cadence™

Virtuoso® Spectre® Circuit Simulator Reference

Product Version 13.1.1
April 2014

© 2003–2013 Cadence Design Systems, Inc. All rights reserved.

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

MMSIM contains technology licensed from, and copyrighted by: C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh © 1979, J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson © 1988, J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling © 1990; University of Tennessee, Knoxville, TN and Oak Ridge National Laboratory, Oak Ridge, TN © 1992-1996; Brian Paul © 1999-2003; M. G. Johnson, Brisbane, Queensland, Australia © 1994; Kenneth S. Kundert and the University of California, 1111 Franklin St., Oakland, CA 94607-5200 © 1985-1988; Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304-1185 USA © 1994, Silicon Graphics Computer Systems, Inc., 1140 E. Arques Ave., Sunnyvale, CA 94085 © 1996-1997, Moscow Center for SPARC Technology, Moscow, Russia © 1997; Regents of the University of California, 1111 Franklin St., Oakland, CA 94607-5200 © 1990-1994, Sun Microsystems, Inc., 4150 Network Circle Santa Clara, CA 95054 USA © 1994-2000, Scriptics Corporation, and other parties © 1998-1999; Aladdin Enterprises, 35 Efal St., Kiryat Arye, Petach Tikva, Israel 49511 © 1999 and Jean-loup Gailly and Mark Adler © 1995-2005; RSA Security, Inc., 174 Middlesex Turnpike Bedford, MA 01730 © 2005.

All rights reserved. Associated third party license terms may be found at <install_dir>/doc/OpenSource/*

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

4
Other Simulation Topics 355

© 2003-2014 All Rights Reserved.

5
Circuit Checks

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

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. Verilog®-A uses 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:

© 2003-2014 All Rights Reserved.

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 *Cadence 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 *SpectreRF Simulation Option User Guide*.
- For more information about RF theory, see *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*[1].

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.

1. Kundert, Kenneth S. *The Designer's Guide to SPICE & Spectre*. Boston: Kluwer Academic Publishers, 1995.

© 2003-2014 All Rights Reserved.

|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 information.

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

1

Introducing the Virtuoso Spectre Circuit Simulator

This chapter discusses the following:

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.

© 2003-2014 All Rights Reserved.

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

© 2003-2014 All Rights Reserved.

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

© 2003-2014 All Rights Reserved.

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.

© 2003-2014 All Rights Reserved.

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.
- 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 address, see:

  http://support.cadence.com/wps/mypoc/cos?uri=deeplinkmin:COSHome

Analog HDLs

The Spectre circuit simulator works with Verilog®-A, an analog high-level description language. This language is part of the Spectre Verilog-A Simulation option, and is an open standard. The Verilog-A language is preferred because it is upward compatible with Verilog-AMS, a powerful and industry-standard mixed-signal language.

© 2003-2014 All Rights Reserved.

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

© 2003-2014 All Rights Reserved.

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

© 2003-2014 All Rights Reserved.

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 (non commensurate) 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.

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 system. 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.

© 2003-2014 All Rights Reserved.

2

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 28

Default Parameter Values on page 29

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

`spectre options` *`inputfile`*

If no options are specified, the Virtuoso® Spectre® circuit simulator saves the `.print` file in the current working directory, and saves the `.measure` and `.mt0` files in the `.raw` subdirectory of the netlist directory.

The 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 component 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 component 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`.

© 2003-2014 All Rights Reserved.

`-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`.
`-param`	Does not read the file containing the suggested parameter range limits. You can use `-p` as an abbreviation of `-param`.
`+param` *file*	Reads *file* for the suggested parameter range limits. You can use `+p` as an abbreviation of `+param`.
`-log`	Does not copy the output messages to a file. You can use `-l` as an abbreviation of `-log`.
`+log` *file*	Copies all messages to *file*. You can use `+l` as an abbreviation of `+log`.
`=log` *file*	Sends all messages to *file*. You can use `=l` as an abbreviation of `=log`.
`-raw` *raw*	Puts results in a file or directory named *raw*. In *raw*, `%C` is replaced by a circuit name. You can use `-r` as an abbreviation of `-raw`.
`-format` *fmt*	Produces raw data in the format *fmt*. You can use `-f` as an abbreviation of `-format`. Possible values for *fmt* are `nutbin`, `nutascii`, `wsfbin`, `wsfascii`, `psfbin`, `psfascii`, `psfbinf`, `psfxl`, `awb`, `sst2`, `fsdb`, `wdf`, `uwi`, or `tr0ascii`.
`+rtsf`	Enables the fast waveform viewing mode for psf output. Requires `-f psfbin, -f psfbinf` or `-f psfxl` format options.
`-outdir` *path*	Changes the default location of Spectre output files. It does not change the location of raw directory if explicitly specified with the `-raw` option, and of files that contain slashes in the name.
`-uwifmt` *name*	User defined output format. To specify multiple formats use `:` as a delimiter. This option is valid only when waveform format is defined as `uwi`.
`-uwilib` *lib*	Absolute path to the user-defined output format library. This option is used together with `-uwifmt`. Use `:` to specify more than one library.

© 2003-2014 All Rights Reserved.

`+checkpoint`	Turns on the checkpoint capability. You can use `+cp` as an abbreviation of `+checkpoint`.
`-checkpoint`	Turns off the checkpoint capability. You can use `-cp` as an abbreviation of `-checkpoint`.
`+savestate`	Turns on the savestate capability. You may use `+ss` as an abbreviation of `+savestate`.
`-savestate`	Turns off the savestate capability. You may use `-ss` as an abbreviation of `-savestate`.
`-recover`	Does not restart the simulation, even if a checkpoint file exists. You can use `-rec` as an abbreviation of `-recover`.
`+recover[=`*`filename`*`]`	Restarts the simulation from a checkpoint or savestate file. Savestate file will be used if both files exist. You can use `+rec[=`*`filename`*`]` as an abbreviation of `+recover[=`*`filename`*`]`.
`-cols` *`N`*	Sets screen width (in characters) to *`N`*. This is needed only if the simulator cannot determine screen width automatically, and if default value of 80 is not acceptable. Spectre cannot determine screen width if output is redirected to a file or a pipe. You can use `-c` as an abbreviation of `-cols`.
`-colslog` *`N`*	Sets the log-file width (in characters) to *`N`*. Defaults to 80.
`-%`*`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 *`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 on the screen.
`-warn`	Does not print warning messages on the screen.

© 2003-2014 All Rights Reserved