpectreRF User Guide.

You can specify sweep limits by giving the end points or by providing the center value and the
span of the sweep. Steps can be linear or logarithmic, and you can specify the number of
steps or the size of each step. You can give a step size parameter (step, lin, log, dec) to
determine whether the sweep is linear or logarithmic. If you do not give a step size parameter,
the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when
this ratio is 10 or greater. Alternatively, you may specify the particular values that the sweep
parameter should take using the values parameter. If you give both a specific set of values
and a set specified using a sweep range, the two sets are merged and collated before being used. All frequencies are in Hertz.

Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

```
annotate 32   maxcycles 22   refsideband 14   step 5
center 3      maxsideband 18  relharmnum 11   stop 2
 crossingdirection 16  nestlvl 21   save 20   sweeptype 10
 cycles 23     noiseskipcount 24  saveallsidebands 27  thresholdvalue 15
 dec 7         noisetimepoints 25  sidebands 19   title 34
 gear_order 29  noisertype 17   solver 30   tolerance 28
 iprobe 13     numberofpoints 26  span 4   values 9
 lin 6          oprobe 12   start 1
 log 8          oscsolver 31   stats 33
```

Periodic S-Parameter Analysis (psp)

Description

The periodic SP (PSP) analysis is used to compute scattering and noise parameters for n-port circuits that exhibit frequency translation, such as mixers. It is a small-signal analysis like SP analysis, except, as in PAC and PXF, the circuit is first linearized about a periodically varying operating point as opposed to a simple DC operating point. Linearizing about a periodically time-varying operating point allows the computation of S-parameters between circuit ports that convert signals from one frequency band to another. PSP can also calculate
noise parameters in frequency-converting circuits. PSP computes noise figure (both single-
sideband and double-sideband), input referred noise, equivalent noise parameters, and noise
correlation matrices. As in PNoise, but unlike SP, the noise features of the PSP analysis
include noise folding effects due to the periodically time-varying nature of the circuit.

Computing the n-port S-parameters and noise parameters of a periodically varying circuit is
a two step process. First, the small stimulus is ignored and the periodic steady-state response
of the circuit to possibly large periodic stimulus is computed using PSS analysis. As a normal
part of the PSS analysis, the periodically time-varying representation of the circuit is
computed and saved for later use. The second step is applying small-signal excitations to
compute the n-port S-parameters and noise parameters. This is done using the PSP analysis.
A PSP analysis cannot be used alone, it must follow a PSS analysis. However, any number
of periodic small-signal analyses such as PAC, PSP, PXF, PNoise, can follow a single PSS
analysis.

Unlike other analyses in Spectre, this analysis can only sweep frequency.

**Definition**

*Name psp parameter=value ...

**Parameters**

**Sweep interval parameters**

1 start=0 Start sweep limit.
2 stop Stop sweep limit.
3 center Center of sweep.
4 span=0 Sweep limit span.
5 step Step size, linear sweep.
6 lin=50 Number of steps, linear sweep.
7 dec Points per decade.
8 log=50 Number of steps, log sweep.
9 values= [...] Array of sweep values.
Spectre Circuit Simulator Reference
Analysis Statements

10 **sweeptype**
   Specifies if the sweep frequency range is absolute frequency of
   input or if it is relative to the port harmonics.
   Possible values are *absolute*, *relative* or *wrong-sweep-type*.

**Port parameters**

11 **ports=**
   List of active ports. Ports are numbered in the order given. For
   purposes of noise figure computation, the input is considered
   port 1 and the output is port 2.

12 **portharmsvec=**
   List of harmonics active on specified list of ports. Must have a
   one-to-one correspondence with the ports vector.

13 **harmsvec=**
   List of harmonics, in addition to ones associated with specific
   ports by portharmsvec, that are active.

**Output parameters**

14 **freqaxis**
   Specifies whether the results should be output versus the input
   frequency, the output frequency, or the absolute value of the
   input frequency. Default is *in*.
   Possible values are *absin*, *in* or *out*.

**Noise parameters**

15 **donoise=yes**
   Perform noise analysis. If oprobe is specified as a valid port, this
   is set to yes, and a detailed noise output is generated.
   Possible values are *no* or *yes*.

**Probe parameters**

16 **maxsideband=7**
   Maximum sideband included when computing noise either up-
   converted or down-converted to the output by the periodic drive
   signal.

**Convergence parameters**

17 **tolerance=1e-9**
   Relative tolerance for linear solver.
18 gear_order=2  Gear order used for small-signal integration.

19 solver=turbo  Solver type.
    Possible values are std or turbo.

20 oscsolver=turbo  Oscillator solver type.
    Possible values are std or turbo.

**Annotation parameters**

21 annotate=sweep  Degree of annotation.
    Possible values are no, title, sweep, status, or steps.

22 stats=no  Analysis statistics.
    Possible values are no or yes.

23 title  Analysis title.

To specify the PSP analysis the port and port harmonic relations must be specified. You can select the ports of interest by setting the port parameter and the set of periodic small-signal output frequencies of interest by setting portharmsvec or the harmsvec parameters. For a given set of n integer numbers representing the harmonics K1, K2, ... Kn, the scattering parameters at each port are computed at the frequencies f(scattered)= f(rel) + Ki * fund(pss), where f(rel) represents the relative frequency of a signal incident on a port, f(scattered) represents the frequency to which the relevant scattering parameter represents the conversion, and fund(pss) represents the fundamental frequency used in the corresponding PSS analysis.

Thus, when analyzing a down-converting mixer, with signal in the upper sideband, and sweeping the RF input frequency, the most relevant harmonic for RF input is Ki= 1 and for IF output Ki= 0. Hence we can associate K2=1 with the IF port and K1=0 with the RF port. S21 will represent the transmission of signal from the RF to IF, and S11 the reflection of signal back to the RF port. If the signal was in the lower sideband, then a choice of K1=-1 would be more appropriate.

Either portharmsvec or harmsvec parameters can be used to specify the harmonics of interest. If portharmsvec is given, the harmonics must be in one-to-one correspondence with the ports, with each harmonic associated with a single port. If harmonics are specified in the optional harmsvec parameter, then all possible frequency-translating scattering parameters associated with the specified harmonics are computed.

With PSP the frequency of the input and of the response are usually different (this is an important way in which PSP differs from SP). Because the PSP computation involves inputs
and outputs at frequencies that are relative to multiple harmonics, the freqaxis and sweeptyp
parameters behave somewhat differently in PSP than in PAC and PXF.

The sweeptyp parameter controls the way the frequencies in the PSP analysis are swept. Specifying a relative sweep indicates to sweep relative to the analysis harmonics (not the PSS fundamental) and an absolute sweep is a sweep of the absolute input source frequency. For example, with a PSS fundamental of 100MHz, portharmsvec set to [9 1] to examine a downconverting mixer, sweeptyp=relative, and a sweep range of f(rel)=0->50MHz, then S21 would represent the strength of signal transmitted from the input port in the range 900->950MHz to the output port at frequencies 100->150MHz. Using sweeptyp=absolute and sweeping the frequency from 900->950MHz would calculate the same quantities, since f(abs)=900->950MHz, and f(rel) = f(abs) - K1 * fund(pss) = 0->50MHz, because K1=9 and fund(pss) = 100MHz.

Usually it is not necessary to sweep frequency in PSP over more than one fundamental PSS period.

The freqaxis parameter is used to specify whether the results should be output versus the scattered frequency at the input port (in), the scattered frequency at the output port (out), or the absolute value of the frequency swept at the input port (absin).

Unlike in PAC/PXF/PNoise, increasing the number of requested ports and harmonics will increase the simulation time substantially.

To ensure accurate results in PSP, the maxacfreq of the corresponding PSS analysis should be set to guarantee that | max{f(scattered)} | is less than maxacfreq, otherwise the computed solution might be contaminated by aliasing effects.

PSP analysis also computes noise figures, equivalent noise sources, and noise parameters. The noise computation, which is skipped only when donoise=no, requires additional simulation time. If

\[ \text{No} = \text{total output noise at frequency } f \]
\[ \text{Ns} = \text{noise at the output due to the input probe (the source)} \]
\[ \text{Nsi} = \text{noise at the output due to the image harmonic at the source} \]
\[ \text{Nso} = \text{noise at the output due to harmonics other than input at the source} \]
\[ \text{Ni} = \text{noise at the output due to the output probe (the load)} \]
\[ \text{IRN} = \text{input referred noise} \]
\[ G = \text{gain of the circuit} \]
F = noise factor (single side band)
NF = noise figure (single side band)
Fdsb = double sideband noise factor
NFdsb = double sideband noise figure
Fieee = IEEE single sideband noise factor
NFieee = IEEE single sideband noise figure

Then,
\[ IRN = \sqrt{\frac{N_0^2}{G^2}} \]
\[ F = \frac{(N_0^2 - N_l^2)}{N_s^2} \]
\[ NF = 10 \log_{10}(F) \]
\[ Fdsb = \frac{(N_0^2 - N_l^2)}{(N_s^2 + N_{si}^2)} \]
\[ NFdsb = 10 \log_{10}(Fdsb) \]
\[ Fieee = \frac{(N_0^2 - N_l^2 - N_{so}^2)}{N_s^2} \]
\[ NFieee = 10 \log_{10}(Fieee) \]

When the results are output, IRN is named **in**, G is named **gain**, F, NF, Fdsb, NFdsb, Fieee, and NFieee are named **F**, **NF**, **Fdsb**, **NFdsb**, **Fieee**, and **NFieee** respectively. Note that the gain computed by PSP is the voltage gain from the actual circuit input to the circuit output, not the gain from the internal port voltage source to the output.

To ensure accurate noise calculations, the **maxsideband** or **sidebands** parameters must be set to include the relevant noise folding effects. **maxsideband** is only relevant to the noise computation features of PSP.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (**step**, **lin**, **log**, **dec**) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. Alternatively, you may specify the particular values that the sweep parameter should take using the **values** parameter. If you give both a specific set of values and a set specified using a sweep range, the two sets are merged and collated before being used. All frequencies are in Hertz.
Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate 21  harmsvec 13  ports 11  stop 2
center 3  lin 6  solver 19  sweeptype 10
dec 7  log 8  span 4  title 23
donoise 15  maxsideband 16  start 1  tolerance 17
freqaxis 14  oscsolver 20  stats 22  values 9
gear_order 18  portharmsvec 12  step 5

Periodic Steady-State Analysis (pss)

Description

This analysis computes the periodic steady-state (PSS) response of a circuit, with a simulation time independent of the time-constants of the circuit. Also, it sets the circuits periodic operating point, which can then be used during a periodic time-varying small-signal analysis, such as PAC, PXF, and PNOISE.

PSS analysis is capable of handling both autonomous (non-driven) and driven (non-autonomous) circuits. Autonomous circuits are time-invariant circuits that have time-varying responses. Thus, autonomous circuits generate non-constant waveforms even though they are not driven by a time-varying stimulus. Driven circuits require some time-varying stimulus to generate a time-varying response. The most common example of an autonomous circuit is an oscillator. Common driven circuits include amplifiers, filters, mixers, etc.

With driven circuits the user specifies the analysis period, or its corresponding fundamental frequency fund. The period must be an integer multiple of the period of the drive signal or signals. Autonomous circuits have no drive signal and the actual period of oscillation is not known precisely by the user in advance. Instead, the user specifies an estimate of the oscillation period and PSS analysis computes the precise period along with the periodic solution waveforms.
When applied to autonomous circuits, PSS analysis requires the user to specify a pair of nodes, \( p \) and \( n \). In fact this is how PSS analysis determines whether it is being applied to an autonomous or a driven circuit. If the pair of nodes is supplied, PSS assumes the circuit is autonomous; if not, the circuit is assumed to be driven.

A PSS analysis consists of two phases, an initial transient phase, which allows the circuit to be initialized, and the shooting phase, which is where the periodic steady-state solution is computed. The transient phase consists of three intervals.

The first starts at \( t_{\text{start}} \), which is normally 0, and continues through the onset of periodicity \( t_{\text{onset}} \) for the independent sources. The onset of periodicity, which is automatically generated, is the minimum time for which all sources are periodic. The second is an optional user specified stabilization interval whose length is \( t_{\text{stab}} \). The final interval whose length is \( t_{\text{period}} \) for driven circuits, or \( 4 \times t_{\text{period}} \) for autonomous circuits has a special use for the autonomous PSS analysis, i.e., the PSS analysis monitors the waveforms in the circuit and develops a better estimate of the oscillation period. Once the initial transient phase is complete, the shooting interval begins. In this phase, the circuit is repeatedly simulated over one period while adjusting the initial condition (and the period when applied to autonomous circuits) to find the periodic steady-state solution.

Typically the process takes three to five such iterations to reach steady-state. Upon completion, if requested by the user, the frequency-domain response is computed. For driven circuits, one can use writepss and readpss to reuse the results in a previous simulation.

**Definition**

```
Name  [p]  [n] pss parameter=value ...
```

**Parameters**

**Simulation interval parameters**

1. **period (s)**
   - Steady state analysis period (or its estimate for autonomous circuits).

2. **fund (Hz)**
   - Alternative to period specification. Steady state analysis fundamental frequency (or its estimate for autonomous circuits).

3. **tstab=0.0 s**
   - Extra stabilization time after the onset of periodicity for independent sources.

4. **tstart=0.0 s**
   - Initial transient analysis start time.
**Time-step parameters**

5 maxstep (s)  
Maximum time step. Default derived from errpreset.

6 maxacfreq  
Maximum frequency requested in a subsequent periodic small-signal analysis. Default derived from errpreset and harms.

7 step=0.001 period s  
Minimum time step that would be used solely to maintain the aesthetics of the results.

**Initial-condition parameters**

8 ic=all  
What should be used to set initial condition. Possible values are dc, node, dev, or all.

9 skipdc=no  
If yes, there will be no dc analysis for transient. Possible values are no, yes or sigrampup.

10 readic  
File that contains initial condition.

**Convergence parameters**

11 readns  
File that contains estimate of initial transient solution.

12 cmin=0 F  
Minimum capacitance from each node to ground.

**Output parameters**

13 harms=9  
Number of harmonics to output when outputtype=freq or all.

14 harmsvec=[...]

Array of desired harmonics. Alternate form of harms that allows selection of specific harmonics.

15 outputtype= all'forPSS,'freq'forQPSS'

Output type. Possible values are all, time or freq.

16 save  
Signals to output. Possible values are all, lvl, allpub, lvlpub, selected, or none.
17 nestlvl Levels of subcircuits to output.

18 oppoint=no Should operating point information be computed for initial timestep, and if so, where should it be sent. Possible values are no, screen, logfile, or rawfile.

19 skipstart=starttime s The time to start skipping output data.

20 skipstop=stoptime s The time to stop skipping output data.

21 skipcount Save only one of every skipcount points.

22 strobeperiod (s) The output strobe interval (in seconds of transient time).

23 strobedelay=0 s The delay (phase shift) between the skipstart time and the first strobe point.

24 compression=no Do data compression on output. See full description below. Possible values are no or yes.

25 saveinit=no If set, the waveforms for the initial transient before steady state are saved. Possible values are no or yes.

State-file parameters

26 write File to which initial transient solution (before steady-state) is to be written.

27 writefinal File to which final transient solution in steady-state is to be written.

28 swapfile Temporary file that holds steady-state information. Tells Spectre to use a regular file rather than virtual memory to hold the periodic operating point. Use this option if Spectre complains about not having enough memory to complete this analysis.

29 writepss File to which the converged steady-state solution is to be written. finitediff is set to yes automatically to improve PSS results.
30 readpss  File from which a previously converged steady-state solution is to be read. PSS loads the solution and checks the residue of the circuit equations only. The solution is re-used if the residue is satisfying. Otherwise, the solution is re-converged using the finite difference method.

31 checkpss=yes  If yes, the previous PSS results (from readpss file) will be checked and redo PSS+MIC if any condition has changed. If no, program assume nothing change and use the solution from file without check/redo PSS+MIC.. Possible values are no or yes.

Integration method parameters

32 method  Integration method. Default derived from errpreset. Possible values are euler, trap, traponly, gear2, or gear2only.

33 tstabmethod  Integration method used in stabilization time. Default is traponly for autonomous circuits, or is derived from errpreset for driven circuits. Possible values are euler, trap, traponly, gear2, or gear2only.

Accuracy parameters

34 errpreset  Selects a reasonable collection of parameter settings. Possible values are liberal, moderate or conservative.

35 relref  Reference used for the relative convergence criteria. Default derived from errpreset. Possible values are pointlocal, alllocal, sigglobal, or allglobal.

36 lteratio  Ratio used to compute LTE tolerances from Newton tolerance. Default derived from errpreset.

37 steadyratio  Ratio used to compute steady state tolerances from LTE tolerance. Default derived from errpreset.

38 maxperiods  Maximum number of simulated periods to reach steady-state.
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>itres=1e-4</td>
</tr>
<tr>
<td>40</td>
<td>finitediff</td>
</tr>
<tr>
<td>41</td>
<td>highorder</td>
</tr>
<tr>
<td>42</td>
<td>psaratio=1</td>
</tr>
<tr>
<td>43</td>
<td>maxorder=16</td>
</tr>
<tr>
<td>44</td>
<td>fullpssvec</td>
</tr>
</tbody>
</table>

**Annotation parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>stats=no</td>
</tr>
<tr>
<td>46</td>
<td>annotate=sweep</td>
</tr>
<tr>
<td>47</td>
<td>title</td>
</tr>
</tbody>
</table>

**Newton parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>maxiters=5</td>
</tr>
<tr>
<td>49</td>
<td>restart=yes</td>
</tr>
</tbody>
</table>
**Circuit age**

50 circuitage (Years) Stress Time. Age of the circuit used to simulate hot-electron degradation of MOSFET and BSIM circuits.

The initial transient analysis provides a flexible mechanism to direct the circuit to a particular steady-state solution of interest, and to avoid undesired solutions. Another usage of the initial transient simulation is helping convergence by eliminating large but fast decaying modes that are present in many circuits. For example, in case of driven circuits, consider the reset signal in the figure below.

```
_ _ _ _ _ _ _ _
clock ______| | | | | | | | | | |
__________________________
reset __________________|
_____       _____       _____
Q2    ________________________|     |_____|     |_____|
^        ^         ^           ^
|<-tstab->|<- period->|
```

In the figure above, the initial transient analysis is executed from tstart to tstop. If initial transient results are relevant, you can output them by setting saveinit to yes. The steady-state results are always computed for the specified period, from tinit to tstop. By default, tstart and tstop are set to zero, while tinit, tonset and tstop are always automatically generated.

It happens in some circuits that the linearity of the relationship between the initial and final state depends on when the shooting interval begins. Conceptually, when the shooting interval begins should not matter, as long as it is after the time when the stimuli have become periodic, because the periodic response repeats endlessly. However in practice, one can improve the convergence by starting at a good point, and degrade the convergence, which slows the analysis, by starting in a bad spot. In general, it is best to try to avoid starting the shooting interval at a point where the circuit is undergoing strong nonlinear behavior. For example, in switch-capacitor filters it is best if tinit falls at the beginning of a clock transition, preferably a transition that follows a relatively long period of settling. If instead tinit occurred during a clock transition or soon after one, then it is likely the opamps would be undergoing slew-rate
limiting at the start of the shooting interval, which would act to slow convergence. Switching mixers follow similar rules.

When applying PSS analysis to oscillators, it is necessary to start the oscillator, just as you would if you were simulating the turn-on transient of the oscillator using transient analysis. The Designers Guide to Spice and Spectre [K. S. Kundert, Kluwer Academic Publishers, 1995] describes techniques for starting oscillators in some depth. In summary, there are two techniques for starting oscillators, using initial conditions, or using a brief impulsive stimulus. Initial conditions would be provided for the components of the oscillators resonator. If an impulsive stimulus is used, it should be applied so as to couple strongly into the oscillatory mode of the circuit, and poorly into any other long-lasting modes, such as those associated with bias circuitry. Either way, once the trigger is applied to start the oscillator, it is important to allow the oscillator to run for a while before the shooting methods are applied to compute the steady-state result. To do so, specify an additional stabilization interval using the \texttt{tstab} parameter. In practice, an additional stabilization interval often improves convergence.

By default, only the time-domain results are computed. If you specify either \texttt{harm} or \texttt{harmvec} or set \texttt{outputtype} to \texttt{freq} or \texttt{all}, the frequency-domain results will also be computed. If frequency-domain results are requested, but the desired harmonics are not specified, its default value is 10. The time-domain output waveform generation can be inhibited by setting \texttt{outputtype} to \texttt{freq}.

The accuracy of the results does not depend on the number of harmonics that are requested, only on the accuracy parameters, which are set in the same fashion as in the transient analysis. Besides a few new parameters, like \texttt{steadyratio} and \texttt{maxacfreq}, all the others parameters work in PSS analysis in the exact same fashion as they work on transient analysis.

Several parameters determine the accuracy of the PSS analysis. \texttt{reltol} and \texttt{abstol} control the accuracy of the discretized equation solution. These parameters determine how well charge is conserved and how accurately steady-state or equilibrium points are computed. You can set the integration error, or the errors in the computation of the circuit dynamics (such as time constants), relative to \texttt{reltol} and \texttt{abstol} by setting the \texttt{iteratio} parameter.

The \texttt{steadyratio} parameter adjusts the maximum allowed mismatch in node voltages or current branches from the beginning to the end of the steady-state period. This value is multiplied by the \texttt{iteratio} and \texttt{reltol} to determine the convergence criterion. The relative convergence norm is printed out along with the actual mismatch value at the end of each iteration, thus indicating the progress of the steady-state iteration.
The parameter `maxperiods` controls the maximum number of shooting iterations for PSS analysis. Its default value is set to 20 for driven PSS and 50 for autonomous PSS.

The `finitediff` parameter allows the use of finite difference (FD) after shooting. Usually this will eliminate the above mismatch in node voltages or current branches. It can also refine the grid of time steps. In some cases, numerical error of the linear solver still introduces a mismatch. One can set `steadyratio` to a smaller value to activate a tighter tolerance for the iterative linear solver. If `finitediff` is set to no, FD method is turned off. If it is set to yes, pss applies FD method and trying to improve the beginning small time steps if necessary. If it is set to refine, pss applies FD method and tries to refine the time steps. When the simulation uses 2nd-order method, uniform 2nd order gear is used. `finitediff` is changed from no to yes automatically when `readpss` and `writepss` are used to re-use PSS results.

The `maxacfreq` parameter is used to automatically adjust the `maxstep` to reduce errors due to aliasing in frequency-domain results. By default, the `maxacfreq` is set to 4x the frequency of the largest requested harmonic, but is never set to less than 40x the fundamental.

The parameter `relref` determines how the relative error is treated. The `relref` options are:

- `relref=pointlocal`: Compares the relative errors in quantities at each node to that node alone.
- `relref=alllocal`: Compares the relative errors at each node to the largest values found for that node alone for all past time.
- `relref=sigglobal`: Compares relative errors in each of the circuit signals to the maximum for all signals at any previous point in time.
- `relref=allglobal`: Same as `relref=sigglobal` except that it also compares the residues (KCL error) for each node to the maximum of that nodes past history.

The `errpreset` parameter lets you adjust the simulator parameters to fit your needs quickly. In most cases, it should also be the only parameter you need to adjust.

Guidelines for using `errpreset` in driven circuits are described in the following. If the circuit contains only one periodic tone and you are only interested in obtaining the periodic operating point, you might set `errpreset` to `liberal`, which gives a reasonably accurate result and the fastest simulation speed. If the circuit contains more than one periodic tone and you are interested in intermodulation results, you might set `errpreset` to `moderate`, which gives a very accurate result. If you want a very low noise floor in your simulation result and accuracy is your main interest, you might set `errpreset` to `conservative`. Multi-interval Chebyshev (MIC) is activated automatically for both moderate and conservative settings unless you explicitly set `highorder=no`. MIC falls back to the original method if it encounters difficulty converging. If you sets `highorder=yes`, MIC will try harder to converge.
The effect of \textit{errpreset} on other parameters for driven circuits is shown in the following table.

### Parameter defaults and estimated numerical noise floor in simulation result as a function of \textit{errpreset}

<table>
<thead>
<tr>
<th>errpreset</th>
<th>reltol</th>
<th>relref</th>
<th>method</th>
<th>Iteratio</th>
<th>steadyratio</th>
<th>noisefloor</th>
</tr>
</thead>
<tbody>
<tr>
<td>liberal</td>
<td>1e-3</td>
<td>sigglobal</td>
<td>traponly</td>
<td>3.5</td>
<td>0.001</td>
<td>-70dB</td>
</tr>
<tr>
<td>moderate</td>
<td>1e-3</td>
<td>alllocal</td>
<td>gear2only</td>
<td>3.5</td>
<td>0.001</td>
<td>-120dB+mic</td>
</tr>
<tr>
<td>conservative</td>
<td>1e-4</td>
<td>alllocal</td>
<td>gear2only</td>
<td>*</td>
<td>0.01</td>
<td>-200dB+mic</td>
</tr>
</tbody>
</table>

*: \textit{Iteratio}=10.0 for conservative \textit{errpreset}. Only if user specified reltol \(\leq 1e-4 \times 10.0/3.5\), it will change to \textit{Iteratio}=3.5.

The new \textit{errpreset} settings include a new default \textit{reltol} which is actually an upper limit. An increase of \textit{reltol} above default will be ignored by the simulator. User can decrease this value in the options statement. The only way to increase \textit{reltol} is to relax \textit{errpreset}.

Estimated numerical noise floor is for a weakly nonlinear circuit with a successful MIC simulation. For a linear circuit, the noise floor is even lower and for a very nonlinear circuit, you might need to tighten \textit{psaratio} or increase \textit{maxacfreq} to achieve this noise floor. The actual values used for the PSS analysis are given in the log file.

Guidelines for using \textit{errpreset} in autonomous circuits are described in the following. If you want a fast simulation with reasonable accuracy, you might set \textit{errpreset} to \textit{liberal}. If you have some concern for accuracy, you might set \textit{errpreset} to \textit{moderate}. If accuracy is your main interest, you might set \textit{errpreset} to \textit{conservative}.

The effect of \textit{errpreset} on other parameters for autonomous circuits is shown in the following table.

### Parameter defaults as a function of \textit{errpreset}

<table>
<thead>
<tr>
<th>errpreset</th>
<th>reltol</th>
<th>relref</th>
<th>method</th>
<th>Iteratio</th>
<th>steadyratio</th>
<th>maxstep</th>
</tr>
</thead>
<tbody>
<tr>
<td>liberal</td>
<td>1e-3</td>
<td>sigglobal</td>
<td>traponly</td>
<td>3.5</td>
<td>0.001</td>
<td>period/50</td>
</tr>
<tr>
<td>moderate</td>
<td>1e-4</td>
<td>alllocal</td>
<td>gear2only</td>
<td>3.5</td>
<td>0.1</td>
<td>period/200</td>
</tr>
<tr>
<td>conservative</td>
<td>1e-5</td>
<td>alllocal</td>
<td>gear2only</td>
<td>*</td>
<td>0.1</td>
<td>period/400</td>
</tr>
</tbody>
</table>
* : Iteratio=10.0 for conservative errpreset by default. Only if user specified reltol <= 1e-4*10.0/3.5, it will change to Iteratio=3.5.

The value of reltol can be decreased from default in the options statement. The only way to increase reltol is to relax errpreset. Spectre sets the value of maxstep so that it is no larger than the value given in the table. Except for reltol and maxstep, errpreset does not change the value of any parameters you have explicitly set. The actual values used for the PSS analysis are given in the log file.

If errpreset is not specified in the netlist, liberal settings will be used.

If the circuit you are simulating can have infinitely fast transitions (for example, a circuit that contains nodes with no capacitance), Spectre might have convergence problems. To avoid this, you must prevent the circuit from responding instantaneously. You can accomplish this by setting cmin, the minimum capacitance to ground at each node, to a physically reasonable nonzero value. This often significantly improves Spectre convergence.

You may specify the initial condition for the transient analysis by using the ic statement or the ic parameter on the capacitors and inductors. If you do not specify the initial condition, the DC solution is used as the initial condition. The ic parameter on the transient analysis controls the interaction of various methods of setting the initial conditions. The effects of individual settings are:

ic=dc: Any initial condition specifiers are ignored, and the DC solution is used.

ic=node: The ic statements are used, and the ic parameter on the capacitors and inductors are ignored.

ic=dev: The ic parameters on the capacitors and inductors are used, and the ic statements are ignored.

ic=all: Both the ic statements and the ic parameters are used, and the ic parameters override the ic statements.

If you specify an initial condition file with the readic parameter, initial conditions from the file are used, and any ic statements are ignored.

Once you specify the initial conditions, Spectre computes the actual initial state of the circuit by performing a DC analysis. During this analysis, Spectre forces the initial conditions on nodes by using a voltage source in series with a resistor whose resistance is rforce (see options).

With the ic statement it is possible to specify an inconsistent initial condition (one that cannot be sustained by the reactive elements). Examples of inconsistent initial conditions include setting the voltage on a node with no path of capacitors to ground or setting the current
through a branch that is not an inductor. If you initialize Spectre inconsistently, its solution jumps; that is, it changes instantly at the beginning of the simulation interval. You should avoid such changes if possible because Spectre can have convergence problems while trying to make the jump.

You can skip the DC analysis entirely by using the parameter `skipdc`. If the DC analysis is skipped, the initial solution will be either trivial, or given in the file you specified by the `readic` parameter, or, if the `readic` parameter is not given, the values specified on the `ic` statements. Device based initial conditions are not used for `skipdc`. Nodes that you do not specify with the `ic` file or `ic` statements will start at zero. You should not use this parameter unless you are generating a nodeset file for circuits that have trouble in the DC solution; it usually takes longer to follow the initial transient spikes that occur when the DC analysis is skipped than it takes to find the real DC solution. The `skipdc` parameter might also cause convergence problems in the transient analysis.

The possible settings of parameter `skipdc` and their meanings are:

`skipdc=no`: Initial solution is calculated using the normal DC analysis (default).

`skipdc=yes`: Initial solution is given in the file specified by the `readic` parameter or the values specified on the `ic` statements.

`skipdc=sigrampup`: Independent source values start at 0 and ramp up to their initial values in the first phase of the simulation. The waveform production in the time-varying independent source is enabled after the rampup phase. The rampup simulation is from `tstart` to time=0 s, and the main simulation is from time=0 s to `tstab`. If the `tstart` parameter is not specified, the default `tstart` time is set to `-0.1*tstab`.

Nodesets help find the DC or initial transient solution. You can supply them in the circuit description file with nodeset statements, or in a separate file using the `readns` parameter. When nodesets are given, Spectre computes an initial guess of the solution by performing a DC analysis while forcing the specified values onto nodes by using a voltage source in series with a resistor whose resistance is `rforce`. Spectre then removes these voltage sources and resistors and computes the true solution from this initial guess.

Nodesets have two important uses. First, if a circuit has two or more solutions, nodesets can bias the simulator towards computing the desired one. Second, they are a convergence aid. By estimating the solution of the largest possible number of nodes, you might be able to eliminate a convergence problem or dramatically speed convergence.

When you simulate the same circuit many times, we suggest that you use both the `write` and `readns` parameters and give the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.
Nodesets and initial conditions have similar implementation but produce different effects. Initial conditions actually define the solution, whereas nodesets only influence it. When you simulate a circuit with a transient analysis, Spectre forms and solves a set of differential equations. However, differential equations have an infinite number of solutions, and a complete set of initial conditions must be specified in order to identify the desired solution. Any initial conditions you do not specify are computed by the simulator to be consistent. The transient waveforms then start from initial conditions. Nodesets are usually used as a convergence aid and do not affect the final results. However, in a circuit with more than one solution, such as a latch, nodesets bias the simulator towards finding the solution closest to the nodeset values.

The `method` parameter specifies the integration method. The possible settings and their meanings are:

- `method=euler`: Backward-Euler is used exclusively.
- `method=traponly`: Trapezoidal rule is used almost exclusively.
- `method=trap`: Backward-Euler and the trapezoidal rule are used.
- `method=gear2only`: Gears second-order backward-difference method is used almost exclusively.
- `method=gear2`: Backward-Euler and second-order Gear are used.

The trapezoidal rule is usually the most efficient when you want high accuracy. This method can exhibit point-to-point ringing, but you can control this by tightening the error tolerances. For this reason, though, if you choose very loose tolerances to get a quick answer, either backward-Euler or second-order Gear will probably give better results than the trapezoidal rule. Second-order Gear and backward-Euler can make systems appear more stable than they really are. This effect is less pronounced with second-order Gear or when you request high accuracy.

Spectre provides two methods for reducing the number of output data points saved: strob ing, based on the simulation time, and skipping time points, which saves only every Nth point.

The parameters `strobeperiod` and `strobedelay` control the strob ing method. `strobeperiod` sets the interval between points that you want to save, and `strobedelay` sets the offset within the period relative to `skipstart`. The simulator forces a time step on each point to be saved, so the data is computed, not interpolated.

The skipping method is controlled by `skipcount`. If this is set to N, then only every Nth point is saved.
The parameters skipstart and skipstop apply to both data reduction methods. Before skipstart and after skipstop, Spectre saves all computed data.

The default value for compression is no. The output file stores data for every signal at every time point for which Spectre calculates a solution. Spectre saves the x axis data only once, since every signal has the same x value. If compression=yes, Spectre writes data to the output file only if the signal value changes by at least 2*the convergence criteria. In order to save data for each signal independently, x axis information corresponding to each signal must be saved. If the signals stay at constant values for large periods of the simulation time, setting compression=yes results in a smaller output data file. If the signals in your circuit move around a lot, setting compression=yes results in a larger output data file.

**Parameter Index**

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate  46      itres  39      readic  10      step  7
checkpss  31      lteratio  36      readns  11      strobedelay  23
circuitage  50      maxacfreq  6      readpss  30      strobeperiod  22
cmin  12      maxiters  48      relref  35      swapfile  28
compression  24      maxorder  43      restart  49      title  47
errpreset  34      maxperiods  38      save  16      tstab  3
finitediff  40      maxstep  5      saveinit  25      tstabmethod  33
fullpssvec  44      method  32      skipcount  21      tstart  4
fund  2      nestlvl  17      skipdc  9      write  26
harms  13      oppoint  18      skipstart  19      writefinal  27
harmssvec  14      outputtype  15      skipstop  20      writepss  29
highorder  41      period  1      stats  45
Periodic Transfer Function Analysis (pxf)

Description

A conventional transfer function analysis computes the transfer function from every source in the circuit to a single output. It differs from a conventional AC analysis in that the AC analysis computes the response from a single stimulus to every node in the circuit. The difference between PAC and PXF analysis are similar. The Periodic Transfer Function or PXF analysis computes the transfer functions from any source at any frequency to a single output at a single frequency. Thus, like PAC analysis, PXF analysis includes frequency conversion effects.

The PXF analysis directly computes such useful quantities as conversion efficiency (transfer function from input to output at desired frequency), image and sideband rejection (input to output at undesired frequency), and LO feed-through and power supply rejection (undesired input to output at all frequencies).

As with a PAC, PSP, and PNoise analyses, a PXF analysis must follow a PSS analysis.

Unlike other analyses in Spectre, this analysis can only sweep frequency.

Definition

Name  [p]  [n]  pxf parameter=value ...

The optional terminals (p and n) specify the output of the circuit. If you do not give the terminals, then you must specify the output with a probe component.

Parameters

Sweep interval parameters

1  start=0  Start sweep limit.

2  stop  Stop sweep limit.
Spectre Circuit Simulator Reference
Analysis Statements

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>center</td>
<td>Center of sweep.</td>
</tr>
<tr>
<td>4</td>
<td>span=0</td>
<td>Sweep limit span.</td>
</tr>
<tr>
<td>5</td>
<td>step</td>
<td>Step size, linear sweep.</td>
</tr>
<tr>
<td>6</td>
<td>lin=50</td>
<td>Number of steps, linear sweep.</td>
</tr>
<tr>
<td>7</td>
<td>dec</td>
<td>Points per decade.</td>
</tr>
<tr>
<td>8</td>
<td>log=50</td>
<td>Number of steps, log sweep.</td>
</tr>
<tr>
<td>9</td>
<td>values=[...]</td>
<td>Array of sweep values.</td>
</tr>
<tr>
<td>10</td>
<td>sweeptype</td>
<td>Specifies if the sweep frequency range is absolute frequency of input or if it is relative to the port harmonics. Possible values are absolute, relative or wrong-sweep-type.</td>
</tr>
<tr>
<td>11</td>
<td>relharmnum=1</td>
<td>Harmonic to which relative frequency sweep should be referenced.</td>
</tr>
</tbody>
</table>

**Probe parameters**

| 12 | probe | Compute every transfer function to this probe component. |

**Output parameters**

| 13 | stimuli=sources | Stimuli used for pxf analysis. Possible values are sources or nodes_and_terminals. |
| 14 | sidebands=[...] | Array of relevant sidebands for the analysis. |
| 15 | maxsideband=0 | An alternative to the sidebands array specification, which automatically generates the array: [-maxsideband ... 0 ... +maxsideband]. |
| 16 | freqaxis | Specifies whether the results should be output versus the input frequency, the output frequency, or the absolute value of the input frequency. Default is out for logarithmic frequency sweeps and absin otherwise. Possible values are absin, in or out. |
17 save  Signals to output.  
Possible values are all, lvl, allpub, lvlpub, selected, or none.

18 nestlvl  Levels of subcircuits to output.

**Convergence parameters**

19 tolerance=1e-9  Relative tolerance for linear solver.

20 gear_order=2  Gear order used for small-signal integration.

21 solver=turbo  Solver type.  
Possible values are std or turbo.

22 oscsolver=turbo  Oscillator solver type.  
Possible values are std or turbo.

**Annotation parameters**

23 annotate=sweep  Degree of annotation.  
Possible values are no, title, sweep, status, or steps.

24 stats=no  Analysis statistics.  
Possible values are no or yes.

25 title  Analysis title.

**Modulation conversion parameters**

26 modulated=no  Compute transfer functions/conversion between modulated sources and outputs.  
Possible values are single, first, second, or no.

27 outmodharmnum=1  Harmonic for the PXF output modulation.

28 inmodharmvec= [...] Harmonic list for the PXF modulated sources.

29 moduppersideband=1  Index of the upper sideband included in the modulation of an output for PAC or an input for PXF.
The variable of interest at the output can be voltage or current, and its frequency is not constrained by the period of the large periodic solution. While sweeping the selected output frequency, you can select the periodic small-signal input frequencies of interest by setting either the maxsideband or the sidebands parameters. For a given set of \( n \) integer numbers representing the sidebands \( K_1, K_2, \ldots, K_n \), the input signal frequency at each sideband is computed as \( f(\text{in}) = f(\text{out}) + K_i \times \text{fund(pss)} \), where \( f(\text{out}) \) represent the (possibly swept) output signal frequency, and \( \text{fund(pss)} \) represents the fundamental frequency used in the corresponding PSS analysis. Thus, when analyzing a down-converting mixer, and sweeping the IF output frequency, \( K_i = +1 \) for the RF input represents the first upper-sideband, while \( K_i = -1 \) for the RF input represents the first lower-sideband. By setting the maxsideband value to \( K_{\text{max}} \), all \( 2 \times K_{\text{max}} + 1 \) sidebands from \(-K_{\text{max}}\) to \(+K_{\text{max}}\) are be selected.

The number of requested sidebands does not change substantially the simulation time. However, the maxacfreq of the corresponding PSS analysis should be set to guarantee that \( | \text{max}(f(\text{in})) | \) is less than maxacfreq, otherwise the computed solution might be contaminated by aliasing effects. The PXF simulation is not executed for \( | f(\text{out}) | \) greater than maxacfreq. Diagnostic messages are printed for those extreme cases, indicating how maxacfreq should be set in the PSS analysis. In the majority of the simulations, however, this is not an issue, because maxacfreq is never allowed to be smaller than \( 40 \times \) the PSS fundamental.

With PXF the frequency of the stimulus and of the response are usually different (this is an important way in which PXF differs from XF). The freqaxis parameter is used to specify whether the results should be output versus the input frequency (\( \text{in} \)), the output frequency (\( \text{out} \)), or the absolute value of the input frequency (\( \text{absin} \)).

You can specify the output with a pair of nodes or a probe component. Any component with two or more terminals can be a voltage probe. When there are more than two terminals, they are grouped in pairs; and you use the portv parameter to select the appropriate pair of terminals. Alternatively, you can simply specify a voltage to be the output by giving a pair of nodes on the PXF analysis statement.

Any component that naturally computes current as an internal variable can be a current probe. If the probe component computes more than one current, you use the porti parameter to select the appropriate current. It is an error to specify both portv and porti. If neither is specified, the probe component provides a reasonable default.

The stimuli parameter specifies what is used for the inputs for the transfer functions. There are two choices. stimuli=sources indicates that the sources present in the circuit should be used. The xfmag parameters provided by the sources may be used to adjust the computed gain to compensate for gains or losses in a test fixture. One can limit the number of sources in hierarchical netlists by using the save and nestlvl parameters. stimuli=nodes_and_terminals indicates that all possible transfer functions should be computed.
This is useful when it is not known in advance which transfer functions are interesting. Transfer functions for nodes are computed assuming that a unit magnitude flow (current) source is connected from the node to ground. Transfer functions for terminals are computed assuming that a unit magnitude value (voltage) source is connected in series with the terminal. By default, the transfer functions from a small set of terminals are computed. If transfer functions from specific terminals are desired, specify the terminals in the save statement. You must use the :probe modifier (ex. Rout:1:probe) or specify useprobes=yes on the options statement. If transfer functions from all terminals are desired, specify currents=all and useprobes=yes on the options statement.

Modulated small signal measurements are possible using the Analog Artist (ADE) environment. The modulated option for PXF and other modulated parameters are set by Artist. PXF analyses with this option will produce results which could have limited use outside such environment. Direct Plot is configured to analyze these results and combine several wave forms to measure AM and PM transfer function from single sideband or modulated stimuli to the specified output. For details, please see the SpectreRF User Guide.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, dec) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. Alternatively, you may specify the particular values that the sweep parameter should take using the values parameter. If you give both a specific set of values and a set specified using a sweep range, the two sets are merged and collated before being used. All frequencies are in Hertz.

**Parameter Index**

In the following index, the number following each parameter name indicates where to find the description of that parameter.

- annotate 23
- maxsideband 15
- save 17
- stop 2
- center 3
- modulated 26
- sidebands 14
- sweeptype 10
- dec 7
- moduppersideband 29
- solver 21
- title 25
- freqaxis 16
- nestlvl 18
- span 4
- tolerance 19
- gear_order 20
- oscsolver 22
- start 1
- values 9
PZ Analysis (pz)

Description

The PZ analysis linearizes the circuit about the DC operating point and computes the poles and zeros of the linearized network. To compute zeros, users need to specify input sources and output voltages or currents. If no input or output are given, then only poles are computed. In case there are frequency dependent components, poles and zeros are computed by approximating those components as equivalent conductances and capacitances evaluated at 1Hz. The PZ analysis currently uses direct solver for better accuracy. Small to medium circuit size will achieve better performance.

(Note: A frequency dependent component means the capacitance or conductance equivalent representation of the component is frequency varying. Examples are transmission lines or bjts with excess phases. A linear capacitor is not a frequency dependent component.)

Spectre can perform the analysis while sweeping a parameter. The parameter can be temperature, component instance parameter, component model parameter, or netlist parameter. If changing a parameter affects the DC operating point, the operating point is recomputed on each step. You can sweep the parameter temp or a netlist parameter by giving the parameter name with no dev, or mod parameter. After the analysis has completed, the modified parameter returns to its original value.

Pole-zero cancellation is performed when a neighboring pole-zero pair is located within absdiff distance. The distance is also determined relatively as reldiff times the magnitude of the pole or zero. Spectre uses the larger value of the two distances to do cancellation.

Definition

Name ... pz parameter=value ...
Parameters

_Probe parameters_

1 iprobe  
Input probe for zeros of the transfer function.

2 oprobe  
Output probe for zeros of the transfer function.

_Port parameters_

3 portv  
Voltage across this oprobe port is output of the analysis.

4 porti  
Current through this oprobe port is output of the analysis. Should be used when oprobe is a voltage source or a current probe.

_Sweep interval parameters_

5 start=0  
Start sweep limit.

6 stop  
Stop sweep limit.

7 center  
Center of sweep.

8 span=0  
Sweep limit span.

9 step  
Step size, linear sweep.

10 lin=50  
Number of steps, linear sweep.

11 dec  
Points per decade.

12 log=50  
Number of steps, log sweep.

13 values=[... ]  
Array of sweep values.

_Sweep variable parameters_

14 dev  
Device instance whose parameter value is to be swept.

15 mod  
Model whose parameter value is to be swept.
16 param Name of parameter to sweep.
17 freq (Hz) Frequency at which components will be evaluated in setting up the linearized network.

**State-file parameters**

18 readns File that contains estimate of DC solution (nodeset).

**Output parameters**

19 oppoint=no Should operating point information be computed, and if so, where should it be sent. Possible values are no, screen, logfile, or rawfile.
20 zeroonly=no If set, only zeros are requested. Possible values are no or yes.

**Filtering parameters**

21 fmax (Hz) Maximum pole and zero frequency value to filter out spurious poles and zeros. This parameter is passed to psf outputs for plotting filtering.
22 docancel=yes If set, pole-zero cancellation is requested. Possible values are no or yes.
23 absdiff=1e-6 Hz Pole-Zero cancel absolute distance in Hz.
24 reldiff=1e-4 Pole-Zero cancel relative distance.

**Convergence parameters**

25 prevoppoint=no Use operating point computed on the previous analysis. Possible values are no or yes.
26 restart=yes Restart the DC solution from scratch if any condition has changed. If not, use the previous solution as initial guess. Possible values are no or yes.
Annotation parameters

27 stats=no  
Analysis statistics.  
Possible values are no or yes.

28 annotate=sweep  
Degree of annotation.  
Possible values are no, title, sweep, status, or steps.

29 title  
Analysis title.

Examples

1. mypz pz
   Pole analysis will be performed.

2. mypz2 (n1 n2) pz iprobe=VIN
   Input is VIN, output is voltage difference between nodes n1 and n2.
   Both pole and zero analyses will be performed.

3. mypz3 (n1 n2) pz iprobe=I1
   - input is I1, output is voltage difference between n1 and n2.
   - both pole and zero analyses will be performed.

4. mypz4 pz iprobe=VIN oprobe=IP1 porti=1
   - input is VIN, output is current through IP1, where IP1 is an iprobe.
   - both pole and zero analyses will be performed.

5. mypz5 pz iprobe=VIN oprobe=V3 porti=1
   - input is VIN, output is current through voltage source V3.
   - both pole and zero analyses will be performed.

6. mypz6 pz iprobe=VIN oprobe=R3 portv=1
   - input is VIN, output is the voltage across the resistor R3.
   - both pole and zero analyses will be performed.

7. mypz7 (n1 n2) pz iprobe=I1 param=temp start=25 stop=100 step=25
   - sweep temperature from 25 C to 100 C with increment of 25 C.
8. parameters rval=2.0
   R2 3 4 resistor r=rval
   ...
   sweep1 sweep param=rval start=1 stop=10 step=1 {
       mypz8 (n1 n2) iprobe=VIN
   }
   - external sweep parameter rval from 1 to 10 with increment of 1.

9. mypz9 (n1 n2) pz iprobe=VIN docancel=no
   - do not perform pole-zero cancellation.

Note: porti allows users to select a current associated with a specific device given in oprobe as an output. This device, however, has to have its terminal currents as network variables. Thus, to avoid confusion, porti should be used with voltage sources and current probes, and other components that have voltage-defined branches exclusively.

Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

absdiff 23   iprobe 1   portv 3   step 9
annotate 28   lin 10   prevoppoint 25   stop 6
center 7   log 12   readns 18   title 29
dec 11   mod 15   reldiff 24   values 13
dev 14   oppoint 19   restart 26   zeroonly 20
docancel 22   oprobe 2   span 8
fmax 21   param 16   start 5
freq 17   porti 4   stats 27
Quasi-Periodic AC Analysis (qpac)

Description

The quasi periodic AC (QPAC) analysis is used to compute transfer functions for circuits that exhibit multitone frequency translation. Such circuits include mixers, switched-capacitor filters, samplers, phase-locked loops, and the like. It is a small-signal analysis like AC analysis, except the circuit is first linearized about a quasiperiodically varying operating point as opposed to a simple DC operating point. Linearizing about a quasiperiodically time-varying operating point allows transfer-functions that include frequency translation, whereas simply linearizing about a DC operating point could not because linear time-invariant circuits do not exhibit frequency translation. Also, the frequency of the sinusoidal stimulus is not constrained by the period of the large periodic solution.

Computing the small-signal response of a quasiperiodically varying circuit is a two step process. First, the small stimulus is ignored and the quasiperiodic steady-state response of the circuit to possibly large periodic stimuli is computed using QPSS analysis. As a normal part of the QPSS analysis, the quasiperiodically time-varying representation of the circuit is computed and saved for later use. The second step is to apply the small stimulus to the periodically varying linear representation to compute the small signal response. This is done using the QPAC analysis.

A QPAC analysis cannot be used alone, it must follow a QPSS analysis. However, any number of quasiperiodic small-signal analyses such as QPAC, QPSP, QPXF, QPNOISE, can follow a QPSS analysis.

Unlike other analyses in Spectre, this analysis can only sweep frequency.

Definition

Name qpac parameter=value ...

Parameters

Sweep interval parameters

1 start=0

Starte is an absolute frequency, i.e. actual frequency; or if it is relative to the "relharmvec" sideband frequency. In QPSP, relative means relative to the input port frequency.

Possible values are absolute, relative or wrong-sweep-type.
11 relharmvec= [...] Sideband - vector of QPSS harmonics - to which relative frequency sweep should be referenced.

Output parameters

12 sidevec= [...] Array of relevant sidebands for the analysis.

13 clockmaxharm=0 An alternative to the sidevec array specification, which automatically generates the array: [ -clockmaxharm ... 0 ... +clockmaxharms]- maxharms(QPSS)[2]...0...maxharms(QPSS)[2] ][...,]

14 freqaxis Specifies whether the results should be output versus the input frequency, the output frequency, or the absolute value of the output frequency. Default is in for logarithmic frequency sweeps and absout otherwise. Possible values are absout, out or in.

15 save Signals to output. Possible values are all, lvl, allpub, lvlpub, selected, or none.

16 nestlvl Levels of subcircuits to output.

Convergence parameters

17 tolerance=1e-9 Relative tolerance for linear solver.

18 gear_order=2 Gear order used for small-signal integration, 1 or 2.

19 solver=turbo Solver type. Possible values are std or turbo.

Annotation parameters

20 annotate=sweep Degree of annotation. Possible values are no, title, sweep, status, or steps.

21 stats=no Analysis statistics. Possible values are no or yes.

22 title Analysis title.
User can select the set of periodic small-signal output frequencies of interest by setting either the `clockmaxharm` or `sidevec` parameters. Sidebands are vectors in QPAC. Assume we have one large tone and one moderate tone in QPSS. A sideband K1 will be represented as `[K1_1 K1_2]`. Corresponding frequency is

\[ K1_1 \times \text{fund(large tone of QPSS)} + K1_2 \times \text{fund(moderate tone of QPSS)} \]

We assume that there are L large and moderate tones in QPSS analysis and a given set of n integer vectors representing the sidebands

\[ K1 = \{ K1_1,...K1_j,..., K1_L \} , K2, ... Kn. \]

The output frequency at each sideband is computed as

\[ f(\text{out})= f(\text{in}) + \sum_{j=1}^{L} \{K_i_j \times \text{fund}_j(qpss)}\],

where \(f(\text{in})\) represents the (possibly swept) input frequency, and \(\text{fund}_j(qpss)\) represents the fundamental frequency used in the corresponding QPSS analysis. Thus, when analyzing a down-converting mixer, while sweeping the RF input frequency, the most relevant sideband for IF output is \{-1, 0\}. When simulating an up-converting mixer, while sweeping IF input frequency, the most relevant sideband for RF output is \{1, 0\}. User would enter `sidevec` as a sequence of integer numbers, separated by spaces. The set of vectors \{1 1 0\} \{1 -1 0\} \{1 1 1\} becomes `sidevec`=[1 1 0 1 -1 0 1 1 1]. For `clockmaxharm`, only the large tone - first fundamental will be affected by this entry, all the rest - moderate tones - will be limited by `maxhars`, specified for a QPSS analysis. Given `maxhars`=[k1max k2max ... knmax] in QPSS and `clockmaxharm`=Kmax all \((2^{*}Kmax + 1)\*(2^{*}k2max+1)*...*(2^{*}knmax+1)\) sidebands are generated.

The number of requested sidebands changes substantially the simulation time.

With QPAC the frequency of the stimulus and of the response are usually different (this is an important way in which QPAC differs from AC). The `freqaxis` parameter is used to specify whether the results should be output versus the input frequency (`in`), the output frequency (`out`), or the absolute value of the output frequency (`absout`).

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (`step`, `lin`, `log`, `dec`) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. Alternatively, you may specify the particular values that the sweep parameter should take using the `values` parameter. If you give both a specific set of values and a set specified using a sweep range, the two sets are merged and collated before being used. All frequencies are in Hertz.
Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate 20     lin 6     solver 19     sweepype 10
center 3        log 8     span 4       title 22
clockmaxharm 13 nestlvl 16 start 1      tolerance 17
dec 7            relharmvec 11 stats 21     values 9
freqaxis 14      save 15    step 5
gear_order 18    sidevec 12  stop 2

Quasi-Periodic Noise Analysis (qpnoise)

Description

The Quasi-Periodic Noise, or QPNOISE analysis is similar to the conventional noise analysis, except that it includes frequency conversion and intermodulation effects. Hence is it useful for predicting the noise behavior of mixers, switched-capacitor filters, and other periodically or quasi-periodically driven circuits.

QPNOISE analysis linearizes the circuit about the quasi-periodic operating point computed in the prerequisite QPSS analysis. It is the quasiperiodically time-varying nature of the linearized circuit that accounts for the frequency conversion and intermodulation. In addition, the affect of a quasi-periodically time-varying bias point on the noise generated by the various components in the circuit is also included.

The time-average of the noise at the output of the circuit is computed in the form of a spectral density versus frequency. The output of the circuit is specified with either a pair of nodes or a probe component. To specify the output of a circuit with a probe, specify it using the oprobe parameter. If the output is voltage (or potential), choose a resistor or a port as the output probe. If the output is current (or flow), choose a vsource or iprobe as the output probe.

If the input-referred noise is desired, specify the input source using the iprobe parameter. Currently, only a vsource, an isource, or a port may be used as an input probe. If the
input source is noisy, as is a port, the noise analysis will compute the noise factor (F) and noise figure (NF). To match the IEEE definition of noise figure, the input probe must be a port with no excess noise and its noisetemp must be set to 16.85°C (290K). In addition, the output load must be a resistor or port and must be identified as the oprobe.

If port is specified as the input probe, then both input-referred noise and gain are referred back to the equivalent voltage source inside the port. S-parameter analysis calculates those values in traditional sense.

The reference sideband (refsideband) specifies which conversion gain is used when computing input-referred noise, noise factor, and noise figure. The reference sideband satisfies:

$$|f_{\text{input}}| = |f_{\text{out}} + \text{refsideband frequency shift}|.$$  

The reference sideband option (refsidebandoption) specifies whether to consider the input at the frequency or the input at the individual quasi-periodic sideband specified. Note that different sidebands can lead to the same frequency.

Sidebands are vectors in QPNOISE. Assume we have one large tone and one moderate tone in QPSS. A sideband Ki will be a vector [Ki_1 Ki_2]. It gives the frequency at

$$Ki_1 \times \text{fund(large tone of QPSS)} + Ki_2 \times \text{fund(moderate tone of QPSS)}$$

Use refsideband=[0 0 ...] when the input and output of the circuit are at the same frequency (such as with amplifiers and filters).

The noise analysis always computes the total noise at the output, which includes contributions from the input source and the output load. The amount of the output noise that is attributable to each noise source in the circuit is also computed and output individually. If the input source is identified (using iprobe) and is a vsource or isource, the input-referred noise is computed, which includes the noise from the input source itself. Finally, if the input source is identified (using iprobe) and is noisy, as is the case with ports, the noise factor and noise figure are computed. Thus if

No = total output noise
Ns = noise at the output due to the input probe (the source)
Nsi = noise at the output due to the image harmonic at the source
Nso = noise at the output due to harmonics other than input at the source
Nl = noise at the output due to the output probe (the load)
IRN = input referred noise

G = gain of the circuit

F = noise factor

NF = noise figure

Fdsb = double sideband noise factor

NFdsb = double sideband noise figure

Fieee = IEEE single sideband noise factor

NFieee = IEEE single sideband noise figure

then,

\[ IRN = \sqrt{\frac{No^2}{G^2}} \]

\[ F = \frac{No^2 - Ni^2}{Ns^2} \]

\[ NF = 10 \times \log_{10}(F) \]

\[ Fdsb = \frac{No^2 - Ni^2}{Ns^2 + Nsi^2} \]

\[ NFdsb = 10 \times \log_{10}(Fdsb) \]

\[ Fieee = \frac{No^2 - Ni^2 - Nso^2}{Ns^2} \]

\[ NFieee = 10 \times \log_{10}(Fieee). \]

When the results are output, No is named \texttt{out}, IRN is named \texttt{in}, G is named \texttt{gain}, F, NF, Fdsb, NFdsb, Fieee, and NFieee are named F, NF, Fdsb, NFdsb, Fieee, and NFieee respectively.

The computation of gain and IRN in QPNOISE assumes that the circuit under test is impedance-matched to the input source. This can introduce inaccuracy into the gain and IRN computation.

Unlike other analyses in Spectre, this analysis can only sweep frequency.

\textbf{Definition}

\texttt{Name } [p] [n] \texttt{qpnoise parameter=value ...}
The optional terminals (p and n) specify the output of the circuit. If you do not give the terminals, then you must specify the output with a probe component.

**Parameters**

**Sweep interval parameters**

1. `start=0`  
   Start sweep limit.

2. `stop`  
   Stop sweep limit.

3. `center`  
   Center of sweep.

4. `span=0`  
   Sweep limit span.

5. `step`  
   Step size, linear sweep.

6. `lin=50`  
   Number of steps, linear sweep.

7. `dec`  
   Points per decade.

8. `log=50`  
   Number of steps, log sweep.

9. `values=[...]`  
   Array of sweep values.

10. `sweeptype`  
    Specifies if the sweep frequency range is an absolute frequency, i.e. actual frequency; or if it is relative to the "relharmvec" sideband frequency. In QPSP, relative means relative to the input port frequency. Possible values are absolute, relative or wrong-sweep-type.

11. `relharmvec=[...]`  
    Sideband - vector of QPSS harmonics - to which relative frequency sweep should be referenced.

**Probe parameters**

12. `oprobe`  
    Compute total noise at the output defined by this component.

13. `iprobe`  
    Refer the output noise to this component.
14 refsideband= [...] Conversion gain associated with this sideband is used when computing input-referred noise or noise figure.

15 refsidebandoption= individual
Whether to view the sideband as a specification of a frequency or a specification of an individual sideband.
Possible values are freq or individual.

Output parameters

16 clockmaxharm=7 Maximum large tone harmonics included when computing noise either up-converted or down-converted to the output by that large signal..

17 sidevec= [...] Array of relevant sidebands for the analysis.

18 save Signals to output.
Possible values are all, lvl, allpub, lvlpub, selected, or none.

19 nestlvl Levels of subcircuits to output.

20 saveallsidebands=no Save noise contributors by sideband.
Possible values are no or yes.

Convergence parameters

21 tolerance=1e-9 Relative tolerance for linear solver.

22 gear_order=2 Gear order used for small-signal integration, 1 or 2.

23 solver=turbo Solver type.
Possible values are std or turbo.

Annotation parameters

24 annotate=sweep Degree of annotation.
Possible values are no, title, sweep, status, or steps.
Analysis statements.

Possible values are no or yes.

Analysis title.

In practice, noise can mix with each of the harmonics of the quasi-periodic drive signal applied in the QPSS analysis and end up at the output frequency. The QPNOISE analysis only includes the noise that mixes with a finite set of harmonics that are specified using the \texttt{clockmaxharm} and \texttt{sidevec} parameters. Sidebands are vectors in quasi-periodic analyses. For one large tone and one moderate tone in QPSS, a sideband \(K_1\) will be represented as \([K_{1_1} K_{1_2}]\). Corresponding frequency shift is

\[
K_{1_1} \times \text{fund(large tone of QPSS)} + K_{1_2} \times \text{fund(moderate tone of QPSS)}
\]

We assume that there are \(L\) large and moderate tones in QPSS analysis and a given set of \(n\) integer vectors representing the sidebands

\[
K_1 = \{ K_{1_1},...,K_{1_j},...,K_{1_L}\},
\]

\(K_2, ... , K_n\).

If \(K_i\) represents sideband \(i\), then

\[
f(\text{noise source}) = f(\text{out}) + \sum_{j=1}^{L} K_{i_j} \times \text{fund}_j(\text{qpss}),
\]

The \texttt{clockmaxharm} parameter only affects clock frequency. It can be less or more than \(\text{maxharms}[1]\) in QPSS. Moderate tones are limited by \texttt{maxharms} specified in QPSS. If selected sidebands are specified using the \texttt{sidevec} parameter, then only those are included in the calculation. Care should be taken when specifying the \texttt{sidevec} or \texttt{clockmaxharm} QPNOISE and \texttt{maxharms} in QPSS. Noise results will be in error if you do not include the sidebands that contribute significant noise to the output.

The number of requested sidebands will change substantially the simulation time.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (\texttt{step, lin, log, dec}) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. Alternatively, you may specify the particular values that the sweep parameter should take using the \texttt{values} parameter. If you give both a specific set of values and a set specified using a sweep range, the two sets are merged and collated before being used. All frequencies are in Hertz.
Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate  24  log  8  saveallsidebands  stop  2
      20
center  3  nestlvl  19  sidevec  17  sweeptype  10
clockmaxharm  16  oprobe  12  solver  23  title  26
deck  7  refsideband  14  span  4  tolerance  21
gear_order  22  refsidebandoption  15  start  1  values  9
      1
iprobe  13  relharmvec  11  stats  25
lin  6  save  18  step  5

Quasi-Periodic S-Parameter Analysis (qpss)

Description

The quasi-periodic SP (QPSP) analysis is used to compute scattering and noise parameters for n-port circuits that exhibit frequency translation. Such circuits include mixers, switched-capacitor filters, samplers, phase-locked loops, and the like. It is a small-signal analysis like SP analysis, except, as in QPAC and QPXF, the circuit is first linearized about a quasiperiodically varying operating point as opposed to a simple DC operating point. Linearizing about a quasiperiodically time-varying operating point allows the computation of S-parameters between circuit ports that convert signals from one frequency band to another. QPSP can also calculate noise parameters in frequency-converting circuits. QPSP computes noise figure (both single-sideband and double-sideband), input referred noise, equivalent noise parameters, and noise correlation matrices. As in QPNOISE, but unlike SP, the noise features of the QPSP analysis include noise folding effects due to the periodically time-varying nature of the circuit.

Computing the n-port S-parameters and noise parameters of a quasiperiodically varying circuit is a two step process. First, the small stimulus is ignored and the quasiperiodic steady-state response of the circuit to possibly large periodic stimulus is computed using QPSS...
analysis. As a normal part of the QPSS analysis, the quasiperiodically time-varying representation of the circuit is computed and saved for later use. The second step is applying small-signal excitations to compute the n-port S-parameters and noise parameters. This is done using the QPSP analysis. A QPSP analysis cannot be used alone, it must follow a QPSS analysis. However, any number of periodic small-signal analyses such as QPAC, QPSP, QPXF, QPNOISE, can follow a single QPSS analysis.

Unlike other analyses in Spectre, this analysis can only sweep frequency.

**Definition**

Name qpsp parameter=value ...

**Parameters**

**Sweep interval parameters**

1. **start=0**
   - Start sweep limit.
2. **stop**
   - Stop sweep limit.
3. **center**
   - Center of sweep.
4. **span=0**
   - Sweep limit span.
5. **step**
   - Step size, linear sweep.
6. **lin=50**
   - Number of steps, linear sweep.
7. **dec**
   - Points per decade.
8. **log=50**
   - Number of steps, log sweep.
9. **values= [...]**
   - Array of sweep values.
10. **sweeptype**
    - Specifies if the sweep frequency range is an absolute frequency, i.e. actual frequency; or if it is relative to the "relharmvec" sideband frequency. In QPSP, relative means relative to the input port frequency. Possible values are absolute, relative or wrong-sweep-type.
Port parameters

11 ports=[...]
List of active ports. Ports are numbered in the order given. For purposes of noise figure computation, the input is considered port 1 and the output is port 2.

12 portharmsvec=[...]
List of the reference sidebands for the specified list of ports. Must have a one-to-one correspondence with the ports vector.

13 harmsvec=[...]
List of sidebands, in addition to ones associated with specific ports by portharmsvec, that are active. Call them secondary.

Output parameters

14 freqaxis
Specifies whether the results should be output versus the input port frequency, the output port frequency, or the absolute value of the input frequency. Default is in. Possible values are absin, in or out.

Noise parameters

15 donoise=yes
Perform noise analysis. If oprobe is specified as a valid port, this is set to yes, and a detailed noise output is generated. Possible values are no or yes.

Probe parameters

16 clockmaxharm=7
Maximum large tone harmonics included when computing noise either up-converted or down-converted to the output by that large signal.

Convergence parameters

17 tolerance=1e-9
Relative tolerance for linear solver.

18 gear_order=2
Gear order used for small-signal integration, 1 or 2.

19 solver=turbo
Solver type. Possible values are std or turbo.
Annotation parameters

20 annotate=sweep  Degree of annotation.
Possible values are no, title, sweep, status, or steps.

21 stats=no  Analysis statistics.
Possible values are no or yes.

22 title  Analysis title.

To specify the QPSP analysis the port and sideband combinations must be specified. You can select the ports of interest by setting the port parameter and the set of periodic small-signal output frequencies of interest by setting portharmsvec or harmsvec parameters. Sidebands are vectors in QPSP. Assume we have one large tone and one moderate tone in QPSS. A sideband K1 will be represented as [K1_1 K1_2]. Corresponding frequency is

\[ K1_1 \cdot \text{fund(large tone of QPSS)} + K1_2 \cdot \text{fund(moderate tone of QPSS)} = \sum_{j=1}^{L} \{ K_i_j \cdot \text{fund}_j(qpss) \} \]

We assume that there are L (1 large plus L-1 moderate) tones in QPSS analysis and a given set of n integer vectors representing the sidebands

\[ K1 = \{ K1_1, \ldots, K1_j, \ldots, K1_L \} , K2, \ldots, Kn. \]

If we specify the relative frequency then the scattering parameters at each port are computed at the frequencies

\[ f(\text{scattered}) = f(\text{rel}) + \sum_{j=1}^{L} \{ K_i_j \cdot \text{fund}_j(qpss) \}, \]

where f(\text{rel}) represents the relative frequency of a signal incident on a port, f(\text{scattered}) represents the frequency to which the relevant scattering parameter represents the conversion, and fund_j(qpss) represents the fundamental frequency used in the corresponding QPSS analysis.

In analysis of a down-converting mixer with a blocker and the signal in the upper sideband, we sweep the input frequency of the signal coming into RF port. The most relevant sideband for this input is Ki = {1, 0} - and for IF output Ki = {0, 0}. Hence we can associate K1={1, 0} with the RF port and K2={0, 0} with the IF port. S21 will represent the transmission of signal from the RF to IF, and S11 the reflection of signal back to the RF port. If the signal was in the lower sideband, then a choice of K1={-1, 0} would be more appropriate.

Either portharmsvec or harmsvec parameters can be used to specify the sidebands of the interest. If portharmsvec is given, the sidebands must be in one-to-one correspondence with the ports, with each sideband associated with a single port. If sidebands are specified in
the optional harmsvec parameter, then all possible frequency-translating scattering parameters associated with the specified sidebands on each port are computed.

With QPSP the frequency of the input and of the response are usually different (this is an important way in which QPSP differs from SP). Because the QPSP computation involves inputs and outputs at frequencies that are relative to multiple sidebands, the freqaxis and sweeptype parameters behave somewhat differently in QPSP than in QPAC and QPXF.

The sweeptype parameter controls the way the frequencies in the QPSP analysis are swept. Specifying a relative sweep indicates to sweep relative to the port sideband (not the QPSS fundamental) and an absolute sweep is a sweep of the absolute input source frequency. For example, with a QPSS fundamentals of 1000MHz (LO) and 966MHz (blocker in RF channel), portharmsvec could be set to [0 1 -1 1] to examine a downconverting mixer. Let's set sweeptype=relative and a sweep range of f(rel)=-10MHz<->10MHz. Then S21 would represent the strength of the signal transmitted from the input port in the range 956->976MHz to the output port to the frequencies 24<->44MHz. Using sweeptype=absolute and sweeping the frequency from 966<->976MHz would calculate the same quantities, since f(abs)=956<->976MHz, and f(rel) = f(abs) - ( K1_1 * fund_1(qpss) + K1_2 * fund_2(qpss) ) = -10MHz<->10MHz, because K1_1=0, K1_2=1 and fund_1(qpss) = 1000MHz, fund_2(qpss) = 966MHz.

The freqaxis parameter is used to specify whether the results should be output versus the scattered frequency at the input port (in), the scattered frequency at the output port (out), or the absolute value of the frequency swept at the input port (absin).

An increase in the number of requested ports will increase the simulation time substantially. The same will happen if we increase the number of sidebands to be included into the noise computations.

QPSP analysis also computes noise figures, equivalent noise sources, and noise parameters. The noise computation, which is skipped only when donoise=no, requires additional simulation time. If

\[ \text{No} = \text{total output noise at frequency } f \]
\[ \text{Ns} = \text{noise at the output due to the input probe (the source)} \]
\[ \text{Nsi} = \text{noise at the output due to the image harmonic at the source} \]
\[ \text{Nso} = \text{noise at the output due to harmonics other than input at the source} \]
\[ \text{NI} = \text{noise at the output due to the output probe (the load)} \]
\[ \text{IRN} = \text{input referred noise} \]
G = gain of the circuit
F = noise factor (single side band)
NF = noise figure (single side band)
Fdsb = double sideband noise factor
NFdsb = double sideband noise figure
Fieee = IEEE single sideband noise factor
NFieee = IEEE single sideband noise figure

then,

IRN = \sqrt{\frac{No^2}{G^2}}
F = \frac{No^2 - Nl^2}{Ns^2}
NF = 10 \log_{10}(F)
Fdsb = \frac{No^2 - Nl^2}{Ns^2 + Nsi^2}
NFdsb = 10 \log_{10}(Fdsb)
Fieee = \frac{No^2 - Nl^2 - Nso^2}{Ns^2}
NFieee = 10 \log_{10}(Fieee).

When the results are output, IRN is named in, G is named gain, F, NF, Fdsb, NFdsb, Fieee, and NFieee are named F, NF, Fdsb, NFdsb, Fieee, and NFieee respectively. Note that the gain computed by QPSP is the voltage gain from the actual circuit input to the circuit output, not the gain from the internal port voltage source to the output.

To ensure accurate noise calculations, the clockmaxharm parameters must be set to include the relevant noise folding effects. clockmaxharm is only relevant to the noise computation features of QPSP.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, dec) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. Alternatively, you may specify the particular values that the sweep parameter should take using the values parameter. If you give both a specific set of values
and a set specified using a sweep range, the two sets are merged and collated before being used. All frequencies are in Hertz.

Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate 20  gear_order 18  solver 19  sweeptype 10
center 3  harmsvec 13  span 4  title 22
clockmaxharm 16  lin 6  start 1  tolerance 17
dec 7  log 8  stats 21  values 9
donoise 15  portharmsvec 12  step 5
freqaxis 14  ports 11  stop 2

Quasi-Periodic Steady State Analysis (qpss)

Description

This analysis computes the quasi-periodic steady-state (QPSS) response of a circuit that operates on multiple time scales. A quasi-periodic signal has dynamics in multiple fundamental frequencies. Closely spaced or incommensurate fundamentals cannot be resolved by PSS efficiently. QPSS allows you to compute a circuit responses to several moderately large input signals in addition to a strongly nonlinear tone which represents a LO or clock signal. A typical example is the intermodulation distortion measurements of a mixer with two closely spaced moderate input signals. QPSS treats one particular input signal (usually the one that causes the most nonlinearity or the largest response) as the large signal, and the others as moderate signals.

An initial transient analysis is carried out by first suppressing all moderate input signals. Then, a number of (at least 2) stabilizing iterations with all signals activated is run. They are followed by the shooting Newton method. QPSS employs the Mixed Frequency Time (MFT) algorithm extended to multiple fundamental frequencies. For details of MFT algorithm, please see

Like PSS, QPSS uses the shooting Newton method as its backbone. However, instead of doing a single transient integration, each Newton iteration does a number of transient integrations of one large signal period.

Each of the integrations differs by a phase-shift in each moderate input signal. The number of integrations is determined by the numbers of harmonics of moderate fundamentals specified by the user. Given maxharms=[k1 k2 ... kn], the total number of integrations is \((2^\ast k_2+1)\ast(2^\ast k_3+1)\ast ... \ast(2^\ast k_n+1)\). As one consequence, the efficiency of the algorithm depends significantly on the number of harmonics required to model the responses of moderate fundamentals. As another consequence, the number of harmonics of the large fundamental does not significantly affect the efficiency of the shooting algorithm. The boundary conditions of a shooting interval are such that the time domain integrations are consistent with a frequency domain transformation with a shift of one large signal period.

QPSS inherits a majority of PSS parameters. A few new parameters are added. The most important ones are `funds` and `maxharms`. They replace PSS parameters, `fund` (or `period`) and `harm`, respectively. The `funds` parameter accepts a list of names of fundamentals that are present in the sources. These names are specified in the sources by parameter `fundname`. The first fundamental is considered as the large signal. A few heuristics can be used for picking the large fundamental.

1. Pick the one which is not a sinusoidal.
2. Pick the one which causes the most nonlinearity.
3. Pick the one which causes the largest response.

The `maxharms` parameter accepts a list of numbers of harmonics that are required to sufficiently model responses due to different fundamentals.

**Definition**

Name `qpss parameter=value` ...

**Parameters**

**QPSS fundamental parameters**

1. `funds=[...]` Array of fundamental frequency names for fundamentals to use in analysis.
maxharms= [...]  Array of number of harmonics of each fundamental to consider for each fundamental.

Simulation interval parameters

tstab=0.0 s  Extra stabilization time after the onset of periodicity for independent sources.

stabcycles=2  Stabilization cycles with both large and moderate sources enabled.

tstart=0.0 s  Initial transient analysis start time.

Time-step parameters

maxstep (s)  Maximum time step. Default derived from errpreset.

step=0.001 period s  Minimum time step that would be used solely to maintain the aesthetics of the results.

Initial-condition parameters

ic=all  What should be used to set initial condition. Possible values are dc, node, dev, or all.

skipdc=no  If yes, there will be no dc analysis for transient. Possible values are no, yes or sigrampup.

readic  File that contains initial condition.

Convergence parameters

readns  File that contains estimate of initial transient solution.

cmin=0 F  Minimum capacitance from each node to ground.
### Output parameters

13 **save**

Signals to output. Possible values are `all, lvl, allpub, lvlpub, selected, or none`.

14 **nestlvl**

Levels of subcircuits to output.

15 **oppoint=no**

Should operating point information be computed for initial timestep, and if so, where should it be sent. Possible values are `no, screen, logfile, or rawfile`.

16 **skipstart=starttime s**

The time to start skipping output data.

17 **skipstop=stoptime s**

The time to stop skipping output data.

18 **skipcount**

Save only one of every `skipcount` points.

19 **strobeperiod (s)**

The output strobe interval (in seconds of transient time).

20 **strobedelay=0 s**

The delay (phase shift) between the skipstart time and the first strobe point.

21 **compression=no**

Do data compression on output. See full description below. Possible values are `no or yes`.

22 **saveinit=no**

If set, the waveforms for the initial transient before steady state are saved. Possible values are `no or yes`.

### State-file parameters

23 **write**

File to which initial transient solution (before steady-state) is to be written.

24 **writefinal**

File to which final transient solution in steady-state is to be written.

25 **swapfile**

Temporary file that holds steady-state information. Tells Spectre to use a regular file rather than virtual memory to hold the
periodic operating point. Use this option if Spectre complains about not having enough memory to complete this analysis.

**Integration method parameters**

26 method Integration method. Default derived from errpreset. Possible values are euler, trap, traponly, gear2, or gear2only.

**Accuracy parameters**

27 errpreset Selects a reasonable collection of parameter settings. Possible values are liberal, moderate or conservative.

28 relref Reference used for the relative convergence criteria. Default derived from errpreset. Possible values are pointlocal, alllocal, sigglobal, or allglobal.

29 lteratio Ratio used to compute LTE tolerances from Newton tolerance. Default derived from errpreset.

30 steadyratio Ratio used to compute steady state tolerances from LTE tolerance. Default derived from errpreset.

31 maxperiods Maximum number of simulated periods to reach steady-state.

32 itres=1e-4 Relative tolerance for linear solver.

33 finitediff Options for finite difference method refinement after quasi-periodic shooting method. finitediff is changed from no to samegrid automatically when readqpss and writeqpss are used to re-use QPSS results. Possible values are no, yes or refine.

**Annotation parameters**

34 stats=no Analysis statistics. Possible values are no or yes.
**Spectre Circuit Simulator Reference**

**Analysis Statements**

35 annotate=sweep  
Degree of annotation.  
Possible values are no, title, sweep, status, or steps.

36 title  
Analysis title.

**Newton parameters**

37 maxiters=5  
Maximum number of iterations per time step.

38 restart=yes  
Restart the DC/PSS/QPSS solution from scratch if set to yes, if set to no, reuse the previous solution as initial guess.  
Possible values are no or yes.

**Circuit age**

39 circuitage (Years)  
Stress Time. Age of the circuit used to simulate hot-electron degradation of MOSFET and BSIM circuits.

40 writeqpss  
File to which final quasi-periodic steady-state solution is to be written. Small signal analyses such as qpac, qpxf and qpnoise can read in the steady-state solution from this file directly instead of running the qpss analysis again.

41 readqpss  
File from which final quasi-periodic steady-state solution is to be read. Small signal analyses such as qpac, qpxf and qpnoise can read in the steady-state solution from this file directly instead of running the qpss analysis again.

Most of QPSS analysis parameters are inherited from PSS analysis, and their meanings remain essentially unchanged. Two new important parameters are funds and maxharms. They replace and extend the role of fund and harms parameters of PSS analysis. One important difference is that funds accepts a list of fundamental names instead of actual frequencies. The frequencies associated with fundamentals are figured out automatically by the simulator. An important feature is that each input signal can be a composition of more than one sources. However, these sources must have the same fundamental name. For each fundamental name, its fundamental frequency is the greatest common factor of all frequencies associated with the name. Missing or not listing all fundamental names using the parameter funds will result in an amputation of the current simulation. However if maxharms is not given, a warning message will be issued, and the number of harmonics is defaulted to 1 for each of the fundamentals.
For QPSS analyses, the role of some PSS parameters is extended. The parameter \texttt{maxperiods} that controls the maximum number of shooting iterations for PSS analysis also controls the number of the maximum number of shooting iterations for QPSS analysis. Its default value is set to 50.

The \texttt{tstab} parameter controls both the length of the initial transient integration with only the clock tone activated and the number of stable iterations with moderate tones activated. The stable iterations are run before Newton iterations.

The \texttt{errpreset} parameter lets you adjust several simulator parameters to fit your needs. In most cases, \texttt{errpreset} should be the only parameter you need to adjust. If you want a fast simulation with reasonable accuracy, you might set \texttt{errpreset} to \texttt{liberal}. If have some concern for accuracy, you might set \texttt{errpreset} to \texttt{moderate}. If accuracy is your main interest, you might set \texttt{errpreset} to \texttt{conservative}.

If users do not specify \texttt{steadyratio}, it is always 1.0, and it is not affected by \texttt{errpreset}. The following table shows the effect of \texttt{errpreset} on other parameters.

### Parameter defaults as a function of \texttt{errpreset}

<table>
<thead>
<tr>
<th>\texttt{errpreset}</th>
<th>\texttt{reltol}</th>
<th>\texttt{relref}</th>
<th>\texttt{method}</th>
<th>\texttt{Iteratio}</th>
<th>\texttt{maxstep}</th>
</tr>
</thead>
<tbody>
<tr>
<td>liberal</td>
<td>1e-3</td>
<td>sigglobal</td>
<td>gear2only</td>
<td>3.5</td>
<td>clock period/80</td>
</tr>
<tr>
<td>moderate</td>
<td>1e-4</td>
<td>sigglobal</td>
<td>gear2only</td>
<td>3.5</td>
<td>clock period/100</td>
</tr>
<tr>
<td>conservative</td>
<td>1e-5</td>
<td>sigglobal</td>
<td>gear2only</td>
<td>*</td>
<td>clock period/200</td>
</tr>
</tbody>
</table>

* : \texttt{Iteratio}=10.0 for conservative \texttt{errpreset} by default. Only if user specified \texttt{reltol} <= 1e-4*10.0/3.5, it will change to \texttt{Iteratio}=3.5.

The new \texttt{errpreset} settings include a new default \texttt{reltol} which is actually an enforced upper limit for appropriate setting. An increase of \texttt{reltol} above default will be ignored by the simulator. User can decrease this value in the options statement. The only way to increase \texttt{reltol} is to relax \texttt{errpreset}. Spectre sets the value of \texttt{maxstep} so that it is no larger than the value given in the table. Except for \texttt{reltol} and \texttt{maxstep}, \texttt{errpreset} does not change the value of any parameters you have explicitly set. The actual values used for the QPSS analysis are given in the log file.

If \texttt{errpreset} is not specified in the netlist, \texttt{liberal} settings will be used.

The default value for \texttt{compression} is no. The output file stores data for every signal at every time point for which Spectre calculates a solution. Spectre saves the x axis data only once,
since every signal has the same x value. If \texttt{compression=yes}, Spectre writes data to the output file only if the signal value changes by at least 2*the convergence criteria. In order to save data for each signal independently, x axis information corresponding to each signal must be saved. If the signals stay at constant values for large periods of the simulation time, setting \texttt{compression=yes} results in a smaller output data file. If the signals in your circuit move around a lot, setting \texttt{compression=yes} results in a larger output data file.

**Parameter Index**

In the following index, the number following each parameter name indicates where to find the description of that parameter.

| annotate 35 | maxiters 37 | save 13 | strobeperiod 19 |
| circuitage 39 | maxperiods 31 | saveinit 22 | swapfile 25 |
| cmin 12 | maxstep 6 | skipcount 18 | title 36 |
| compression 21 | method 26 | skipdc 9 | tstab 3 |
| errpreset 27 | nestlvl 14 | skipstart 16 | tstart 5 |
| finitediff 33 | oppoint 15 | skipstop 17 | write 23 |
| funds 1 | readic 10 | stabcycles 4 | writefinal 24 |
| ic 8 | readns 11 | stats 34 | writeqpss 40 |
| itres 32 | readqpss 41 | steadyratio 30 |
| lteratio 29 | relref 28 | step 7 |
| maxharms 2 | restart 38 | strobedelay 20 |
Quasi-Periodic Transfer Function Analysis (qpxf)

Description

A conventional transfer function analysis computes the transfer function from every source in the circuit to a single output. It differs from a conventional AC analysis in that the AC analysis computes the response from a single stimulus to every node in the circuit. The difference between QPAC and QPXF analysis are similar. The Quasi Periodic Transfer Function or QPXF analysis computes the transfer functions from any source at any frequency to a single output at a single frequency. Thus, like QPAC analysis, QPXF analysis includes frequency conversion effects.

The QPXF analysis directly computes such useful quantities as conversion efficiency (transfer function from input to output at desired frequency), image and sideband rejection (input to output at undesired frequency), and LO feed-through and power supply rejection (undesired input to output at all frequencies).

As with a QPAC, QPSP, and QPNOISE analyses, a QPXF analysis must follow a QPSS analysis.

Unlike other analyses in Spectre, this analysis can only sweep frequency.

Definition

Name  [p]  [n] qpxf parameter=value ...

The optional terminals (p and n) specify the output of the circuit. If you do not give the terminals, then you must specify the output with a probe component.

Parameters

Sweep interval parameters

1  start=0  Start sweep limit.
2  stop  Stop sweep limit.
3  center  Center of sweep.
4  span=0  Sweep limit span.
### Analysis Statements

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>step</td>
</tr>
<tr>
<td>6</td>
<td>lin=50</td>
</tr>
<tr>
<td>7</td>
<td>dec</td>
</tr>
<tr>
<td>8</td>
<td>log=50</td>
</tr>
<tr>
<td>9</td>
<td>values= [...]</td>
</tr>
<tr>
<td>10</td>
<td>sweeptype</td>
</tr>
<tr>
<td>11</td>
<td>relharmvec= [...]</td>
</tr>
</tbody>
</table>

**Probe parameters**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>probe</td>
</tr>
</tbody>
</table>

**Output parameters**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>stimuli=sources</td>
</tr>
<tr>
<td>14</td>
<td>sidevec= [...]</td>
</tr>
<tr>
<td>15</td>
<td>clockmaxharm=0</td>
</tr>
<tr>
<td>16</td>
<td>freqaxis</td>
</tr>
</tbody>
</table>
The variable of interest at the output can be voltage or current, and its frequency is not constrained by the period of the large periodic solution. While sweeping the selected output frequency, you can select the periodic small-signal input frequencies of interest by setting either the \texttt{clockmaxharm} or the \texttt{sidevec} parameters. Sidebands are vectors in QPXF. Assume we have one large tone and one moderate tone in QPSS. A sideband $K_1$ will be represented as $[K_{1_1} \ K_{1_2}]$. Corresponding frequency is

$$K_{1_1} \cdot \text{fund(large tone of QPSS)} + K_{1_2} \cdot \text{fund(moderate tone of QPSS)}$$

We assume that there are $L$ (1 large plus $L-1$ moderate) tones in QPSS analysis and a given set of $n$ integer vectors representing the sidebands

$$K_1 = \{ K_{1_1}, \ldots, K_{1_j}, \ldots, K_{1_L}, K_2, \ldots, K_n \}$$

The input signal frequency at each sideband is computed as

$$f(\text{in}) = f(\text{out}) + \sum_{j=1}^{L} (K_{i_j} \cdot \text{fund}_j(\text{qpss})),$$
where $f_{\text{out}}$ represent the (possibly swept) output signal frequency, and $\text{fund}_j(pss)$ represents the fundamental frequency used in the corresponding QPSS analysis. Thus, when analyzing a down-converting mixer, and sweeping the IF output frequency, $K_i = \{1, 0\}$ for the RF input represents the first upper-sideband, while $K_i = \{-1, 0\}$ for the RF input represents the first lower-sideband.

User would enter `sidevec` as a sequence of integer numbers, separated by spaces. The set of vectors $\{1 1 0\} \{1 -1 0\} \{1 1 1\}$ becomes $\text{sidevec} = [1 1 0 1 -1 0 1 1 1]$. For $\text{clockmaxharm}$, only the large tone - first fundamental will be affected by this entry, all the rest - moderate tones - will be limited by $\text{maxharms}$, specified for a QPSS analysis. Given $\text{maxharms}=[k_1 \text{max} k_2 \text{max} \ldots k_n \text{max}]$ in QPSS and $\text{clockmaxharm}=K_{\text{max}}$ all $(2*K_{\text{max}} + 1)*(2*k_{2\text{max}}+1)*(2*k_{3\text{max}}+1)*...*(2*k_{n\text{max}}+1)$ sidebands are generated.

The number of requested sidebands changes substantially the simulation time.

With QPXF the frequency of the stimulus and of the response are usually different (this is an important way in which QPXF differs from XF). The `freqaxis` parameter is used to specify whether the results should be output versus the input frequency ($\text{in}$), the output frequency ($\text{out}$), or the absolute value of the input frequency ($\text{absin}$).

You can specify the output with a pair of nodes or a probe component. Any component with two or more terminals can be a voltage probe. When there are more than two terminals, they are grouped in pairs; and you use the `portv` parameter to select the appropriate pair of terminals. Alternatively, you can simply specify a voltage to be the output by giving a pair of nodes on the QPXF analysis statement.

Any component that naturally computes current as an internal variable can be a current probe. If the probe component computes more than one current, you use the `porti` parameter to select the appropriate current. It is an error to specify both `portv` and `porti`. If neither is specified, the probe component provides a reasonable default.

The `stimuli` parameter specifies what is used for the inputs for the transfer functions. There are two choices. `stimuli=sources` indicates that the sources present in the circuit should be used. The `xfmag` parameters provided by the sources may be used to adjust the computed gain to compensate for gains or losses in a test fixture. One can limit the number of sources in hierarchical netlists by using the `save` and `nestlvl` parameters. `stimuli=nodes_and_terminals` indicates that all possible transfer functions should be computed.

This is useful when it is not known in advance which transfer functions are interesting. Transfer functions for nodes are computed assuming that a unit magnitude flow (current) source is connected from the node to ground. Transfer functions for terminals are computed assuming that a unit magnitude value (voltage) source is connected in series with the terminal. By default, the transfer functions from a small set of terminals are computed.
transfer functions from specific terminals are desired, specify the terminals in the save statement. You must use the :probe modifier (ex. Rout:1:probe) or specify useprobes=yes on the options statement. If transfer functions from all terminals are desired, specify currents=all and useprobes=yes on the options statement.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, dec) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. Alternatively, you may specify the particular values that the sweep parameter should take using the values parameter. If you give both a specific set of values and a set specified using a sweep range, the two sets are merged and collated before being used. All frequencies are in Hertz.

Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotate</td>
<td>22</td>
</tr>
<tr>
<td>lin</td>
<td>6</td>
</tr>
<tr>
<td>sidevec</td>
<td>14</td>
</tr>
<tr>
<td>stimuli</td>
<td>13</td>
</tr>
<tr>
<td>center</td>
<td>3</td>
</tr>
<tr>
<td>log</td>
<td>8</td>
</tr>
<tr>
<td>solver</td>
<td>21</td>
</tr>
<tr>
<td>stop</td>
<td>2</td>
</tr>
<tr>
<td>clockmaxharm</td>
<td>15</td>
</tr>
<tr>
<td>nestlvl</td>
<td>18</td>
</tr>
<tr>
<td>span</td>
<td>4</td>
</tr>
<tr>
<td>sweeptype</td>
<td>10</td>
</tr>
<tr>
<td>dec</td>
<td>7</td>
</tr>
<tr>
<td>probe</td>
<td>12</td>
</tr>
<tr>
<td>start</td>
<td>1</td>
</tr>
<tr>
<td>title</td>
<td>24</td>
</tr>
<tr>
<td>freqaxis</td>
<td>16</td>
</tr>
<tr>
<td>relharmvec</td>
<td>11</td>
</tr>
<tr>
<td>stats</td>
<td>23</td>
</tr>
<tr>
<td>tolerance</td>
<td>19</td>
</tr>
<tr>
<td>gear_order</td>
<td>20</td>
</tr>
<tr>
<td>save</td>
<td>17</td>
</tr>
<tr>
<td>step</td>
<td>5</td>
</tr>
<tr>
<td>values</td>
<td>9</td>
</tr>
</tbody>
</table>

Deferred Set Options (set)

Description

The deferred set options statement sets or changes various program control options. You can set the options in any order and, once set, the options retain their value until reset. The set statement is queued with all analyses and is executed sequentially (The changes made to
these options are deferred until the statement setting them is encountered). To set temp, 
tnom, scalem, or scale, use the alter statement. For further options, see individual 
analyses.

**Definition**
Name set parameter=value ...

**Parameters**

*Tolerance parameters*

1  reltol=0.001  Relative convergence criterion.
2  residualtol=1.0  Tolerance ratio for residual (multiplies reltol).
3  vabstol=1e-06 V  Voltage absolute tolerance convergence criterion.
4  iabstol=1e-12 A  Current absolute tolerance convergence criterion.

*Temperature parameters*

5  tempeffects=all  Temperature effect selector. If tempeffect = vt, only thermal 
voltage varies with temperature; if tempeffect = tc, parameters 
that start with tc are active and thermal voltage is dependent on 
temperature; and if tempeffect = all, all built-in temperature 
models are enabled. 
Possible values are vt, tc or all.

*Convergence parameters*

6  homotopy=all  Method used when no convergence on initial attempt of DC 
analysis. 
Possible values are none, gmin, source, dptran, ptran, 
arclength, or all.

7  limit=dev  Limiting algorithms to aid DC convergence. 
Possible values are delta, log or dev.
Component parameters

8 compatible=spectre
Encourage device equations to be compatible with a foreign simulator. This option does not affect input syntax.
Possible values are spectre, spice2, spice3, cdssspice, hspice, or spiceplus.

9 approx=no
Use approximate models. Difference between approximate and exact models is generally very small.
Possible values are no or yes.

Error-checking parameters

10 diagnose=no
Print additional information that might help diagnose accuracy and convergence problems.
Possible values are no or yes.

11 opptcheck=yes
Check operating point parameters against soft limits.
Possible values are no or yes.

Resistance parameters

12 gmin=1e-12 S
Minimum conductance across each nonlinear device.

13 gmin_check=max_v_only
Specifies that effect of gmin should be reported if significant.
Possible values are no, max_v_only, max_only, or all.

14 rforce=1 Ω
Resistance used when forcing nodesets and node-based initial conditions.

Quantity parameters

15 quantities=no
Print quantities.
Possible values are no, min or full.
**Annotation parameters**

16 narrate=yes Narrate the simulation. Possible values are no or yes.

17 debug=no Give debugging messages. Possible values are no or yes.

18 info=yes Give informational messages. Possible values are no or yes.

19 note=yes Give notice messages. Possible values are no or yes.

20 maxnotes=5 Maximum number of times any notice will be issued per analysis.

21 warn=yes Give warning messages. Possible values are no or yes.

22 maxwarns=5 Maximum number of times any warning message will be issued per analysis.

23 maxwarnstologfile=5 Maximum number of times any warning message will be printed to the log file per analysis.

24 maxnotestologfile=5 Maximum number of times any notice message will be printed to the log file per analysis.

25 error=yes Give error messages. Possible values are no or yes.

26 digits=5 Number of digits used when printing numbers.

27 notation=eng When printing real numbers to the screen, what notation should be used. Possible values are eng, sci or float.

28 annotate=no Degree of annotation. Possible values are no or title.
Matrix parameters

29 pivotdc=no  Use numeric pivoting on every iteration of DC analysis. Possible values are no or yes.

30 pivrel=0.001 Relative pivot threshold.

31 pivabs=0 Absolute pivot threshold.

Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate 28  gmin_check 13  maxwarnstologfile quantities 15
            23
approx 9  homotopy 6  narrate 16  reltol 1
compatible 8  iabstol 4  notation 27  residualtol 2
debug 17  info 18  note 19  rforce 14
diagnose 10  limit 7  opptcheck 11  tempeffects 5
digits 26  maxnotes 20  pivabs 31  vabstol 3
error 25  maxnotestologfile pivotdc 29  warn 21
            24

Shell Command (shell)

Description

The shell analysis passes a command to the operating system command interpreter given in the SHELL environment variable. The command behaves as if it were typed into the command interpreter, except that any %X codes in the command are expanded first.
The default action of the shell analysis is to terminate the simulation.

**Definition**

Name shell parameter=value ... 

**Parameters**

1. cmd="kill %P"  
   Shell command.

2. iferror=quit  
   What to do if command returns nonzero error status.  
   Possible values are continue or quit.

3. annotate  
   Degree of annotation.  
   Possible values are no, title or yes.

**S-Parameter Analysis (sp)**

**Description**

The S-parameter analysis linearizes the circuit about the DC operating point and computes S-parameters of the circuit taken as an N-port. The port statements define the ports of the circuit. Each active port is turned on sequentially, and a linear small-signal analysis is performed. Spectre converts the response of the circuit at each active port into S-parameters and outputs these parameters. There must be at least one active port statement in the circuit.

If a filename is specified using the file parameter, the S-parameter analysis generates an ASCII file containing the S-parameters of the circuit that can later be read-in by the nport component. The generated file can be in either Spectre's native format or in Touchstone format.

Spectre can perform the analysis while sweeping a parameter. The parameter can be frequency, temperature, component instance parameter, component model parameter, or netlist parameter. If changing a parameter affects the DC operating point, the operating point is recomputed on each step. You can sweep the circuit temperature by giving the parameter name as temp with no dev or mod parameter. You can sweep a netlist parameter by giving the parameter name with no dev, or mod parameter. After the analysis has completed, the modified parameter returns to its original value.
Definition
Name sp parameter=value ...

Parameters
1  prevoppoint=no  Use operating point computed on the previous analysis. Possible values are no or yes.

Sweep interval parameters
2  start=0  Start sweep limit.
3  stop  Stop sweep limit.
4  center  Center of sweep.
5  span=0  Sweep limit span.
6  step  Step size, linear sweep.
7  lin=50  Number of steps, linear sweep.
8  dec  Points per decade.
9  log=50  Number of steps, log sweep.
10  values=[... ]  Array of sweep values.

Sweep variable parameters
11  dev  Device instance whose parameter value is to be swept.
12  mod  Model whose parameter value is to be swept.
13  param  Name of parameter to sweep.
14  freq (Hz)  Frequency when parameter other than frequency is being swept.
Port parameters

15 \texttt{ports= [...]}
List of active ports. Ports are numbered in the order given.

State-file parameters

16 \texttt{readns}
File that contains estimate of DC solution (nodeset).

Output parameters

17 \texttt{file}
S-parameters output file name.

18 \texttt{datafmt=spectre}
Data format of the S-parameter output file.
Possible values are \texttt{spectre} or \texttt{touchstone}.

19 \texttt{datatype=realimag}
Data type of the S-parameter output file.
Possible values are \texttt{realimag}, \texttt{magphase} or \texttt{dbphase}.

20 \texttt{noisedata=no}
Should noise data be saved to the S-parameter output file, and if so, in what format.
Possible values are \texttt{no}, \texttt{twoport} or \texttt{cy}.

21 \texttt{oppoint=no}
Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if operating point is computed in the previous analysis and is unchanged.
Possible values are \texttt{no}, \texttt{screen}, \texttt{logfile}, or \texttt{rawfile}.

Noise parameters

22 \texttt{donoise=no}
Perform noise analysis. If \texttt{ozone} is specified as a valid port, this is set to yes, and a detailed noise output is generated.
Possible values are \texttt{no} or \texttt{yes}.

23 \texttt{ozone}
Compute total noise at the output defined by this component.

24 \texttt{iprobe}
Input probe. Refer the output noise to this component.
Convergence parameters

25 restart=yes Restart the DC solution from scratch if any condition has changed. If not, use the previous solution as initial guess. Possible values are no or yes.

Annotation parameters

26 annotate=sweep Degree of annotation. Possible values are no, title, sweep, status, or steps.

27 stats=no Analysis statistics. Possible values are no or yes.

28 title Analysis title.

If the list of active ports is specified with the ports parameter, then the ports are numbered sequentially from one in the order given. Otherwise, all ports present in the circuit are active, and the port numbers used are those that were assigned on the port statements. If donoise=yes is specified, then the noise correlation matrix is computed. If in addition, the output is specified using oprobe, the amount that each noise source contributes to the output is computed. Finally, if an input is also specified (using iprobe), the two-port noise parameters are computed (F, Fmin, NF, NFmin, Gopt, Bopt, and Rn).

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, dec) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. All frequencies are in Hertz.

The small-signal analysis begins by linearizing the circuit about an operating-point. By default this analysis computes the operating-point if it is not known, or recomputes it if any significant component or circuit parameter has changed. However, if a previous analysis computed an operating point, you can set prevoppoint=yes to avoid recomputing it. For example, if you use this option when the previous analysis was a transient analysis, the operating point is the state of the circuit on the final time point.
Parameter Index

In the following index, the number following each parameter name indicates where to find the
description of that parameter.

annotate 26    file 17    oppoint 21    span 5
center 4       freq 14     oprobe 23    start 2
datafmt 18     iprobe 24   param 13     stats 27
datatype 19     lin 7      ports 15     step 6
dec 8          log 9      prevoppoint 1 stop 3
dev 11         mod 12     readns 16    title 28
donoise 22     noisedata 20 restart 25    values 10

Stability Analysis (stb)

Description

The STB analysis linearizes the circuit about the DC operating point and computes the loop
gain, gain and phase margins (if the sweep variable is frequency), for a feedback loop or a
gain device.

Spectre can perform the analysis while sweeping a parameter. The parameter can be
frequency, temperature, component instance parameter, component model parameter, or
netlist parameter. If changing a parameter affects the DC operating point, the operating point
is recomputed on each step. You can sweep the circuit temperature by giving the parameter
name as temp with no dev or mod parameter. You can sweep a netlist parameter by giving
the parameter name with no dev, or mod parameter. After the analysis has completed, the
modified parameter returns to its original value.

Definition

Name stb parameter=value ...
Parameters

1 prevoppoint=no Use operating point computed on the previous analysis. Possible values are no or yes.

Sweep interval parameters

2 start=0 Start sweep limit.
3 stop Stop sweep limit.
4 center Center of sweep.
5 span=0 Sweep limit span.
6 step Step size, linear sweep.
7 lin=50 Number of steps, linear sweep.
8 dec Points per decade.
9 log=50 Number of steps, log sweep.
10 values=[...] Array of sweep values.

Sweep variable parameters

11 dev Device instance whose parameter value is to be swept.
12 mod Model whose parameter value is to be swept.
13 param Name of parameter to sweep.
14 freq (Hz) Frequency when parameter other than frequency is being swept.

Probe parameters

15 probe Probe instance around which the loop gain is calculated.
State-file parameters

16 readns File that contains estimate of DC solution (nodeset).

Output parameters

17 save Signals to output. Possible values are all, lvl, allpub, lvlpub, selected, or none.

18 nestlvl Levels of subcircuits to output.

19 oppoint=no Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if operating point is computed in the previous analysis and is unchanged. Possible values are no, screen, logfile, or rawfile.

Convergence parameters

20 restart=yes Restart the DC solution from scratch if any condition has changed. If not, use the previous solution as initial guess. Possible values are no or yes.

Annotation parameters

21 annotate=sweep Degree of annotation. Possible values are no, title, sweep, status, or steps.

22 stats=no Analysis statistics. Possible values are no or yes.

23 title Analysis title.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, dec) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. All frequencies are in Hertz.
The small-signal analysis begins by linearizing the circuit about an operating-point. By default this analysis computes the operating-point if it is not known, or recomputes it if any significant component or circuit parameter has changed. However, if a previous analysis computed an operating point, you can set `prevoppoint=yes` to avoid recomputing it. For example, if you use this option when the previous analysis was a transient analysis, the operating point is the state of the circuit on the final time point.

**Understanding Loop based and Device Based Algorithms**

Two algorithms--the loop based and the device based, are available for small-signal stability analysis. Both algorithms are based on the calculation of Bode's return ratio. Loop gain waveform, gain margin, and phase margin are the analysis output.

The `probe` parameter must be specified to perform stability analysis. When it points to a current probe or voltage source instance, the loop based algorithm will be invoked; when it points to a supported active device instance, the device based algorithm will be invoked.

**Loop Based Algorithm**

The loop based algorithm calculates the true loop gain that consists of normal loop gain and reverse loop gain. The loop based algorithm requires the `probe` being placed on the feedback loop to identify and characterize the particular loop of interest. The introduction of the `probe` component should not change any of the circuit characteristics.

The loop based algorithm provides accurate stability information for single loop circuits, and multiloop circuits in which a `probe` component can be placed on a critical wire to break all loops. For a general multiloop circuit, such a critical wire may not be available. The loop based algorithm can only be performed on individual feedback loops to ensure they are stable. Although the stability of all feedback loops is only a necessary condition for the whole circuit to be stable, the multiloop circuit tends to be stable if all individual loops are associated with reasonable stability margins.

**Device Based Algorithm**

The device based algorithm calculates the loop gain around a particular active device. This algorithm is often applied to assess the stability of circuit design in which local feedback loops cannot be neglected; the loop based algorithm cannot be performed for these applications since the local feedback loops are inside the devices, they are not accessible from the schematic level or netlist level to insert the `probe` component.
With the `probe` parameter points to a particular active device, the dominant controlled source in the device will be nulled during the analysis. The dominant controlled source is defined as by nulling this source renders the active device to be passive. The device based algorithm produces accurate stability information for a circuit in which a critical active device can be identified such that nulling the dominant gain source of this device renders the whole network to be passive.

**Stability Analysis of Differential Feedback Circuits**

A balanced fully differential feedback circuit is illustrated below:

```
|------[ ZF ]------|
|                |
|                |
|                |
|                |
X1
|------------------|  |
|------------------|  |
|                |
|                |
|                |
|                |
|------[ ZS ]------|--|in+       out-|--[ ZL ]-- ... |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
|                |
```

The feedback loops are broken at X1 and X2, with x1in and x2in being the input side nodes, x1out and x2out being the output side nodes. The following subcircuit connects these four nodes togethers:
subckt diffprobe x1in x2in x1out x2out

ibranch inout x1out iprobe

vinj inout x1in iprobe

evinj x2in x2out x1in x1out vcvs gain=0

fiinj 0 x2out pcccs probes=[ibranch vinj] coeffs=[0 1 1] gain=0

ends diffprobe

Let **diffprobe_inst** be the instance of subcircuit **diffprobe**, the following analysis measures the differential-mode loop gain:

DMalterv alter dev=diffprobe_inst.evinj param=gain value=-1

DMalteri alter dev=diffprobe_inst.fiinj param=gain value=-1

DMloopgain stb probe=diffprobe_inst.vinj

and the following analysis measures the common-mode loop gain:

CMalterv alter dev=diffprobe_inst.evinj param=gain value=1

CMalteri alter dev=diffprobe_inst.fiinj param=gain value=1

CMloopgain stb probe=diffprobe_inst.vinj

**Parameter Index**

In the following index, the number following each parameter name indicates where to find the description of that parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotate</td>
<td>21</td>
</tr>
<tr>
<td>log</td>
<td>9</td>
</tr>
<tr>
<td>probe</td>
<td>15</td>
</tr>
<tr>
<td>stats</td>
<td>22</td>
</tr>
<tr>
<td>center</td>
<td>4</td>
</tr>
<tr>
<td>mod</td>
<td>12</td>
</tr>
<tr>
<td>readns</td>
<td>16</td>
</tr>
<tr>
<td>step</td>
<td>6</td>
</tr>
<tr>
<td>dec</td>
<td>8</td>
</tr>
<tr>
<td>nestlvl</td>
<td>18</td>
</tr>
<tr>
<td>restart</td>
<td>20</td>
</tr>
<tr>
<td>stop</td>
<td>3</td>
</tr>
<tr>
<td>dev</td>
<td>11</td>
</tr>
<tr>
<td>oppoint</td>
<td>19</td>
</tr>
<tr>
<td>save</td>
<td>17</td>
</tr>
<tr>
<td>title</td>
<td>23</td>
</tr>
<tr>
<td>freq</td>
<td>14</td>
</tr>
<tr>
<td>param</td>
<td>13</td>
</tr>
<tr>
<td>span</td>
<td>5</td>
</tr>
<tr>
<td>values</td>
<td>10</td>
</tr>
</tbody>
</table>
Sweep Analysis (sweep)

Description

The sweep analysis sweeps a parameter executing the list of analyses (or multiple analyses) for each value of the parameter. The swept parameter can be circuit temperature, a device instance parameter, a device model parameter, a netlist parameter, or a subcircuit parameter for a particular subcircuit instance.

A set of parameters can be swept simultaneously, using the paramset parameter. The other sweep interval or variable parameters cannot be specified with the paramset parameter. Do spectre -h paramset for information on defining a paramset.

Within a sweep statement, you can specify analyses statements. These statements should be bound within braces. The opening brace is required at the end of the line defining the sweep. Sweep statements can be nested.

You can sweep the circuit temperature by giving the parameter name as param=temp with no dev, mod, or sub parameter. You can sweep a top-level netlist parameter by giving the parameter name with no dev, mod, or sub parameter. You can sweep a subcircuit parameter for a particular subcircuit instance by specifying the subcircuit instance name with the sub parameter and the subcircuit parameter name with the param parameter. The same can be done using dev for the device instance name or mod for the device model name.

After the analysis has completed, the modified parameter returns to its original value.

Definition

Name sweep parameter=value ...

Parameters

Sweep interval parameters

1  start=0       Start sweep limit.
Spectre Circuit Simulator Reference
Analysis Statements

2 stop      Stop sweep limit.
3 center    Center of sweep.
4 span=0    Sweep limit span.
5 step      Step size, linear sweep.
6 lin=50    Number of steps, linear sweep.
7 dec       Points per decade.
8 log=50    Number of steps, log sweep.
9 values=[...] Array of sweep values.

Sweep variable parameters
10 dev       Device instance whose parameter value is to be swept.
11 sub       Subcircuit instance whose parameter value is to be swept.
12 mod       Model whose parameter value is to be swept.
13 param     Name of parameter to sweep.
14 paramset  Name of parameter set to sweep.

Annotation parameters
15 annotate=sweep Degree of annotation.
                   Possible values are no, title or sweep.
16 title      Analysis title.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, or dec) and determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of the stop-to-start values is less than 10 and logarithmic when this ratio is 10 or greater.
Example:

swp sweep param=temp values=[-50 0 50 100 125] {
    oppoint dc oppoint=logfile
}

Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate 15  lin 6  paramset 14  stop 2
center 3  log 8  span 4  sub 11
dec 7  mod 12  start 1  title 16
dev 10  param 13  step 5  values 9

Time-Domain Reflectometer Analysis (tdr)

Description

The time-domain reflectometer analysis linearizes the circuit about the DC operating point and computes the reflection coefficients versus time, looking from the active ports into the circuit.

Definition

Name tdr parameter=value ...

Parameters

1  stop  Stop time.
2  settling=stop  Time required for circuit to settle.
Such a small-signal analysis begins by linearizing the circuit about an operating point. By default, this analysis computes the operating point, if it is not yet known, or recomputes it, if any significant component or circuit parameter has changed. However, if a previous analysis computed an operating point, you can set `prevoppoint=yes` to avoid recomputing it. For example, if you use this command when the previous analysis was a transient analysis, the operating point is the state of the circuit on the final time point.
Parameter Index

In the following index, the number following each parameter name indicates where to find the description of that parameter.

- annotate 10
- prevoppoint 13
- smoothing 4
- vel 5
- oppoint 12
- readns 8
- start 3
- points 6
- restart 9
- stop 1
- ports 7
- settling 2
- title 11

Transient Analysis (tran)

Description

This analysis computes the transient response of a circuit over the interval from start to stop. The initial condition is taken to be the DC steady-state solution if not otherwise given.

Definition

Name tran parameter=value ...

Parameters

Simulation interval parameters

1. stop (s) Stop time.
2. start=0 s Start time.
3. outputstart=start s Output is saved only after this time is reached.
4. autostop=no If yes, the analysis is terminated when all event-type measurement expressions have been evaluated. Event-type expressions use thresholding, event or delay type functions. Possible values are no or yes.
**Time-step parameters**

5 maxstep (s) Maximum time step. Default derived from errpreset.

6 step=0.001 (stop-start) s Minimum time step used by the simulator solely to maintain the aesthetics of the computed waveforms.

**Initial-condition parameters**

7 ic=all What should be used to set initial condition. Possible values are dc, node, dev, or all.

8 skipdc=no If yes, there will be no dc analysis for transient. Possible values are no, yes, waveless, rampup, autodc, or sigrampup.

9 readic File that contains initial condition.

**Convergence parameters**

10 readns File that contains estimate of initial transient solution.

11 cmin=0 F Minimum capacitance from each node to ground.

**State-file parameters**

12 write File to which initial transient solution is to be written.

13 writefinal File to which final transient solution is to be written.

14 ckptperiod Checkpoint the analysis periodically using the specified period.

**Integration method parameters**

15 method Integration method. Default derived from errpreset. Possible values are euler, trap, traponly, gear2, gear2only, or trapgear2.
Accuracy parameters

16 errpreset
Selects a reasonable collection of parameter settings.
Possible values are liberal, moderate or conservative.

17 relref
Reference used for the relative convergence criteria. Default derived from errpreset.
Possible values are pointlocal, alllocal, sigglobal, or allglobal.

18 lteratio
Ratio used to compute LTE tolerances from Newton tolerance. Default derived from errpreset.

19 fastbreak=no
If yes, VHDLAMS Break statement is handled using faster Verilog method.
Possible values are no or yes.

20 fastcross=discrete
Using limited threshold reject method for fast cross detection.
Possible values are no, yes or discrete.

Annotation parameters

21 stats=no
Analysis statistics.
Possible values are no or yes.

22 annotate=sweep
Degree of annotation.
Possible values are no, title, sweep, status, or steps.

23 title
Analysis title.

Output parameters

24 save
Signals to output.
Possible values are all, lvl, alllpub, lvlpub, selected, or none.

25 nestlvl
Levels of subcircuits to output.

26 oppoint=no
Should operating point information be computed for initial timestep, and if so, where should it be sent.
Possible values are no, screen, logfile, or rawfile.
Spectre Circuit Simulator Reference
Analysis Statements

27 skipstart=starttime
   The time to start skipping output data.

28 skipstop=stoptime
   The time to stop skipping output data.

29 skipcount
   Save only one of every skipcount points.

30 strobeperiod (s)
   The output strobe interval (in seconds of transient time).

31 strobedelay=0 s
   The delay (phase shift) between the skipstart time and the first strobe point.

32 compression=no
   Do data compression on output. See full description below. Possible values are no or yes.

33 flushpoints
   Flush outputs after number of calculated points.

34 flushtime (s)
   Flush outputs after real time has elapsed.

35 flushofftime (s)
   Time to stop flushing outputs.

36 infoname
   Name of info analysis to be performed at each time point in the infotimes array.

37 infotimes= [...] s
   Times when info analysis specified by infoname is performed.

Newton parameters

38 maxiters=5
   Maximum number of iterations per time step.

39 restart=yes
   Restart the DC solution from scratch if any condition has changed. If not, use the previous solution as initial guess. Possible values are no or yes.

Circuit age

40 circuitage (Years)
   Stress Time. Age of the circuit used to simulate hot-electron degradation of MOSFET and BSIM circuits.

You may specify the initial condition for the transient analysis by using the ic statement or the ic parameter on the capacitors and inductors. If you do not specify the initial condition, the DC solution is used as the initial condition. The ic parameter on the transient analysis
controls the interaction of various methods of setting the initial conditions. The effects of
individual settings are:

- **ic=dc**: Any initial condition specifiers are ignored, and the DC solution is used.
- **ic=node**: The `ic` statements are used, and the `ic` parameter on the capacitors and
  inductors are ignored.
- **ic=dev**: The `ic` parameters on the capacitors and inductors are used, and the `ic`
  statements are ignored.
- **ic=all**: Both the `ic` statements and the `ic` parameters are used, and the `ic`
  parameters override the `ic` statements.

If you specify an initial condition file with the `readic` parameter, initial conditions from the file
are used, and any `ic` statements are ignored.

Once you specify the initial conditions, Spectre computes the actual initial state of the circuit
by performing a DC analysis. During this analysis, Spectre forces the initial conditions on
nodes by using a voltage source in series with a resistor whose resistance is `rforce` (see
options).

With the `ic` statement it is possible to specify an inconsistent initial condition (one that cannot
be sustained by the reactive elements). Examples of inconsistent initial conditions include
setting the voltage on a node with no path of capacitors to ground or setting the current
through a branch that is not an inductor. If you initialize Spectre inconsistently, its solution
jumps; that is, it changes instantly at the beginning of the simulation interval. You should avoid
such changes if possible because Spectre can have convergence problems while trying to
make the jump.

You can skip the DC analysis entirely by using the parameter `skipdc`. If the DC analysis is
skipped, the initial solution will be either trivial, or given in the file you specified by the `readic`
parameter, or, if the `readic` parameter is not given, the values specified on the `ic` statements.
Device based initial conditions are not used for `skipdc`. Nodes that you do not specify with
the `ic` file or `ic` statements will start at zero. You should not use this parameter unless you
are generating a nodeset file for circuits that have trouble in the DC solution; it usually takes
longer to follow the initial transient spikes that occur when the DC analysis is skipped than it
takes to find the real DC solution. The `skipdc` parameter might also cause convergence
problems in the transient analysis.

The possible settings of parameter `skipdc` and their meanings are:

- **skipdc=no**: Initial solution is calculated using the normal DC analysis (default).
skipdc=yes: Initial solution is given in the file specified by the readic parameter or the values specified on the ic statements.

skipdc=waveless: Same initial solution as skipdc=yes, but the waveform production in the time-varying independent sources is disabled during the transient analysis. Independent source values are fixed to their initial values (not their DC values).

skipdc=rampup: Independent source values start at 0 and ramp up to their initial values in the first 10% of the analysis interval. After that their values remain constant. Zero initial solution is used.

skipdc=autodc: Same as skipdc=waveless if a nonzero initial condition is specified. Otherwise, same as skipdc=rampup.

skipdc=sigrampup: Independent source values start at 0 and ramp up to their initial values in the first phase of the simulation. Unlike skipdc=rampup, the waveform production in the time-varying independent source is enabled after the rampup phase. The rampup simulation is from if the start parameter is not specified, the default start time is set to -0.1*stop.

Nodesets help find the DC or initial transient solution. You can supply them in the circuit description file with nodeset statements, or in a separate file using the readns parameter. When nodesets are given, Spectre computes an initial guess of the solution by performing a DC analysis while forcing the specified values onto nodes by using a voltage source in series with a resistor whose resistance is rforce. Spectre then removes these voltage sources and resistors and computes the true solution from this initial guess.

Nodesets have two important uses. First, if a circuit has two or more solutions, nodesets can bias the simulator towards computing the desired one. Second, they are a convergence aid. By estimating the solution of the largest possible number of nodes, you might be able to eliminate a convergence problem or dramatically speed convergence.

When you simulate the same circuit many times, we suggest that you use both the write and readns parameters and give the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.

Nodesets and initial conditions have similar implementation but produce different effects. Initial conditions actually define the solution, whereas nodesets only influence it. When you simulate a circuit with a transient analysis, Spectre forms and solves a set of differential equations. However, differential equations have an infinite number of solutions, and a complete set of initial conditions must be specified in order to identify the desired solution. Any initial conditions you do not specify are computed by the simulator to be consistent. The transient waveforms then start from initial conditions. Nodesets are usually used as a convergence aid and do not affect the final results. However, in a circuit with more than one
solution, such as a latch, nodesets bias the simulator towards finding the solution closest to the nodeset values.

The method parameter specifies the integration method. The possible settings and their meanings are:

**method=euler:** Backward-Euler is used exclusively.

**method=traponly:** Trapezoidal rule is used almost exclusively.

**method=trap:** Backward-Euler and the trapezoidal rule are used.

**method=gear2only:** Gears second-order backward-difference method is used almost exclusively.

**method=gear2:** Backward-Euler and second-order Gear are used.

**method=trapgear2:** Allows all three integration methods to be used.

The trapezoidal rule is usually the most efficient when you want high accuracy. This method can exhibit point-to-point ringing, but you can control this by tightening the error tolerances. For this reason, though, if you choose very loose tolerances to get a quick answer, either backward-Euler or second-order Gear will probably give better results than the trapezoidal rule. Second-order Gear and backward-Euler can make systems appear more stable than they really are. This effect is less pronounced with second-order Gear or when you request high accuracy.

Several parameters determine the accuracy of the transient analysis. reltol and abstol control the accuracy of the discretized equation solution. These parameters determine how well charge is conserved and how accurately steady-state or equilibrium points are computed. You can set the integration error, or the errors in the computation of the circuit dynamics (such as time constants), relative to reltol and abstol by setting the iteratio parameter.

The parameter relref determines how the relative error is treated. The relref options are:

**relref=pointlocal:** Compares the relative errors in quantities at each node to that node alone.

**relref=alllocal:** Compares the relative errors at each node to the largest values found for that node alone for all past time.

**relref=sigglobal:** Compares relative errors in each of the circuit signals to the maximum for all signals at any previous point in time.
relref=allglobal: Same as relref=sigglobal except that it also compares the residues (KCL error) for each node to the maximum of that nodes past history.

The errpreset parameter lets you adjust the simulator parameters to fit your needs quickly. You can set errpreset to conservative if the circuit is very sensitive, or you can set it to liberal for a fast, but possibly inaccurate, simulation. The setting errpreset=moderate suits most needs.

The effect of errpreset on other parameters is shown in the following table. In this table, T= stop - start.

```
errpreset  reltol relref  method      maxstep       iteratio
--------------------------------------------------------------------
liberal      * 10 allglobal gear2       Interval/10   3.5
moderate      sigglobal traponly    Interval/50   3.5
conservative * 0.1 alllocal gear2only  Interval/100  10.0
```

The default value for errpreset is moderate.

The value of reltol is increased or decreased from its value in the options statement, but it is not allowed to be larger than 0.01. Spectre sets the value of maxstep so that it is no larger than the value given in the table. Except for reltol and maxstep, errpreset does not change the value of any parameters you have explicitly set. The actual values used for the transient analysis are given in the log file.

If the circuit you are simulating can have infinitely fast transitions (for example, a circuit that contains nodes with no capacitance), Spectre might have convergence problems. To avoid this, you must prevent the circuit from responding instantaneously. You can accomplish this by setting cmin, the minimum capacitance to ground at each node, to a physically reasonable nonzero value. This often significantly improves Spectre convergence.

Spectre provides two methods for reducing the number of output data points saved: strobing, based on the simulation time, and skipping time points, which saves only every Nth point.

The parameters strobeperiod and strobedelay control the strobing method. strobeperiod sets the interval between points that you want to save, and strobedelay sets the offset within the period relative to skipstart. The simulator forces a time step on each point to be saved, so the data is computed, not interpolated.
The skipping method is controlled by `skipcount`. If this is set to N, then only every Nth point is saved.

The parameters `skipstart` and `skipstop` apply to both data reduction methods. Before `skipstart` and after `skipstop`, Spectre saves all computed data.

If you do not want any data saved before a given time, use `outputstart`. If you do not want any data saved after a given time, change the `stop` time.

The default value for `compression` is no. The output file stores data for every signal at every time point for which Spectre calculates a solution. Spectre saves the x axis data only once, since every signal has the same x value. If `compression=yes`, Spectre writes data to the output file only if the signal value changes by at least 2*the convergence criteria. In order to save data for each signal independently, x axis information corresponding to each signal must be saved. If the signals stay at constant values for large periods of the simulation time, setting `compression=yes` results in a smaller output data file. If the signals in your circuit move around a lot, setting `compression=yes` results in a larger output data file.

**Parameter Index**

In the following index, the number following each parameter name indicates where to find the description of that parameter.

```
annotate  22    flushpoints 33    oppoint 26    skipstop 28

autostop  4    flushtime 34    outputstart 3    start  2

circuitage 40    ic  7    readic  9    stats  21

ckptperiod 14    infoname 36    readns 10    step  6

cmin 11    infotimes 37    relref 17    stop  1

compression 32    lteratio 18    restart  39    strobedelay 31

errpreset 16    maxiters 38    save  24    strobeperiod 30

fastbreak 19    maxstep  5    skipcount 29    title  23

fastcross 20    method  15    skipdc  8    write 12

flushofftime 35    nestlvl 25    skipstart 27    writefinal 13
```
Transfer Function Analysis (xf)

Description

The transfer function analysis linearizes the circuit about the DC operating point and performs a small-signal analysis that calculates the transfer function from every independent source in the circuit to a designated output. The variable of interest at the output can be voltage or current.

You can specify the output with a pair of nodes or a probe component. Any component with two or more terminals can be a voltage probe. When there are more than two terminals, they are grouped in pairs; and you use the portv parameter to select the appropriate pair of terminals. Alternatively, you can simply specify a voltage to be the output by giving a pair of nodes on the xf analysis statement.

Any component that naturally computes current as an internal variable can be a current probe. If the probe component computes more than one current (as transmission lines, microstrip lines, and N-ports do), you use the porti parameter to select the appropriate current. It is an error to specify both portv and porti. If neither is specified, the probe component provides a reasonable default.

The stimuli parameter specifies what is used for the inputs for the transfer functions. There are two choices. stimuli=sources indicates that the sources present in the circuit should be used. The xfmag parameters provided by the sources may be used to adjust the computed gain to compensate for gains or losses in a test fixture. One can limit the number of sources in hierarchical netlists by using the save and nestlvl parameters.

The transfer functions computed versus output and source types are:

<table>
<thead>
<tr>
<th>Source Type</th>
<th>Output Type</th>
<th>Source Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>voltage</td>
<td>current</td>
<td>Amplitude</td>
</tr>
</tbody>
</table>

vsource | V(out)/V(src) | I(out)/V(src) | V(src)=xfmag
isource | V(out)/I(src) | I(out)/I(src) | I(src)=xfmag
port    | 2*V(out)/V(src) | 2*I(out)/V(src) | V(src)=2*xfmag

where xfmag defaults to 1 for each source type. For the port, V(src) is the internal source voltage.
Specifying \texttt{stimuli=nodes_and_terminals} indicates that all possible transfer functions should be computed. This is useful when it is not known in advance which transfer functions are interesting. Transfer functions for nodes are computed assuming that a unit magnitude flow (current) source is connected from the node to ground. Transfer functions for terminals are computed assuming that a unit magnitude potential (voltage) source is connected in series with the terminal. By default, the transfer functions from a small set of terminals are computed. If transfer functions from specific terminals are desired, specify the terminals in the \texttt{save} statement. You must use the \texttt{:probe} modifier (ex. Rout:1:probe) or specify \texttt{useprobes=yes} on the options statement. If transfer functions from all terminals are desired, specify \texttt{currents=all} and \texttt{useprobes=yes} on the options statement.

Spectre can perform the analysis while sweeping a parameter. The parameter can be frequency, temperature, component instance parameter, component model parameter, or netlist parameter. If changing a parameter affects the DC operating point, the operating point is recomputed on each step. You can sweep the circuit temperature by giving the parameter name as \texttt{temp} with no \texttt{dev} or \texttt{mod} parameter. You can sweep a netlist parameter by giving the parameter name with no \texttt{dev}, or \texttt{mod} parameter. After the analysis has completed, the modified parameter returns to its original value.

**Definition**

\texttt{Name \ [p] \ [n] \ xf \ parameter=value ...}

The optional terminals (p and n) specify the output of the circuit. If you do not give the terminals, then you must specify the output with a probe component.

**Parameters**

1. \texttt{prevoppoint=no} \hspace{1cm} Use operating point computed on the previous analysis. Possible values are \texttt{no} or \texttt{yes}.

**Sweep interval parameters**

2. \texttt{start=0} \hspace{1cm} Start sweep limit.
3. \texttt{stop} \hspace{1cm} Stop sweep limit.
4. \texttt{center} \hspace{1cm} Center of sweep.
5. \texttt{span=0} \hspace{1cm} Sweep limit span.
6. \texttt{step} \hspace{1cm} Step size, linear sweep.
7  lin=50            Number of steps, linear sweep.
8  dec               Points per decade.
9  log=50            Number of steps, log sweep.
10 values=[...]      Array of sweep values.

**Sweep variable parameters**

11  dev               Device instance whose parameter value is to be swept.
12  mod               Model whose parameter value is to be swept.
13  param             Name of parameter to sweep.
14  freq (Hz)         Frequency when parameter other than frequency is being swept.

**Probe parameters**

15  probe             Compute every transfer function to this probe component.

**State-file parameters**

16  readns            File that contains estimate of DC solution (nodeset).

**Output parameters**

17  stimuli=sources   Stimuli used for xf analysis.
                        Possible values are sources or nodes_and_terminals.
18  save              Signals to output.
                        Possible values are all, lvl, allpub, lvlpub, selected, or none.
19  nestlvl           Levels of subcircuits to output.
20  oppoint=no        Should operating point information be computed, and if so,
                        where should it be sent. Operating point information would not
                        be output if operating point is computed in the previous analysis
and is unchanged.
Possible values are no, screen, logfile, or rawfile.

**Convergence parameters**

21  restart=yes  
Restart the DC solution from scratch if any condition has changed. If not, use the previous solution as initial guess.
Possible values are no or yes.

**Annotation parameters**

22  annotate=sweep  
Degree of annotation.
Possible values are no, title, sweep, status, or steps.

23  stats=no  
Analysis statistics.
Possible values are no or yes.

24  title  
Analysis title.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, dec) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. All frequencies are in Hertz.

The small-signal analysis begins by linearizing the circuit about an operating-point. By default this analysis computes the operating-point if it is not known, or recomputes it if any significant component or circuit parameter has changed. However, if a previous analysis computed an operating point, you can set prevoppoint=yes to avoid recomputing it. For example, if you use this option when the previous analysis was a transient analysis, the operating point is the state of the circuit on the final time point.

**Parameter Index**

In the following index, the number following each parameter name indicates where to find the description of that parameter.

annotate  22  log  9  probe  15  stats  23
<table>
<thead>
<tr>
<th>Command</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>4</td>
</tr>
<tr>
<td>mod</td>
<td>12</td>
</tr>
<tr>
<td>readns</td>
<td>16</td>
</tr>
<tr>
<td>step</td>
<td>6</td>
</tr>
<tr>
<td>dec</td>
<td>8</td>
</tr>
<tr>
<td>nestlvl</td>
<td>19</td>
</tr>
<tr>
<td>restart</td>
<td>21</td>
</tr>
<tr>
<td>stimuli</td>
<td>17</td>
</tr>
<tr>
<td>dev</td>
<td>11</td>
</tr>
<tr>
<td>oppoint</td>
<td>20</td>
</tr>
<tr>
<td>save</td>
<td>18</td>
</tr>
<tr>
<td>stop</td>
<td>3</td>
</tr>
<tr>
<td>freq</td>
<td>14</td>
</tr>
<tr>
<td>param</td>
<td>13</td>
</tr>
<tr>
<td>span</td>
<td>5</td>
</tr>
<tr>
<td>title</td>
<td>24</td>
</tr>
<tr>
<td>lin</td>
<td>7</td>
</tr>
<tr>
<td>prevoppoint</td>
<td>1</td>
</tr>
<tr>
<td>start</td>
<td>2</td>
</tr>
<tr>
<td>values</td>
<td>10</td>
</tr>
</tbody>
</table>
Syntax

This chapter discusses the following topics:

- Using analogmodel for Model Passing (analogmodel) on page 184
- Behavioural Source Use Model (bsource) on page 186
- Checkpoint - Restart (checkpoint) on page 191
- Configuring CMI Shared Objects (cmiconfig) on page 192
- Built-in Mathematical and Physical Constants (constants) on page 193
- Convergence Difficulties (convergence) on page 195
- Export (export) on page 196
- Expressions (expressions) on page 197
- User Defined Functions (functions) on page 201
- Global Nodes (global) on page 202
- Initial Conditions (ic) on page 203
- The Structural if-statement (if) on page 203
- Include File (include) on page 205
- Spectre Netlist Keywords (keywords) on page 206
- Library - Sectional Include (library) on page 210
- Node Sets (nodeset) on page 213
- Parameter Soft Limits (param_limits) on page 213
- Netlist Parameters (parameters) on page 216
- Parameter Set - Block of Data (paramset) on page 218
- Output Selections (save) on page 219
Sensitivity Analyses (sens) on page 220
SpectreHDL Usage and Language Summary (spectrehdl) on page 221
SpectreRF Summary (spectrerf) on page 230
Subcircuit Definitions (subckt) on page 231
Verilog-A Usage and Language Summary (veriloga) on page 235
Using analogmodel for Model Passing (analogmodel)

Description

analogmodel is a reserved word in Spectre that allows you to bind an instance to different masters based on the value of a special instance parameter called modelname. An instance of analogmodel must have a parameter named modelname whose string value will be the name of the master this instance will be bound to. The value of modelname can be passed into subcircuits.

The analogmodel keyword is used by the Cadence Analog Design Environment to enable model name passing through the schematic hierarchy.

Sample Instance Statement

```
name [([node1 ... nodeN[]]) analogmodel modelname=mastername [[param1=value1] ...[paramN=valueN]]
```

```
name
[[node1...nodeN[]]]
analogmodel
modelname
param1
```

name
Name of the statement or instance label.

[[node1...nodeN[]]]
Names of the nodes that connect to the component.

analogmodel
Special device name to indicate that this instance will have its master name specified by the value of the modelname parameter on the instance.

modelname
Parameter to specify the master of this instance indicated by mastername. The mastername must either be a valid string identifier or a netlist parameter that must resolve to a valid master name - a primitive, a model a subckt, or an AHDL module.

param1
Parameter values for the component. Depending on the master type, these can either be device parameters or netlist parameters. This is an optional field.

Example

```
//example spectre netlist to illustrate modelname parameter

simulator lang=spectre

parameters a="low" b="bottom"
```

ahdl_include "VerilogAStuff.va"

topInst1 (out in) top

topInst2 analogmodel modelname="VAMaster" //VAMaster is defined in "VerilogAStuff.va"

topInst3 (out in) analogmodel modelname="resistor" //topInst3 binds to a primitive

topInst4 (out in) analogmodel modelname="myOwnRes" //topInst4 binds to modelcard "myOwnRes" defined below

topInst5 (out in) weirdRes modelname=modelname //modelname is just another netlist

param

model myOwnRes resistor r=100

subckt (out in) top

parameters a="mid"

x1 (out in) analogmodel modelname=a //topInst1.x1 binds to "mid"

ends top

subckt (out in) mid

parameters c="low"

x1 (out in) analogmodel modelname=b //topInst1.x1.x1 binds to "bottom"

x2 (out in) analogmodel modelname=c //topInst1.x1.x1.x2 binds to "low"

ends mid

subckt (out in) low

x1 (out in) analogmodel modelname="bottom" //topInst1.x1.x1.x2.x1 binds to "bottom"

ends low

subckt (out in) bottom

x1 (out in) analogmodel modelname="resistor" //x1 binds to primitive "resistor"

ends bottom

dc1 dc
Behavioural Source Use Model (bsource)

Description

Behavioural source enables you to model a resistor, inductor, capacitor, voltage or current source as a behavioural component. Using bsource, you can express the value of a resistance, capacitance, voltage or current as a combination of node voltages, branch currents, time expression, and built in Spectre expressions.

In this release bsource simulation performance has been improved by compiling the bsource devices. This is explained in more detail in the bsource compilation section below.

The syntax for bsource is:

```
name (node1 node2) bsource behav_param param_list
```

where behav_param can be

- `c=simple_expr`, Capacitance between the nodes
- `g=simple_expr`, Conductance between the nodes
- `i=generic_expr`, Current through bsource
- `l=simple_expr`, Inductance between the nodes
- `phi=simple_expr`, Flux in the bsource device
- `q=simple_expr`, Charge in bsource device
- `r=simple_expr`, Resistance between the nodes
- `v=generic_expr`, Voltage across the nodes

simple_expr is defined as follows:-

A spectre expression containing,

1. netlist parameters
2. current simulation time, $time
3. node voltages, v(a,b), where a and b are nodes in the spectre netlist
or v(a), which is voltage between node a and ground

4. branch currents, i("inst_id:index"), where inst_id is an instance name
given in the netlist and index is the port index. The default value for
index is 0.
generic_expr is defined as a simple_expr or ddt() or idt() of simple_expr
param_list is param_name=value
param_name can be

Multiplicity factor

m  The value of m will be default to 1.

Temperature Parameters

tc1  Linear temperature co-efficient. Valid for all behavioural elements.
    Default value is 0 1/C.
tc2  Quadratic temperature co-efficient. Valid for all behavioural elements.
    Default value is 0 C^-2
tnom  Parameters measurement temperature. Valid for all behavioural elements.
    Default value is 0.0.
trise  Temperature rise for ambient. Valid for all behavioural elements.
    Default value is 0.0.

Clipping Parameters

max_val  Maximum value of bsource expression. Valid for all behavioural
elements, but generally used with i and v elements for clipping
the current or voltage between the specified values.
min_val  Minimum value of bsource expression. Valid for all behavioural
elements, but generally used with i and v elements for clipping
the current or voltage between the specified values.

Noise Parameters

af  Flicker noise exponent, Valid for r and g elements
    Default value is 2.

fexp  Flicker noise frequency exponent. Valid for r, g, v, and i elements.
    Default value is 1.

isnoisy  Specifies whether to generate noise. Valid for r, g, i, and v elements
    Valid values are yes and no. Default value is yes.

kf  Flicker noise co-efficient. Valid for r and g elements.

white_noise  White noise expression. Valid for v and i elements.

flicker_noise  Flicker noise expression. Valid for v and i elements.

All the parameters in the param_name table are instance parameters. white_noise and
flicker_noise may be assigned behavioural expressions; the other parameters must be
assigned constant or parametric expressions.

Instance Parameters Supported

bsource supports the following instance parameters for the Spectre primitives.

Resistor    isnoisy, m, r, tc1, tc2, and trise

Capacitor   c, m, tc1, tc2, and trise.

Inductor    l, m, tc1, tc2 and trise.

Mathematical Definitions

i = ddt(q) = ddt(simple_expr)

v = ddt(phi) = ddt(simple_expr)

v = i * r = i* simple_expr

v = g * v = simple_expr * v

i = ddt(c * v)
v = ddt(i * l)

Temperature effects on bsource:-

The equation for computing temperature factor is given as follows,

tempFactor = [1 + tc1*(temp-trise+tnom)+tc2*(temp-trise+tnom)^2]

Examples of bsource usage:-

Non-linear resistor/capacitor/inductor modelling

```
res (n1 n2) bsource r=100*(1+(1/2)*v(n1,n2))
res (n1 n2) resistor r=100*(1+(1/2)*v(n1,n2))
cap (n1 n2) bsource c=1.0e-6*(1+(1/2)*v(n1,n2))
cap (n1 n2) capacitor c=1.0e-6*(1+(1/2)*v(n1,n2))
ind (n1 n2) bsource l=0.1*(1+(1/2)*v(n1,n2))
ind (n1 n2) inductor l=0.1*(1+(1/2)*v(n1,n2))
```

Charge model for capacitor

```
cap (n1 n2) bsource q=1.0e-6*v(n1,n2)
```

Voltage and Current (Sinewave) Source

```
vsrc (n1 n2) bsource v=10.0*sin(2*pi*freq*$time)
isrc (n1 n2) bsource i=1.0e-3*sin(2*pi*freq*$time)
```

Current controlled current source

```
vsrc (n1 n2) vsource v=10
cccs1 (n3 n4) bsource i=gain*i("vsrc:0")
```

Current controlled voltage source

```
vsrc (n1 n2) vsource v=10
ccvs1 (n3 n4) bsource v=100*i("vsrc:0")
```

Voltage controlled voltage source
vsrc (n1 n2) resistor r=100k

vcvs1 (n3 n4) bsource v=gain*v(n1,n2)

Voltage controlled current source

vsrc (n1 n2) resistor r=100k

vccs1 (n3 n4) bsource i=v(n1,n2)/2000.0

Giving voltage clipping limit

res (n1 n2) bsource r=100*(1+(1/2)*v(n1,n2)) max_val=105 min_val=95

Giving Temperature Co-efficient for Resistor

res (n1 n2) bsource r=100 tc1=0.01 tc2=0.003 trise=10 tnom=30

Doing Altergroup with bsource

vsrc1 (n1 n2) bsource v=10*sin(2*pi*freq1*$time)

vsrc2 (n3 n4) bsource v=10*cos(2*pi*freq2*$time)

cccs1 (n5 n6) bsource i=gain*i("vsrc1:0")

res (n5 n6) bsource r=100*(1+(1/2)*v(n5,n6))

tran1 tran stop=1u

altAnal altergroup {

cccs1 (n5 n6) bsource i=gain*i("vsrc2:0")

res (n5 n6) bsource r=100*(1+(1/3)*pow(v(n5,n6),2))

}

tran2 tran stop=1u

Note: For resistor/capacitor/inductor, spectre netlist could directly be given with non-linear equation. However, they will internally be treated as behavioural source.

bsource compilation

====================
In this release the performance of bsource devices has been improved by performing a one-time compilation step. The performance improvement obtained is proportional to the complexity of the bsource expression. Following the initial compilation, recompilation will only be performed if the bsource expression is changed.

Bsource compilation is enabled by default. If you are making frequent changes to bsource expressions used in your design, the overhead of the compilation step may become an issue. To turn off compilation, set the CDS_AHDLCMI_ENABLE shell environment variable to NO e.g:

```
setenv CDS_AHDLCMI_ENABLE NO
```

To re-enable bsource compilation set the CDS_AHDLCMI_ENABLE to YES e.g:

```
setenv CDS_AHDLCMI_ENABLE YES
```

or undefine the CDS_AHDLCMI_ENABLE environment variable e.g:

```
unsetenv CDS_AHDLCMI_ENABLE
```

**Checkpoint - Restart (checkpoint)**

**Description**

Spectre has the ability to save checkpoint files while the analyses are running, and to restart an analysis from its checkpoint file. Checkpoint files can be generated in several ways:

1) Periodically based on real time (wall clock time).

2) Asynchronous UNIX signals.

3) By other methods unique to the analyses.

To generate checkpoint files periodically based on real time, set the Spectre option `ckptclock` to the time interval in seconds that you want checkpoints. This option is turned on by default with a value of 1800 seconds (30 minutes). Spectre will delete the checkpoint
file if the simulation completes normally. If the simulation terminates abnormally, the checkpoint file will not be deleted.

If Spectre receives the UNIX signal USR2, then Spectre will immediately write a checkpoint file. If Spectre receives interrupt signals like QUIT, TERM, INT, or HUP, Spectre will attempt to write a checkpoint file and then exit. After other fatal signals, it may not be possible for Spectre to write a checkpoint file.

The name of the checkpoint file is a combination of the circuit name, the analysis name, and the extension .ckpt. For example, if the circuit is named test1 and the transient analysis is named timeSweep, then the checkpoint file will be named test1.timeSweep.tran.ckpt.

Spectre keeps only the latest checkpoint file. When a new checkpoint is created, it creates the file under a temporary name. After the file has been successfully written, it deletes the previous checkpoint file and renames the new file.

Currently only the transient analyses supports checkpoint and restart.

Checkpoint

The transient analysis can generate checkpoint files by using the above methods, or by generating a checkpoint file periodically based on the transient simulation time. This is accessed by a transient analysis parameter called ckptperiod, which is turned off by default.

Restart

To restart an analysis from a checkpoint file, use the +recover option on the Spectre command line. Spectre will look through the requested analyses to see if a checkpoint file exists for any of them. If a checkpoint file for a given analysis does exist, Spectre will skip over any analyses previous to that one, and start the analysis using the information from the file.

Configuring CMI Shared Objects (cmiconfig)

Description

Spectre supports the ability to install devices dynamically from shared objects at run time. CMI Configuration files are used to determine and locate the set of shared objects to be installed. Spectre first reads the default CMI configuration file which specifies the default shared objects provided by Cadence. The configuration file specified by the value of the CMI_CONFIG environment variable is then read. The third configuration file that Spectre reads is ~/.cmiconfig. Finally, the configuration file specified in Spectres -cmiconfig command
line argument is read. Each CMI configuration file modifies the existing configuration established by the configuration files read before.

The following commands can be used in a CMI configuration file.

**setpath**
Specifies and resets the search path.
setpath <path> or setpath ( <path1> <path2> ... <pathN> )

**prepend**
Adds a path before the current search path.
prepend <path> or prepend ( <path1> <path2> ... <pathN> )

**append**
Adds a path after the current search path.
append <path> or append ( <path1> <path2> ... <pathN> )

**load**
Add a shared object to the list of shared objects to load.
loads [path/]<shared_object_name>

**unload**
Removes a shared object to the list of shared objects to load.
unload <shared_object_name>

For example, given the following CMI configuration file

```
   append /hm/spectre_dev/tools.sun4v/spectrecmi/lib/cmi/1.0
   load libbjtx+tfet.so
   load libmosx.so
```

The shared objects **libbjtx+tfet.so** and **libmosx.so** are loaded from **/hm/spectre_dev/tools.sun4v/spectrecmi/lib/cmi/1.0** in addition to the default shared objects provided by Cadence.

**Built-in Mathematical and Physical Constants (constants)**

**Description**
Spectre supports the following list of built-in mathematical and physical constants:

- **M_** is a mathematical constant

  - **M_E** \(= 2.7182818284590452354\) \(\exp(1) = e\)
Spectre Circuit Simulator Reference
Syntax

M_LOG2E     1.4426950408889634074  log2(e)
M_LOG10E    0.43429448190325182765  log10(e)
M_LN2       0.69314718055994530942  ln(2)
M_LN10      2.30258509299404568402  ln(10)
M_PI        3.14159265358979323846  pi
M_TWO_PI    6.28318530717958647652  2 * pi
M_PI_2      1.57079632679489661923  pi/2
M_PI_4      0.78539816339744830962  pi/4
M_1_PI      0.31830988618379067154  1/pi
M_2_PI      0.63661977236758134308  2/pi
M_2_SQRTPI  1.12837916709551257390  2/sqrt(pi)
M_SQRT2     1.41421356237309504880  sqrt(2)
M_SQRT1_2   0.70710678118654752440  sqrt(1/2)
M_DEGPERRAD 57.2957795130823208772  number of degrees per radian

P_ is a physical constant

P_Q         1.6021918e-19  charge of electron in coulombs
P_C         2.997924562e8  speed of light in vacuum in meters/sec
P_K         1.3806226e-23  Boltzmanns constant in joules/kelvin
P_H         6.6260755e-34  Plancks constant in joules*sec
P_EPS0      8.85418792394420013968e-12  permittivity of vacuum in farads/meter
P_U0        (4.0e-7 * M_PI)  permeability of vacuum in henrys/meter
P_CELSIUS0  273.15  zero celsius in kelvin

These constants can be used in expressions, or anywhere that a numeric value of expression is expected.
Convergence Difficulties (convergence)

Description

If you are having convergence difficulties, try the following suggestions:

1. Carefully evaluate and resolve any notice, warning or error messages.

2. Assure topology checker is used (set topcheck=full on options statement) and heed any warnings it gives.

3. Perform sanity checking on the parameter values using the parameter range checker (use +param param-limits-file as a command line argument) and heed any warnings. Print the minimum and maximum parameter value using the info analysis. Assure that the bounds given for instance, model, output, temperature-dependent, and operating-point (if possible) parameters are reasonable.

4. Small floating resistors connected to high impedance nodes can cause convergence difficulties. Avoid very small floating resistors, particularly small parasitic resistors in semiconductors. Use voltage sources or iprobes to measure current rather than small resistors.

5. Use realistic device models. Check all component parameters, particularly nonlinear device model parameters, to assure that they are reasonable.

6. Increase the value of gmin (on options statement).

7. Loosen tolerances, particularly absolute tolerances like iabstol (on options statement). If tolerances are set too tight, they might preclude convergence.

8. Try to simplify the nonlinear component models in order to avoid regions in the model that may contribute to convergence problems.

DC Convergence Suggestions

Once you have a solution, write it to a nodeset file using the write parameter, and read it back in on subsequent simulations using the readns parameter.

1. If you have an estimate of what the solution should be, use nodeset statements or a nodeset file and set as many nodes as possible.

2. If convergence difficulties occur when using nodesets or initial conditions, try increasing rforce (on options statement).
3. If this is not the first analysis, perhaps the solution from the previous analysis is far from
the solution for this analysis. If so, set restart=yes.

4. If simulating a bipolar analog circuit, assure the region parameter on all transistors and
diodes is set correctly.

5. If analysis fails at an extreme temperature, but succeeds at room temperature, then try
adding a DC analysis that sweeps temperature. Start at room temperature, sweep to the
extreme temperature, and write the final solution to a nodeset file.

6. Use numeric pivoting in the sparse matrix factorization by setting pivotdc=yes (on
options statement). Sometimes it is also necessary to increase the pivot threshold to
somewhere in the range of 0.1 to 0.5 using pivrel (on options statement).

7. Divide the circuit into smaller pieces and simulate them individually, but be careful to
assure that the results will be close to what they would be if the rest of the circuit was present.
Use the results to generate nodesets for the whole circuit.

8. If all else fails, replace the DC analysis with a transient analysis and modify all the
independent sources to start at zero and ramp to their DC values. Run the transient analysis
well beyond the time when all the sources have reached their final value (remember that
transient analysis is very cheap when all of the signals in the circuit are not changing) and
write the final point to a nodeset file. To make the transient analysis more efficient set the
integration method to backward Euler (method=euler) and loosen the local truncation error
criteria by increasing lteratio, say to 50. Occasionally, this approach will fail or be very
slow because the circuit contains an oscillator. Often times the oscillation can be eliminated
for the sake of finding the dc solution by setting the minimum capacitance from each node to
ground (cmin) to a large value.

Transient Convergence Suggestions

1. Assure that a complete set of parasitic capacitors is used on nonlinear devices to avoid
jumps in the solution waveforms. On MOS models, specify nonzero source and drain areas.

2. Use the cmin parameter to install a small capacitor from every node in the circuit to
ground. This usually eliminates any jumps in the solution.

Export (export)

The export feature is not supported. It is designated for internal use only.
Expressions (expressions)

Description
An expression is a construct that combines operands with operators to produce a result that is a function of the values of the operands and the semantic meaning of the operators. Any legal operand is also an expression in itself. Legal operands include numeric constants and references to top-level netlist parameters or subcircuit parameters. Calls to algebraic and trigonometric functions are also supported. The complete lists of operators, algebraic, and trigonometric functions are given after some examples.

Examples:
simulator lang=spectre
parameters p1=1 p2=2 // declare some top-level parameters
r1 (1 0) resistor r=p1 // the simplest type of expression
r2 (1 0) resistor r=p1+p2 // a binary (+) expression
r3 (1 0) resistor r=5+6/2 // expression of constants, = 8
x1 s1 p4=8 // instantiate a subcircuit, defined in the following lines
subckt s1
parameters p1=4 p3=5 p4=6 // subcircuit parameters
r1 (1 0) resistor r=p1 // another simple expression
r2 (1 0) resistor r=p2*p2 // a binary multiply expression
r3 (1 0) resistor r=(p1+p2)/p3 // a more complex expression
r4 (1 0) resistor r=sqrt(p1+p2) // an algebraic function call
r5 (1 0) resistor r=3+atan(p1/p2) // a trigonometric function call
r6 (1 0) RESMOD r=(p1 p4+1 : p3) // the ternary operator
ends
// a model card, containing expressions
model RESMOD resistor tc1=p1+p2 tc2=sqrt(p1*p2)

// some expressions used with analysis parameters

time_sweep tran start=0 stop=(p1+p2)*50e-6 // use 5*50e-6 = 150 us

// a vector of expressions (see notes on vectors below)

dc_sweep dc param=p1 values=[0.5 1 +p2 (sqrt(p2*p2))] // sweep p1

Where Expressions Can Be Used:

The Spectre native netlist language allows expressions to be used where numeric values are expected on the right-hand side of an "=" sign, or within a vector, where the vector itself is on the right-hand side of an "=" sign. Expressions can be used when specifying device or analysis instance parameter values (for example specifying the resistance of a resistor or the stop time of a transient analysis, as outlined in the preceding example), when specifying model parameter values in model cards (for example specifying "bf=p1*0.8" for a bipolar model parameter, bf), or when specifying initial conditions and nodesets for individual circuit nodes.

Operators:

The following operators are supported, listed in order of decreasing precedence. Parentheses can be used to change the order of evaluation. For a binary expression like "a+b", "a" is the first operand and "b" is the second operand. All operators are left associative, with the exceptions of the "to the power of" operator (**) and the ternary operator (? : ), which are right associative. For logical operands, any nonzero value is considered true. The relational and equality operators return a value of 1 to indicate true or 0 to indicate false. There is no short circuiting of logical expressions involving && and ||.

Operator | Symbol(s) | Value
--- | --- | ---
Unary +, Unary - | +, - | Value of operand, negative of operand.
To the power of | ** | First operand to raised to power of second operand.
Multiply, Divide | *, / | Product, Quotient of operands.
Binary Plus/Minus | +, - | Sum, Difference of operands.
Shift | <<, >> | First operand shifted left (, right) by number of bits specified by second
operand.

Relational
<, <=, >, >= Less than, less than or equal,
greater than, greater than or equal.

Equality
==, != True if operands are equal, not equal.

Bitwise AND
& Bitwise AND (of integer operands).

Bitwise Exclusive NOR
^ (or ^~) Bitwise Exclusive NOR (of integer operands).

Bitwise OR
| Bitwise OR (of integer operands).

Logical AND
&& True only if both operands true.

Logical OR
|| True if either operand is true.

Ternary Operator
(cond) ? x : y Returns x if cond is true, y if not
where x and y are expressions.

Algebraic and Trigonometric Functions:

The trigonometric and hyperbolic functions expect their operands to be specified in radians. The atan2() and hypot() functions are useful for converting from Cartesian to polar form.

Function Description Domain
log(x) Natural logarithm x > 0
log10(x) Decimal logarithm x > 0
exp(x) Exponential x < 80
sqrt(x) Square Root x > 0
min(x,y) Minimum value All x, all y
max(x,y) Maximum value All x, all y
abs(x) Absolute value All x
pow(x,y) x to the power of y All x, all y
Spectre Circuit Simulator Reference
Syntax

int(x)          integer value of x      All x
floor(x)        largest integer <= x    All x
ceil(x)         smallest integer >= x   All x
fmod(x,y)       floating point modulus  All x, all y, except y=0
sin(x)          Sine                    All x
cos(x)          Cosine                  All x
tan(x)          Tangent                 All x, except
                x = n*(pi/2), where n odd
asin(x)         Arc-sine                -1 <= x <= 1
acos(x)         Arc-cosine              -1 <= x <= 1
atan(x)         Arc-tangent             All x
atan2(x,y)      Arc-tangent of x/y      All x, all y
hypot(x,y)      sqrt(x*x + y*y)         All x, all y
sinh(x)         Hyperbolic sine         All x
cosh(x)         Hyperbolic cosine        All x
tanh(x)         Hyperbolic tangent       All x
asinh(x)        Arc-hyperbolic sine     All x
acosh(x)        Arc-hyperbolic cosine    x >= 1
atanh(x)        Arc-hyperbolic tangent   -1 <= x <= 1

User-defined functions are also supported. See `spectre -h functions` for a description of user-defined functions.

A large number of built-in mathematical and physical constants are available for use in expressions. See `spectre -h constants` for the list of these constants

Using Expressions in Vectors:
Expressions can be used as vector elements, as in the following example:
dc_sweep dc param=p1 values=[0.5 1 +p2 (sqrt(p2*p2)) ] // sweep p1

Note that when expressions are used within vectors, anything other than constants, parameters, or unary expressions (unary +, unary -) must be surrounded by parentheses. Vector elements should be space separated for clarity, though this is not mandatory. The preceding "dc_sweep" example shows a vector of four elements, namely 0.5, 1, +p2, and sqrt(p2*p2). Note that the square root expression is surrounded by parentheses.

User Defined Functions (functions)

Description

Spectre's user-defined function capability allows you to build upon the provided set of built-in mathematical and trigonometric functions. You can write your own functions, and call these functions from within any expression. The syntax for calling a user-defined function is the same as the syntax for calling a built-in algebraic or trigonometric function. Note that user-defined functions must be defined before they are referenced (called). Arguments to user-defined functions will be taken as real values, and the functions will return real values. A user-defined function may contain only a single statement in braces and this statement must return an expression (which will typically be an expression involving the function arguments). The return expression may reference the built-in parameters temp and tnom. User-defined functions must be declared at the top level only, and must not be declared within subcircuits. User-defined functions may be called from anywhere that an expressions can be currently used in Spectre. User-defined functions may call other functions (both user-defined and built-in), however any user-defined function will need to be declared before it can be called. User-defined functions can override built-in mathematical and trigonometric functions.

NOTE: Only real values for arguments and return values are supported in this release.

See spectre -h expressions for a list of built-in algebraic and trigonometric functions.

Definition

    real myfunc( [real arg1, ...real argn] ) {

Examples:

        real myfunc( real a, real b ) {

        return a+b*2+sqrt(a*sin(b));

    }
An example of a function calling a previously defined function follows

```c
real yourfunc( real a, real b ) {
    return a+b*myfunc(a,b);  // call "myfunc"
}
```

The final example shows how a user-defined function may be called from an expression in the Spectre netlist:

```c
r1 (1 0) resistor r=myfunc(2.0, 4.5)
```

**Global Nodes (global)**

**Description**

The global statement allows a set of nodes to be designated as common to the main circuit and all subcircuits. Thus, components inside subcircuits can be attached to global nodes even though the subcircuits terminals are not attached to these nodes.

Any number of global nodes may be specified using the global statement. To do this, follow the keyword global with a list of the node names that you wish to declare as global. The first node name that appears in this list is taken to be the name of the ground node. Ground is also known as the datum or reference node. If a global statement is not used, 0 is taken to be the name of the ground node.

At most one global statement is allowed and, if present, it must be the first statement in the file (however, you can have `simulator lang=spectre` statement, before the global statement so that you can use mixed case names for the node names). Ground is always treated as global even if a global statement is not used.

**Definition**

```c
global <ground
```
Initial Conditions (ic)

Description

The `ic` statement is used to provide initial conditions for nodes in the transient analysis. It can occur multiple times in the input and the information provided in all the occurrences is collected. Initial conditions will only be accepted for inductor currents and node voltages where the nodes have a path of capacitors to ground. For more information, read the description of transient analysis. It should be noted that specifying `cmin` for a transient analysis, will not satisfy the condition that a node has a capacitive path to ground.

Definition

```
ic <node=value
```

This statement takes a list of signals as an argument. The concept of nodes has been generalized to signals where a signal is a value associated with a topological node of the circuit or some other unknown that is solved by the simulator, such as the current through a inductor or the voltage of the internal node in a diode. Topological nodes can be either at the top-level or in a subcircuit.

For example,

```
ic 7=0 out=1 OpAmp1.comp=5 L1:1=1.0u
```

where `7=0` implies that node `7` should start at 0V, node `out` should start at 1V, node `comp` in subcircuit `OpAmp1` should start at 5V, and the current through the first terminal of `L1` should start at 1uA.

The Structural if-statement (if)

Description

The structural if-statement can be used to conditionally instantiate other instance statements.

Definition

```
if <condition
```

The condition is a boolean expression based on the comparisons of various arithmetic expressions which are evaluated during circuit hierarchy flattening. The statement1 and
statement2 fields can be ordinary instance statements, if-statements, or a list of these within braces ({}). (Note that ordinary instance statements need a newline to terminate them.) The else part is optional. When if-statements are nested without braces, an else matches the closest previous unmatched if at the same level.

It is possible to have duplicate instance names within the if statement under strict topological conditions. These are:

* references to instance with duplicate names is only possible within a structural if statement which has both an "if" part and an "else" part.
* both the "if" part and the "else" part must be either a simple one-statement block, or another structural if statement to which these same rules apply.
* The duplicate instances must have the same number of terminals and be bound to the same list of nodes.
* The duplicate instances must refer to the same primitive or model.
* Where duplicate instances refer to a model, the underlying primitive must be the same.

This feature allows automatic model selection based on any netlist or subcircuit parameter. As an example, consider using Spectre's inline subcircuits and structural if statement to implement automatic model selection based on bipolar device area. Here, the duplicate instances are the inline components.

```
model npn_default bjt is=3.2e-16 va=59.8
model npn10x10 bjt is=3.5e-16 va=61.5
model npn20x20 bjt is=3.77e-16 va=60.5

// npn_mod choses scaled models binned on area!
// if ( area < 100e-12 ) use model npn10x10
// else if ( area < 400e-12 ) use model npn20x20
// else use model npn_default
```
inline subckt npn_mod (c b e s)
parameters area=5e-12
if ( area < 100e-12 ) {
    npn_mod (c b e s) npn10x10   // 10u * 10u, inline device
} else if ( area < 400e-12 ) {
    npn_mod (c b e s) npn20x20   // 20u * 20u, inline device
} else {
    npn_mod (c b e s) npn_default   // 5u * 5u, inline device
}

ends npn_mod

q1 (1 2 0 0) npn_mod area=350e-12       // gets 20x20 model
q2 (1 3 0 0) npn_mod area=25e-12        // gets 10x10 model
q3 (1 3 0 0) npn_mod area=1000e-12      // gets default model

**Include File (include)**

**Description**

File inclusion allows the circuit description to be spread over several files. The include statement itself is replaced by the contents of the file named. An included file may also contain include statements. If the name given is not an absolute path specification, then it is taken relative to the directory of the file currently being read.

In order to read existing SPICE library and model files, Spectre automatically switches to SPICE input mode when it opens an include file. Thus, all files that use the Spectre native language must begin with a `simulator lang=spectre` statement. The one exception is files that end with a ".scs" file extension which are treated specially and are read in Spectre input mode. This language mode treatment applies to files included by both Spectres include statement, and CPPs #include statement.
After reading the include file, Spectre restores the language processing mode to what it was before the file was included, and continues reading the original file starting at the line after the include statement. Lines cannot be continued across file boundaries.

The CPP #include statement differs from Spectres include statement in that CPP macro processing will not be performed on files included by Spectre, but will be performed on files included by CPP. If your netlist contains a #include statement, you must run CPP to perform this inclusion, otherwise an error will occur.

If the file to be included cannot be found in the same directory as the including file, Both Spectres include and CPPs #include will search for the file to be included along the search path specified by the -I command line arguments.

Spectres include statement allows you to embed special characters in the name of the file to be included. Spectres include statement will automatically expand the "~" character to the users home directory, and will expand environment variables and % codes, such as

include "~/models/${SIMULATOR}_pd/npn.scs"

which will look in the directory given by the environment variable SIMULATOR followed by "_pd", which is under the "models" directory, in the users home directory. Note: These special character features are not available using CPPs #include statement.

Definition

\textit{include \textit{filename}}

**Spectre Netlist Keywords (keywords)**

**Description**

The following lists the reserved Spectre keywords, including netlist keywords, built-in algebraic and trigonometric functions, and built-in mathematical and physical constants. When creating a netlist, you should avoid using any of the keywords from this list in any context other than that in which it was intended. Creating nodenames, parameter names, instance names, or model names from any of these keywords will result in an error.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Keyword Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>M_1_PI</td>
<td>Mathematical Constant</td>
</tr>
</tbody>
</table>
M_2_PI                  Mathematical Constant
M_2_SQRTP1              Mathematical Constant
M_DEGPERRAD              Mathematical Constant
M_E                     Mathematical Constant
M_LN10                  Mathematical Constant
M_LN2                   Mathematical Constant
M_LOG10E                 Mathematical Constant
M_LOG2E                  Mathematical Constant
M_PI                    Mathematical Constant
M_PI_2                  Mathematical Constant
M_PI_4                  Mathematical Constant
M_SQRT1_2                Mathematical Constant
M_SQRT2                  Mathematical Constant
M_TWO_PI                 Mathematical Constant
P_C                      Mathematical Constant
P_CELSIUS0               Mathematical Constant
P_EPS0                   Mathematical Constant
P_H                      Mathematical Constant
P_K                      Mathematical Constant
P_Q                      Mathematical Constant
P_U0                     Mathematical Constant
abs                      Algebraic Function
acos                     Trigonometric Function
acosh                    Trigonometric Function
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>altergroup</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>asin</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>asinh</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>atan</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>atan2</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>atanh</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>ceil</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>correlate</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>cos</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>cosh</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>else</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>end</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>ends</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>exp</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>export</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>floor</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>fmod</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>for</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>function</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>global</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>hypot</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>ic</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>if</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>inline</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>Word</td>
<td>Type</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------</td>
</tr>
<tr>
<td>int</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>invisible</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>library</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>local</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>log</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>log10</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>march</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>max</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>min</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>model</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>nodeset</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>parameters</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>paramset</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>plot</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>pow</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>print</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>pwr</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>real</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>return</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>save</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>sens</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>sin</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>sinh</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>sqrt</td>
<td>Algebraic Function</td>
</tr>
<tr>
<td>Keyword</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>statistics</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>subckt</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>tan</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>tanh</td>
<td>Trigonometric Function</td>
</tr>
<tr>
<td>to</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>truncate</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>vary</td>
<td>Netlist Keyword</td>
</tr>
<tr>
<td>visible</td>
<td>Netlist Keyword</td>
</tr>
</tbody>
</table>

## Library - Sectional Include (library)

### Description

Library inclusion allows the circuit description to be spread over several files. The library statement itself is replaced by the contents of the section of the library file. A library section may also contain library reference statements. If the file name given is not an absolute path specification, then it is taken relative to the directory of the file currently being read.

There are two kinds of library statements. One that references a library section, and one that defines a library section. The definition of a library section is prohibited in the netlist.

In order to read existing SPICE library and model files, Spectre automatically switches to SPICE input mode when it opens a library file. Thus, all files that use the Spectre native language must contain a simulator lang=spectre statement within each section of the library or the file can have a .scs filename extension. After reading the library section, Spectre restores the language processing mode and continues reading the original file starting at the line after the library statement. Lines cannot be continued across file boundaries.

Spectre allows only one library per file, but a library may contain multiple sections (typically one section per process corner for example.)

### Definition

Inside netlist (reference library section)
Sample Library:

library corner_lib

section tt
  model nch bsim3v3 type=n mobmod=1 capmod=2 version=3.1
  + xj=1.7e-7 vsat=7.99e4 at=3.6e4 a0=0.799 ags=0.4
  + a1=0 a2=1 keta=-0.05 nch=2.8e17 ngate=1.31e20 k1=0.74
  model pch bsim3v3 type=p mobmod=1 capmod=2 version=3.1
  + xj=1.7e-7 vsat=1.38e5 at=1e5 a0=1.3 ags=0.3
  + a1=1.1e-4 a2=1 keta=0 nch=4.1e17 ngate=7.6e19 k1=0.88
  model knpn bjt is=10e-13 bf=170 va=58.7 ik=5.63e-3 rb=rbn rbm=86
  + re=3.2 cje=0.25e-12 pe=0.76 me=0.34 tf=249e-12 cjc=0.34e-12 pc=0.55
  + mc=0.35 ccs=2.4e-12 ms=0.35 ps=0.53 rc=169
  model kpn p bjt type=pnp is=10e-13 bf=60 va=43.1 ik=0.206e-3 rb=rbp rbm=64.3
  + re=33.8 cje=0.16e-12 pe=0.5 me=0.26 tf=36e-9 cjc=0.72e-12 pc=0.58
  + mc=0.34 ccs=2.5e-12 ps=0.53 ms=0.35 rc=276
endsection

section ss
  model nch bsim3v3 type=n mobmod=1 capmod=2 version=3.1
  + xj=1.7e-7 vsat=7.99e4 at=3.6e4 a0=0.799 ags=0.4
  + a1=0 a2=1 keta=-0.05 nch=2.8e17 ngate=1.31e20 k1=0.74
  model pch bsim3v3 type=p mobmod=1 capmod=2 version=3.1
  + xj=1.7e-7 vsat=1.38e5 at=1e5 a0=1.3 ags=0.3
  + a1=1.1e-4 a2=1 keta=0 nch=4.1e17 ngate=7.6e19 k1=0.88
  model knpn bjt is=10e-13 bf=70 va=58.7 ik=5.63e-3 rb=rbn rbm=86
section ff

model nch bsim3v3 type=n mobmod=1 capmod=2 version=3.1
+ xj=1.7e-7 vsat=7.99e4 at=3.6e4 a0=0.799 ags=0.4
+ a1=0 a2=1 keta=-0.05 nch=2.8e17 ngate=1.31e20 k1=0.74
model pch bsim3v3 type=p mobmod=1 capmod=2 version=3.1
+ xj=1.7e-7 vsat=1.38e5 at=1e5 a0=1.3 ags=0.3
+ a1=1.1e-4 a2=1 keta=0 nch=4.1e17 ngate=7.6e19 k1=0.88
model knpn bjt is=10e-13 bf=220 va=58.7 ik=5.63e-3 rb=rbn rbm=86
+ re=3.2 cje=0.25e-12 pe=0.76 me=0.34 tf=249e-12 cjc=0.34e-12 pc=0.55
+ mc=0.35 ccs=2.4e-12 ms=0.35 ps=0.53 rc=169
model kpn bjt type=pnp is=10e-13 bf=90 va=43.1 ik=0.206e-3 rb=rbp rbm=64.3
+ re=33.8 cje=0.16e-12 pe=0.5 me=0.26 tf=36e-9 cjc=0.72e-12 pc=0.58
+ mc=0.34 ccs=2.5e-12 ps=0.53 ms=0.35 rc=276
endsection
Node Sets (nodeset)

Description

The nodeset statement is used to provide an initial guess for nodes in any DC analysis or the initial condition calculation for the transient analysis. It can occur multiple times in the input, the information provided in all the occurrences is collected. For more information, read the description of DC analysis.

Definition

nodeset <node=value

This statement takes a list of signals as an argument. The concept of nodes has been generalized to signals where a signal is a value associated with a topological node of the circuit or some other unknown that is solved by the simulator, such as the current through a inductor or the voltage of the internal node in a diode. Topological nodes can be either at the top-level or in a subcircuit.

For example,

    nodeset 7=0 out=1 OpAmp1.comp=5 L1:1=1.0u

where 7=0 implies that node 7 should be about 0V, node out should be about 1V, node comp in subcircuit OpAmp1 should be about 5V, and the current through the first terminal of L1 should be about 1uA.

Parameter Soft Limits (param_limits)

Description

The parameter values passed to Spectre components and analysis are subject to both hard and soft limits. If you set a parameter to a value that violates a hard limit, such as giving z0=0 to a transmission line, Spectre issues an error message and quits. If the given parameter value violates a soft limit, only a warning is issued, but Spectre uses the value of the component as given. Hard limits are used to prevent you from using values that would cause Spectre to fail or put a model in an invalid region. Soft limits are used to call attention to unusual parameter values that might have been given mistakenly. If a parameter value violates a soft limit, a message similar to one of the following sample messages is printed:

    Parameter rb has the unusually small value of 1uOhms.
or

Parameter \( r_b \) has the unusually large value of 1MOhms.

Spectre has built-in soft limits on a few parameter values. However, it is possible for you to override these limits, or provide limits on parameters that do not have built-in limits. To do so, create a parameter range limits file, and invoke Spectre giving the name of the file after the +param command line option. For example,

```
spectre +param limits-file input-file
```

Limits are given using the following syntax:

```
[PrimitiveName] [model] [LowerLimit <=] ||Param|| [<=] UpperLimit]
```

The limits can be given as strict (using \( <= \)) or nonstrict (using \( < \)). If the limits are strict, there can be no space between \( < \) and \( = \). The limits for one parameter are given on one line. There is no way of continuing the specification of the limits for a parameter over more than one line. If a parameter is given more than once, the limits given the last time override earlier limits. The primitive name must be a Spectre primitive name, not a name used for SPICE compatibility. So, for example, mos3 must be used rather than mos. Parameter limits can be written using Spectre native mode metric scale factors. Thus a limit of \( f <= 1.0e6 \) could also written as \( f <= 1M \).

Examples

```
mos3 0.5u <= l <= 100u

0.5u <= w

0 < as <= 1e-8

0 < ad <= 1e-8

model \|vto\| <= 3
```

Notice that it is not necessary to give the primitive name each time. If not given, it is assumed to be the same as the previous parameter. Upper and lower limits may be given, and if not given there will be no limit on the parameter value. Thus, in the example, if \( w \) is less than 0.5um, a warning will be issued, but there is no limit on how large \( w \) can be. If a parameter is mentioned, but no limits given, then all limits are disabled for that parameter. Limits are placed on model parameters by giving the model keyword. If the model keyword is not given, the limits are applied to instance parameters. Notice that you can also place upper or lower limits on the absolute value of a parameter. For example,
resistor \(0.1 < |r| < 1M\)

indicates that the absolute value of \(r\) should be greater than 0.1 Ohm and less than 1 MOhm. There can be no spaces between the absolute value symbols and the parameter name.

Examples

\[
1 \leq x < 0.5
\]

\[
1 \leq y \leq 1
\]

\[
1 < z < 1
\]

In the first case the lower bound is larger than the upper bound, which indicates that the range of \(x\) is all real numbers except those from 0.5 to 1 and 0.5 itself. The limits are applied separately, thus \(x\) must be both greater than or equal to 1 \((1 \leq x)\) and less than 0.5 \((x < 0.5)\). The second case specifies that \(y\) should be 1 and the third case specifies that \(z\) should not be 1.

It is possible to specify limits for any scalar parameter that takes either a real number, an integer, or an enumeration. To specify the limits of a parameter that takes enumerations use the indices associated with the enumerations. For example, consider the region parameter of the bjt. There are four possible regions: off, fwd, rev, or sat (see `spectre -help bjt`). Each enumeration is assigned a number starting at 0 and counting up. Thus, off=0, fwd=1, rev=2, and sat=3. The specification

```
bjt
3 \leq\ region \leq\ 1
```

indicates that a warning should be printed if \(\text{region}=\text{rev}\) because the conditions \(3 \leq\ \text{region}\) and \(\text{region} \leq\ 1\) exclude only \(\text{region}=2\) and region 2 is rev.

It is possible to read a parameter limits file from within another file. To do so, use an include statement. For example,

```
include "filename"
```

will temporarily suspend the reading of the current file until the contents of `filename` have been read. Include statements may be nested arbitrarily deep with the condition that the operating system may limit the number of files that Spectre may have open at once. Paths in file names are taken to be relative to the directory that contains the current file, not from the directory in which Spectre was invoked.

Spectre can be instructed to always read a parameter limits file by using the `SPECTRE_DEFAULTS` environment variable. For example, if you put the following in your shell initialization file (.profile for sh, .cshrc for csh)
setenv SPECTRE_DEFAULTS "+param /cds/etc/spectre/param.lmts"

Spectre would always read the specified limits file.

**Netlist Parameters (parameters)**

**Description**

The Spectre native netlist language allows parameters to be specified and referenced in the netlist, both at the top-level scope and within subcircuit declarations (run `spectre -h subckt` for more details on parameters within subcircuits).

**Definition**

```
parameters <param=value
```

**Examples:**
```
simulator lang=spectre
parameters p1=1 p2=2       // declare some parameters
 r1 (1 0) resistor r=p1    // use a parameter, value=1
 r2 (1 0) resistor r=p1+p2  // use parameters in an expression, value=3
 x1 s1 p4=8               // "s1" is defined below, pass in value 8 for "p4"
 subckt s1
 parameters p1=4 p3=5 p4=6 // note: no "p2" here, p1 "redefined"
 r1 (1 0) resistor r=p1    // local definition used: value=4
 r2 (1 0) resistor r=p2    // inherit from parent(top-level) value=2
 r3 (1 0) resistor r=p3    // use local definition, value=5
 r4 (1 0) resistor r=p4    // use passed-in value, value=8
 r5 (1 0) resistor r=p1+p2/p3 // use local+inherited/local = (4+2/5) = 4.4
 ends
```
time_sweep tran start=0 stop=(p1+p2)\*50e-6 // use 5\*50e-6 = 150 us

dc_sweep dc param=p1 values=[0.5 1 +p2 (sqrt(p2*p2)) ] // sweep p1

Parameter Declaration:

Parameters can be declared anywhere in the top-level circuit description or on the first line of a subcircuit definition. Parameters must be declared before they are used (referenced). Multiple parameters can be declared on a single line. When parameters are declared in the top-level, their values must be specified. When parameters are declared within subcircuits, their default values are optionally specified.

Parameter Inheritance:

Subcircuit definitions inherit parameters from their parent (enclosing subcircuit definition, or top-level definition). This inheritance continues across all levels of nesting of subcircuit definitions, that is, if a subcircuit s1 is defined, which itself contains a nested subcircuit definition s2, then any parameters accessible within the scope of s1 are also accessible from within s2. Also, any parameters declared within the top-level circuit description are also accessible within both s1 and s2. However, any subcircuit definition can redefine a parameter that it has inherited. In this case, if no value is specified for the redefined parameter when the subcircuit is instantiated, then the redefined parameter uses the locally defined default value, rather than inheriting the actual parameter value from the parent.

Parameter Namespace:

Parameter names must not conflict with device or analysis instance names, that is, it is not possible to reference a parameter called $r1$ if there is an instance of a resistor (or other device or analysis) called $r1$. Parameter names must also not be used where a node name is expected.

Parameter Referencing:

Spectre netlist parameters can be referenced anywhere that a numeric value is normally specified on the right-hand side of an "=" sign or within a vector, where the vector itself is on the right-hand side of an "=" sign. This includes referencing of parameters in expressions (run spectre -h expressions for more details on netlist expression handling), as indicated in the preceding examples. You can use expressions containing parameter references when specifying device or analysis instance parameter values (for example specifying the resistance of a resistor or the stop time of a transient analysis, as outlined in the preceding example), when specifying model parameter values in model cards (for example specifying "bf=p1\*0.8" for a bipolar model parameter, bf), or when specifying initial conditions and nodesets for individual circuit nodes.

Altering/Sweeping Parameters:
Just as certain Spectre analyses (for example `sweep`, `alter`, `ac`, `dc`, `noise`, `sp`, `xf`) can sweep device instance or model parameters, they can also sweep netlist parameters. Run `spectre -h <analysis>` to see the particular details for any of these analyses, where `<analysis>` is the analysis of interest.

Temperature as a parameter:

You can use the reserved parameters `temp` and `tnom` anywhere that an expression can be used, including within expressions and user-defined functions. The `temp` parameter always represents the simulator (circuit) temperature, and `tnom` always represents the measurement temperature. All expressions involving `temp` or `tnom` are re-evaluated any time the circuit temperature or measurement temperature changes.

You can also alter or sweep the `temp` and `tnom` parameters using any of the techniques available for altering or sweeping netlist or subcircuit parameters (with the exception of `altergroups`).

This capability allows you to write temperature dependent models for example, by using `temp` in an equation for a model or instance parameter. For example,

```
r1 1 0 res r=(temp-tnom)*15+10k // temp is temperature
```

```
o1 options temp=55      // causes a change in above resistor r1
```

Reserved Parameters

The following parameters are reserved, and may not be declared as either top-level parameters or subcircuit parameters: `temp`, `tnom`, `scale`, `scalem`, `freq`, `time`.

### Parameter Set - Block of Data (paramset)

**Description**

A parameter set is a block of data, which can be referenced by a sweep analysis. Within a paramset the first row contains an array of top-level netlist parameters. All other rows contain numbers which are used to alter the value of the parameters during the sweep. Each row represents an iteration of the sweep. This data should be bound within braces. The opening brace is required at the end of the line defining the paramset. The paramset cannot be defined within subcircuits or cannot be nested.

**Definition**

```
<Name
```
Example:

data paramset {
    p1  p2  p3
    1.1 2.2 3.3
    4.4 5.5 6.6
}

**Output Selections (save)**

**Description**

The `save` statement indicates that the values of specific nodes or signals should be saved in the output file. It works in conjunction with the `save` parameter on most analyses. The output file is written in Cadence Waveform Storage Format (WSF), Cadence Parameter Storage Format (PSF) or in Nutmeg/SPICE3 format as controlled by a command line argument or a global option (see the options statement). The proper postprocessor should be used to view the output, generate plots, or do any further processing.

**Definition**

```plaintext
save <node|component|terminal
```

This statement takes a list of signals as an argument. The concept of nodes has been generalized to signals where a signal is a value associated with a topological node of the circuit or some other unknown that is solved by the simulator, such as the current through a inductor or the voltage of the internal node in a diode. Topological nodes can be either at the top-level or in a subcircuit.

For example,

```plaintext
save 7 out OpAmp1.comp M1:currents D3:oppoint L1:1 R4:pwr
```

which tells that node 7, node `out`, node `comp` in subcircuit `OpAmp1`, the currents through the terminals of `M1`, the `oppoint` information for diode `D3`, the current through the first terminal of `L1`, and the instantaneous power dissipated by `R4` should be saved. These outputs are saved in addition to any outputs specified with the `save` parameter on the analysis.
To specify a component terminal current, give the name of the component and the name or the index of the terminal separated by a colon. If currents is specified after the component and the colon, then all the terminal currents for the component are saved unless the component has only two terminals, in which case only the current through the first terminal is saved. Current is positive if it enters the terminal flowing into the component.

If a component name is followed by a colon and oppoint, then the operating point information associated with the component is computed and saved. If the colon is followed by an operating point parameter name (see each component for list of operating point parameters), then the value of that parameter is output.

If only a component name is given, all available information about the component, including the terminal currents and the operating point parameter values, is saved.

**Sensitivity Analyses (sens)**

**Description**

Use the sens control statement to find partial or normalized sensitivities of the output variables with respect to component and instance parameters for the list of the analyses performed. Currently DC and AC sensitivity analyses are supported. The results of the sensitivity analyses are stored in the output files written in Cadence Parameter Storage Format (PSF). The global option parameter senstype (see the options statement) is used to control the type of sensitivity being calculated. In addition, you can use +sensdata filename command line argument or a global option (see the options statement) to direct sensitivity analyses results into a specified ASCII file.

**Definition**

\[
\text{sens (output\_variables\_list) to (design\_parameters\_list) for (analyses\_list)}
\]

where

\[
\text{output\_variables\_list} = \text{ovar1 ovar2 ...}
\]

\[
\text{design\_parameters\_list} = \text{dpar1 dpar2 ...}
\]

\[
\text{analyses\_list} = \text{anal1 anal2 ...}
\]

The list of the design parameters may include valid instance and model parameters. You can also specify device instances or device models without a modifier. In this case Spectre will attempt to compute sensitivities with respect to all corresponding instance or model parameters. Caution should be exercised in using this option as warnings or errors may be
generated since many instance and model parameters cannot be modified. If no design parameters are specified then all the instance and model parameters are added. The list of the output variables for both AC and DC analyses may include node voltages and branch currents. For DC analyses, it also may include device instance operating point parameters.

Examples:

```
sens (q1:betadc 2 Out) to (vcc:dc nbjt1:rb) for (analDC)
```

For this statement DC sensitivities of betadc operating point parameter of transistor q1 and of nodes 2 and Out will be computed with respect to the dc voltage level of voltage source vcc and the model parameter rb for the DC analysis analDC. The results will be stored in the output file analDC.sens.dc.

```
sens (1 n2 7) to (q1:area nbjt1:rb) for (analAC)
```

For this statement AC sensitivities of nodes 1, n2, 7 will be computed with respect to the area parameter of transistor q1 and the model parameter rb for each frequency of the AC analysis analAC. The results will be stored in the output file analAC.sens.ac.

```
sens (1 n2 7) for (analAC)
```

For this statement AC sensitivities of nodes 1, n2, 7 will be computed with respect to all instance and model parameters of all devices in the design for each frequency of the AC analysis analAC. The results will be stored in the output file analAC.sens.ac.

```
sens (vbb:p q1:int_c q1:gm 7) to (q1:area nbjt1:rb) for (analDC1)
```

For this statement DC sensitivities of the branch current vbb:p, the operating point parameter gm of the transistor q1, the internal collector voltage q1:int_c and the node 7 voltage will be computed with respect to the instance parameter area for instance q1 and the model parameter rb for model nbjt1.

### SpectreHDL Usage and Language Summary (spectrehdl)

#### Description

SpectreHDL is a proprietary analog hardware description language. It allows analog circuit behavior to be described at a high level of abstraction, using a language which is similar to Verilog-A (run spectre -h veriloga for some details on the Verilog-A modeling language supported by Spectre). Behavioral descriptions of modules/components may be instantiated in a Spectre netlist along with regular Spectre primitives.
SpectreHDL descriptions are written in file(s) separate from the Spectre netlist file. These descriptions are written in modules (see the module alpha below). To include a module in the Spectre netlist, first add the line

```ahdl
ahdl_include "Ahdlfile.def"
```

to the Spectre netlist file (where Ahdlfile.def is the name of the file in which the required module is defined). The module is instantiated in the Spectre netlist in the same manner as Spectre primitives. For example,

```spec
name (node1 node2) alpha arg1=4.0 arg2=2 arg3="parameterized resistor"
```

This instantiates an element alpha, having two nodes and three parameters.

SpectreHDL modules can be debugged using `hdldebug`. `hdldebug` has a GUI and a command line mode. Please refer to the Verilog-A Debugging Tool User Guide for more information.

Module Template

The following is a SpectreHDL module template

```spec
module alpha( n1, n2 ) ( arg1, arg2, arg3 )

node [V,I] n1, n2;
parameter real arg1 = 2.0;
parameter integer arg2;
parameter string arg3;
{
    real local;
    // this is a comment
    initial {
        // initializations performed before the
        // start of an analysis.
    }
}
```

```
analog {
    // module behavioral description
    V(n1, n2) <- I(n1, n2) * arg1;
}
final {
    // tasks performed at the end of an analysis
}

Language Summary
The following provides a summary of the SpectreHDL analog hardware description language. For more information refer to the SpectreHDL Reference Manual.

Derivative and Integral Operators

dot(x)               Differentiate x wrt time.

integ(x <, ic <, assert>>) Integrate x wrt time. Output = ic during DC analysis. assert causes the integration to be reset.

idtmod(x, <ic <, modulus <, offset> > > )
          Circular Integration of x wrt time. Output = ic during DC analysis. Integration is performed with given offset and modulus if specified.

Built-In Mathematical Functions

abs(x)          Absolute value
exp(x)         Exponential if x < 80
ceil(x)         Smallest integer > x
floor(x)       Largest integer < x
ln(x)          Natural logarithm
log(x)         Log base 10
sqrt(x)        Square root
min(x,y)       Minimum
max(x,y)       Maximum
pow(x,y)       x to the power of y

Simulator Time-Step Control Functions
$\text{threshold}(x, \text{direction } <, \text{abstol } <, \text{reitol\_factor}>)$
   Set breakpoint when x crosses zero.
$\text{bound\_step}(\text{max\_step})$      Limit timestep, (timestep $\leq$ max_step).
$\text{break\_point}(\text{target } <, \text{period}>)$
   Set breakpoints at time = target and at times = N*period + target if period is specified.
$\text{last\_crossing}(x, \text{direction})$
   Return time when expression last crossed zero in a given direction.

Waveform Filter Functions
$\text{transition}(x <, \text{delay } <, \text{trise } <, \text{tfall}>>)$
   Specify details of signal transitions. For efficient simulation, it is recommended that x not be a continuous signal, i.e. a function of a probe. See the SpectreHDL manual for further explanation of this issue.
$\text{slew}(x <, \text{SRpos } <, \text{SRneg}>>)$
   Model slew rate behavior.
$\text{tdelay}(x, \text{time\_delay}, \text{max\_delay})$
   \[\text{Response}(t) = x(t - \text{time\_delay})\]
$\text{zdelay}(x,\ period,\ t\ transition,\ sample\ offset\ time,\ ic)$

Fixed period sample and hold function.

$\text{zi\_nd}(x,\ numer,\ denom,\ period,\ t\ transition,\ sample\ offset\ time)$

z-domain filter function, numerator-denominator form.

$\text{zi\_zd}(x,\ zeros,\ denom,\ period,\ t\ transition,\ sample\ offset\ time)$

z-domain filter function, zero-denominator form.

$\text{zi\_np}(x,\ numer,\ poles,\ period,\ t\ transition,\ sample\ offset\ time)$

z-domain filter function, numerator-pole form.

$\text{zi\_zp}(x,\ zeros,\ poles,\ period,\ t\ transition,\ sample\ offset\ time)$

z-domain filter function, zero-pole form.

$\text{laplace\_nd}(x,\ numer,\ denom,\ abstol)$

s-domain filter function, numerator-denominator form.

$\text{laplace\_zd}(x,\ zeros,\ denom,\ abstol)$

s-domain filter function, zero-denominator form.

$\text{laplace\_np}(x,\ numer,\ poles,\ abstol)$

s-domain filter function, numerator-pole form.

$\text{laplace\_zp}(x,\ zeros,\ poles,\ abstol)$

s-domain filter function, zero-pole form.

**Noise Functions**

$\text{white\_noise}(\ power,\ tag)$

Generates white noise with given power. Noise contributions with the same tag are combined for a module.

$\text{flicker\_noise}(\ power,\ exp,\ tag)$
Generates pink noise with given power at 1 Hz that varies in proportion to $1/f^{\text{exp}}$. Noise contributions with the same tag are combined for a module.

$\text{noise\_table( vector <, tag >) }$

Generates noise where power is determined by linear interpolation from the given vector of frequency-power pairs. Noise contributions with the same tag are combined for a module.

**AC Analysis Stimuli**

$\text{ac\_stim( <analysis\_name <, mag > > )}$

Small signal source of specified magnitude, active for given analysis.

**Interpolation Functions**

$\text{build\_table(type, response, inVec1, sizeVec1 <, inVec2, sizeVec2 ...>)}$

Build a table for B-Spline interpolation.

$\text{interpolate(interp\_table, v1<, v2 <, v3 <, v4 >>>)}$

Perform interpolation at given point.

**Simulator IO Functions**

$\text{strobe("format string" <, arguments>)}$

Print data to stdout every time step.

$\text{debug("format string" <, arguments>)}$

Print data to stdout every iteration.

$\text{fstrobe(fptr, "format string" <, arguments>)}$

Print data to a file every time step.

$\text{fdebug(fptr, "format string" <, arguments>)}$

Print data to a file every iteration.
$fread(fptr, "format string" <, arguments>)
    Read data from a file.
$cwarning("format string" <, arguments>)
    Warning message.
$cerror("format string" <, arguments>)
    Error message. Abort analysis.
$cfatal("format string" <, arguments>)
    Fatal message. Abort simulation.
$copen("filename", mode)     Open a file.
$fflush(fptr)                Flush a file to disk.
$fclose(fptr)                Close a file.
$popen( "command", "mode")   Open a pipe with given command in given mode.
$pclose(fptr)                Close a pipe.
$read_table("filename", table_id <, comment_str<, delim_str> > )
    Read from a file into a 2-D real array.
$write_table("filename", table_id <,delim_str> )
    Write from a 2-D real array to a file.
$halt( "string" )          Halt the simulation, printing given string.
$system( "system command" ) Sends a command to the operating system.
$str( "format_string" <,arg1 <,arg2 < ..etc> > )
    Create a string from arguments in given format.
$strcmp(str1, str2)          Compares two strings lexicographically.
$strtoint(int_as_str)        Converts a string, int_as_str, to an integer.
$strtoreal(real_as_str)      Converts a string, real_as_str, to a real.
$strcpy(des_str, src_str)    Copies src_str to des_src.
$strcat(des_str, src_str)    Appends src_str to des_src.
(strlenstr)                 Returns the number of characters in str.
$substr(input_str, start_pos, end_pos)
                     Returns the substring of input_str between start_pos and end_pos.
$strstr(input_str, sub_str)  Returns the first position where sub_str is found in input_str.
$strchr(input_str, character)
                     Returns the first position where character is found in input_str.
$strrchr(input_str, character)
                     Returns the last position where character is found in input_str.
$strspn(input_str, span_set)
                     Returns the number of continuous characters from the start of input_str that
                     are in span_set.
$strcspn(input_str, span_set)
                     Returns the number of continuous characters from the start of input_str that
                     are not in span_set.
$ascii(character)            Returns the ascii code of character.

**Simulator Environment Functions**

$time()                    Returns current simulation time.
$temp()                    Returns ambient simulation temperature.
$vt(<temp>)                 Returns thermal voltage. If temp is defined, returns the thermal voltage
                           at temp.
$analysis(analyses_string1<, analyses_string2 <, ...>)
                     Returns true(1) if the current analysis phase matches one of the given
                     analyses strings. Valid analyses strings are; "dc", "tran", "ac", "pss", "noise", "pdisto", "qpss",
Spectre Circuit Simulator Reference
Syntax

"pac", "pnoise", "pxf", "sp", "tdr", "xf", "envlp", "psp", "qpssp", "qpac", "qpnoise", "qpxf", "static", or "ic".

Simulator Tolerance Functions

$reltol() Returns relative tolerance.
$abstol(name) Returns absolute tolerance of quantity name.

Parameter Functions

$param_given(param) Returns 1 if param was set. param can be a model parameter or an instance parameter.
$pwr( x ) Assignment of model power consumption. Adds the expression x to the pwr parameter of a module.

Data Types

integer Discrete numerical type.
real Continuous numerical type.
string Text string type.
stream File pointer and text stream type.

enum { name1 <, name2 <, name3 <, ... >>> } Discrete name type.

void Null or empty type.
table Interpolation table type.

node [ PotentialName, FlowName ] Interconnection point type.
Data Qualifiers

parameter Indicates that a variable is a parameter and so may be given a different value when the module is instantiated and may have a range specifier.

const Indicates that a variable must be given a constant value when declared that can never be changed.

global Used only for internal nodes. Means that the internal node is an alias for a global node of the same name in the netlist.

Structural Statements

Structural statements are used inside the module block but outside the analog, final and initial blocks.

model module_or_primative new_model (<param1 = expr1 <..>>);

Used to create a model called new_model from module_or_primative.

module_or_primative inst_name (<node1 <..>> )(<param1 = expr1 <..>>);

Creates a new instance of module_or_primative called inst_name.

SpectreRF Summary (spectrerf)

Description

SpectreRF is an optional collection of analyses that are useful for circuits that are driven with a large periodic signal. Examples include mixers, oscillators, switched-capacitor filters, sample-and-holds, chopper stabilized amplifiers, frequency multipliers, frequency dividers, and samplers. They efficiently and directly compute the periodic and quasiperiodic steady-state solution of such circuits. They are capable of computing the large-signal and small-signal behavior, including noise behavior. Thus, SpectreRF is capable of computing the noise figure or intermodulation distortion of a mixer, the phase noise and harmonic distortion of an oscillator, and the frequency-response and noise behavior of a switched-capacitor filter. For more information on the SpectreRF analyses, run spectre -help analysisName where analysisName is pss, pac, pxf, pnoise, psp, qpss, qpac, qpxf, qpnoise, qpsp, or envlp.
Subcircuit Definitions (subckt)

Description

Hierarchical circuit description:

The `subckt` statement is used to define a subcircuit. Subcircuit definitions are simply circuit macros that can be expanded anywhere in the circuit any number of times. When an instance in your input file refers to a subcircuit definition, the instances specified within the subcircuit are inserted into the circuit. Subcircuits may be nested. Thus a subcircuit definition may contain instances of other subcircuits. Subcircuits may also contain component, analysis or model statements. Subcircuit definitions may also be nested, in which case the innermost subcircuit definition can only be referenced from within the subcircuit in which it is defined, and cannot be referenced from elsewhere.

Instances that instantiate a subcircuit definition are referred to as subcircuit calls. The node names (or numbers) specified in the subcircuit call are substituted, in order, for the node names given in the subcircuit definition. All instances that refer to a subcircuit definition must have the same number of nodes as are specified in the subcircuit definition and in the same order. Node names inside the subcircuit definition are strictly local unless declared otherwise in the input file with a global statement.

Subcircuit Parameters:

Parameter specification in subcircuit definitions is optional. In the case of nested subcircuit definitions, any parameters which have been declared for the outer subcircuit definition are also available within the inner subcircuit definition. Any parameters that are specified are referred to by name optionally followed by an equals sign and a default value. If, when making a subcircuit call, you do not specify a particular parameter, this default value is used in the macro expansion. Subcircuit parameters can be used in expressions within the subcircuit consisting of subcircuit parameters,

constants, and various mathematical operators. Run `spectre -h expressions` for more details on Spectre expression handling capability. Run `spectre -h parameters` for more details on how Spectre handles netlist parameters, including subcircuit parameters, and how they inherit within nested subcircuit definitions.

Subcircuits always have an implicitly defined parameter $\mathbf{m}$. This parameter is passed to all components in the subcircuit and each component is expected to multiply it by its own multiplicity factor. In this way, it is possible to efficiently model several copies of the subcircuit in parallel. It is an error to attempt to explicitly define $\mathbf{m}$ on a `parameters` line. Also, because $\mathbf{m}$ is only implicitly defined, it is not available for use in expressions in the subcircuit.
Inline Subcircuits:

An inline subckt is a special case of a subckt where one of the devices or models instantiated within this subckt does not get its full hierarchical name, but rather inherits the subckt call name itself. An inline subckt is syntactically denoted by the presence of the keyword "inline" before the "subckt". It is called in the same manner as a regular subcircuit. The body of the inline subcircuit can typically contain one of the following, corresponding to different use models:

* multiple device instances (one of which is the "inline" component)
* multiple device instances, (one of which is "inline")
  and one or more parameterized models
* a single "inline" device instance and a parameterized
  model to which the device instance refers

The "inline" component is denoted by giving it the same name as the inline subcircuit itself. When the subcircuit is flattened, the "inline" component does not take on a hierarchical name such as X1.M1, but rather takes on the name of the subckt call itself, such as X1. Any non-inline components in the subckt take on the regular hierarchical name, just as if the subcircuit were a regular subckt (i.e. not an inline subckt).

Probing the inline device

Spectre allows the following list of items to be saved or probed for primitive devices. These would also apply to devices modeled as the inline components of inline subcircuits:

1. all terminal currents e.g. "save q1:currents"
2. specific (index) terminal current e.g. "save q1:1"  (#1=collector)
3. specific (named) terminal current e.g. "save q1:b"  ("b"=base)
4. save all operating point info e.g. "save q1:oppoint"
5. save specific operating point info e.g. "save q1:vbe"
6. save all currents and oppoint info e.g. "save q1"

Parameterized Models and Inline Subckts:

Inline subckts can be used in the same way as regular subcircuits to implement parameterized models, however inline subckts provide some powerful new options. When an
inline subcircuit contains both a parameterized model and an inline device which references that model, then the user can create instances of the device, and each device will automatically get an appropriately scaled model assigned to it. For example, the instance parameters to an inline subckt could represent something like emitter width and length of a BJT device and within the subckt a model card could be created which is parameterized for emitter width and length and scales accordingly. When the designer instantiates the macro, he/she supplies the values for the emitter width and length, and a device is instantiated with an appropriate geometrically scaled model. Again, the inline device does not get a hierarchical name, and can be probed in the same manner as the inline device in the previous section on modeling parasitics, that is, it can be probed just as if it was a simple device, and not actually embedded in a subckt.

Automatic Model Selection using Inline Subckts:

See `spectre -h if` for a description on how to combine Spectre's `structural if` statement with inline subckts to perform automatic model selection based on any netlist/subckt parameter.

**Definition**

```plaintext
[inline] subckt <Name

Example 1: subckt

subckt coax (i1 o1 i2 o2)

    parameters zin=50 zout=50 vin=1 vout=1 len=0

    inner i1 o1 i2 o2  tline z0=zin vel=vin len=len

    outer o1 0 o2 0   tline z0=zout vel=vout len=len

ends coax
```

defines a parameterized coaxial transmission line macro from two ideal transmission lines. To instantiate this subcircuit, one could use an instance statement such as:

Coax1 pin nin out gnd coax zin=75 zout=150 len=35m

Example 2: inline subckt - parasitics

Consider the following example of an inline subcircuit, which contains a mosfet instance, and two parasitic capacitances:

```plaintext
inline subckt s1 (a b) // "s1" is name of subckt
```
parameters p1=1u p2=2u

  s1 (a b 0 0) mos_mod l=p1 w=p2  // "s1" is "inline" component
  cap1 (a 0) capacitor c=1n
  cap2 (b 0) capacitor c=1n

ends s1

The following circuit creates a simple mos device instance M1, and calls the inline subcircuit "s1" twice (M2 & M3)

M1 (2 1 0 0) mos_mod

M2 (5 6) s1 p1=6u p2=7u

M3 (6 7) s1

This expands/flattens to:

M1 (2 1 0 0) mos_mod

M2 (5 6 0 0) mos_mod l=6u w=7u  // the "inline" component, inherits call name

M2.cap1 (5 0) capacitor c=1n  // a regular hierarchical name

M2.cap2 (6 0) capacitor c=1n

M3 (6 7 0 0) mos_mod l=1u w=2u  // the "inline" component, inherits call name

M3.cap1 (6 0) capacitor c=1n

M3.cap2 (7 0) capacitor c=1n

Here the final flattened names of the three mosfets (one for each instance) are M1, M2 and M3, rather than M1, M2.s1 and M3.s1 as they would be if s1 was a regular subcircuit. The parasitic capacitors (which the user is not really interested in, or perhaps even aware of, if the inline subckt definition was written by a separate modeling engineer) have full hierarchical names however.

Example 3: inline subckt - scaled models

Consider the following example, in which a parameterized model is declared within an inline subcircuit for a bipolar transistor. The model parameters are the emitter width, length, and area, and also the temperature delta (trise) of the device above nominal. Ninety nine
instances of a 4*4 transistor are then placed, and one instance of a transistor with area=50 is placed. Each transistor gets an appropriately scaled model.

* declare a subckt, which instantiates a transistor with

* a parameterized model. The parameters are emitter width

* and length.

inline subckt bjtmod (c b e s)
parameters le=1u we=2u area=le*we trise=0
model mod1 bjt type=npn bf=100+(le+we)/2*(area/1e-12)
+       is=1e-12*(le/we)*(area/1e-12)
bjtmod (c b e s) mod1 trise=trise       // "inline" component
ends bjtmod

* some instances of this subckt
q1 (2 3 1 0) bjtmod le=4u we=4u         // trise defaults to zero
q2 (2 3 2 0) bjtmod le=4u we=4u trise=2
q3 (2 3 3 0) bjtmod le=4u we=4u
.
.
q99 (2 3 99 0) bjtmod le=4u we=4u
q100 (2 3 100 0) bjtmod le=1u area=50e-12

Verilog-A Usage and Language Summary (veriloga)

Description
Verilog-A is an analog hardware description language standard from Open Verilog International. It allows analog circuit behavior to be described at a high level of abstraction, using a language which is similar to SpectreHDL (run spectre -h spectrehdl for some
details on the SpectreHDL modeling language). Behavioral descriptions of modules/components may be instantiated in a Spectre netlist along with regular Spectre primitives.

Verilog-A descriptions are written in file(s) separate from the Spectre netlist file. These descriptions are written in modules (see the module alpha below). To include a module in the Spectre netlist, first add the line

```
ahdl_include "VerilogAfile.va"
```

to the Spectre netlist file (where VerilogAfile.va is the name of the file in which the required module is defined). The module is instantiated in the Spectre netlist in the same manner as Spectre primitives for example,

```
name (node1 node2) alpha arg1=4.0 arg2=2
```

This instantiates an element alpha, having two nodes and two parameters.

Verilog-A modules can be debugged using hdldebug. hdldebug has a GUI and a command line mode. Please refer to the Verilog-A Debugging Tool User Guide for more information.

In this release Verilog-A simulation performance has been improved by compiling the Verilog-A modules. This is explained in more detail in the Verilog-A compilation section below.

**Module Template**

The following is a Verilog-A module template

```verilog
include "discipline.h"
include "constants.h"
module alpha( n1, n2 );
electrical n1, n2;
parameter real arg1 = 2.0;
parameter integer arg2 = 0;
    real local;
    // this is a comment
    analog begin
```
Verilog-A compilation

In this release the simulation performance of Verilog-A has been improved by performing a one time compilation step. The performance improvement obtained is proportional to the complexity and amount of Verilog-A in your design. Following the initial compilation, recompilation will only be performed if the Verilog-A source is changed.

Verilog-A compilation is enabled by default. If you are making frequent changes to the Verilog-A used in your design, the overhead of the compilation step may become an issue. To turn off compilation set the CDS_AHDLCMI_ENABLE shell environment variable to NO e.g:

```
setenv CDS_AHDLCMI_ENABLE NO
```

To re-enable Verilog-A compilation set the CDS_AHDLCMI_ENABLE to YES e.g.

```
setenv CDS_AHDLCMI_ENABLE YES
```

or undefine the CDS_AHDLCMI_ENABLE environment variable e.g.

```
unsetenv CDS_AHDLCMI_ENABLE
```

The Verilog-A compiler stores the results of the compilation in a cache directory. A cache directory will be created per Verilog-A source file. By default the cache directory is located alongside the corresponding Verilog-A source file. If the corresponding Verilog-A source file is contained in a read only directory then the cache directory will be created in the current run directory.
directory. An alternative location can be specified through the CDS_AHDLCMI_DIR shell environment variable e.g.

```
setenv CDS_AHDLCMI_DIR /projects/ahdlcmiCacheDirs
```

By default the location specified by the CDS_AHDLCMI_DIR shell environment variable will only be used if the Verilog-A source file resides in a read only directory. To force the Verilog-A compiler to always use the CDS_AHDLCMI_DIR you can set the CDS_AHDLCMI_DIR_ALWAYS shell environment variable to YES.

**Language Summary**

The following provides a summary of the Verilog-A analog hardware description language. For more information refer to the Verilog-A Reference Manual.

**Analog Operators/Waveform Filters**

- `ddt(x, abstol)` Differentiate x wrt time.
- `idt(x, ic, assert, abstol)` Integrate x wrt time. Output = ic during dc analysis and when assert is 1.
- `idtmod(x, <ic, modulus, offset>)` Circular Integration of x wrt time. Output = ic during DC analysis. Integration is performed with given offset and modulus if specified.
- `transition(x, delay, trise, tfall)` Specify details of signal transitions. For efficient simulation, it is recommended that x not be a continuous signal, i.e. a function of a probe. See the Verilog-A manual for further explanation of this issue.
- `slew(x, SRpos, SRneg)` Model slew rate behavior.
- `delay(x, time_delay, max_delay)` Response(t) = x(t - time_delay).
- `zi_nd(x, numer, denom, period, < transition, sample offset time >)` z-domain filter function, numerator-denominator form.
zi_zd(x, zeros, denom, period, < ttransition <,sample offset time > )
  z-domain filter function, zero-denominator form.
zi_np(x, numer, poles, period, < ttransition <,sample offset time > )
  z-domain filter function, numerator-pole form.
zi_zp(x, zeros, poles, period, < ttransition <,sample offset time > )
  z-domain filter function, zero-pole form.
laplace_nd(x, numer, denom, <, abstol > )
  s-domain filter function, numerator-denominator form.
laplace_zd(x, zeros, denom, <, abstol > )
  s-domain filter function, zero-denominator form.
laplace_np(x, numer, poles, <, abstol > )
  s-domain filter function, numerator-pole form.
laplace_zp(x, zeros, poles, <, abstol > )
  s-domain filter function, zero-pole form.

**Built-In Mathematical Functions**

abs(x)                       Absolute value
exp(x)                       Exponential if x < 80
ln(x)                        Natural logarithm
log(x)                       Log base 10
sqrt(x)                      Square root
min(x,y)                     Minimum
max(x,y)                     Maximum
pow(x,y)                     x to the power of y
Noise Functions

white_noise( power <, tag > )

Generates white noise with given power. Noise contributions with the same tag are combined for a module.

flicker_noise( power, exp <, tag > )

Generates pink noise with given power at 1 Hz that varies in proportion to 1/f^exp. Noise contributions with the same tag are combined for a module.

noise_table( vector <, tag > )

Generates noise where power is determined by linear interpolation from the given vector of frequency-power pairs. Noise contributions with the same tag are combined for a module.

AC Analysis Stimuli

ac_stim( <analysis_name <, mag > > )

Small signal source of specified magnitude, active for given analysis.

Analog Events

Analog events must be contained in an analog event detection statement; @(analog_event) statement.

cross(x, direction <, timetol <, abstol >>)

Generates an event when x crosses zero.

above(x, <, timetol <, abstol >>)

Generates an event when x becomes greater than or equal to zero. An above event can be generated and detected during initialization. By contrast, a cross event can be generated and detected only after at least one transient time step is complete.

timer(start_time <, period> )

Set (optionally periodic) breakpoint event at time = start_time.

initial_step< ( arg1 <, arg2 <, etc... > > )
Generate an event at the initial step of an analysis. arg1, arg2, etc. may be any of: "dc", "tran", "ac", "pss", "noise", "pdisto", "qpss", "pac", "pnoise", "pxf", "sp", "tdr", "xf", "envelope", "sp", "qpsp", "qpac", "qpnoise", "qpfx", "static", or "ic".

`final_step< ( arg1 _, arg2 _, etc... _ ) >`

Generate an event at the final step of an analysis. arg1, arg2, etc. may be any of: "dc", "tran", "ac", "pss", "noise", "pdisto", "qpss", "pac", "pnoise", "pxf", "sp", "tdr", "xf", "envelope", "sp", "qpsp", "qpac", "qpnoise", "qpfx", "static", or "ic".

**Timestep Control**

`bound_step(max_step)` Limit timestep, (timestep <= max_step).

`last_crossing(x, direction)` Return time when expression last crossed zero in a given direction.

`discontinuity(n)` Hint to simulator that discontinuity is present in nth derivative.

**Simulator IO Functions**

`$display(argument_list)` Print data to stdout. Formatting strings may be interspersed between arguments/data.

`$fdisplay(fptr, argument_list)` Print data to a file. Formatting strings may be interspersed between arguments/data.

`$strobe(argument_list)` Print data to stdout. Formatting strings may be interspersed between arguments/data.

`$fstrobe(fptr, argument_list)` Print data to a file. Formatting strings may be interspersed between arguments/data.

`$fopen("filename")` Open a file for writing.

`$fclose(fptr)` Close a file.

`$finish<(n)>` Finish the simulation.

`$stop<(n)>` Stop the simulation.
Simulator Environment Functions

$\text{realtime}$ \quad Returns current simulation time.

$\text{temperature}$ \quad Returns ambient simulation temperature (K).

$\text{vt}$ \quad Returns thermal voltage.

$\text{vt}(\text{temp})$ \quad Returns thermal voltage at given temp.

$\text{analysis(analysis\_string1<, analysis\_string2 <, ...>>) }$

Returns true(1) if the current analysis phase matches one of the given analyses strings. Valid analyses strings are: "dc", "tran", "ac", "pss", "noise", "pdisto", "qpss", "pac", "pnoise", "pxf", "sp", "tdr", "xf", "envlp", "psp", "qpsp", "qpac", "qpnoise", "qpxf", "static", or "ic".

Parameter Functions

$\text{pwr( x )}$ \quad Assignment of model power consumption. Adds the expression x to the pwr parameter of a module.

Data Types

integer \quad Discrete numerical type.

real \quad Continuous numerical type.

Data Qualifiers

parameter \quad Indicates that a variable is a parameter and so may be given a different value when the module is instantiated, and that it may not be assigned a different value inside the module.

Structural Statements

Structural statements are used inside the module block but outside the analog block.

\text{module\_or\_primative #(\text{<.param1(expr1)<,...>>}) inst\_name (\text{<node1 <, ..>>}) ;}

Creates a new instance of module\_or\_primative called inst\_name.
References

This section gives additional details about the source documents referred to in the text.


References


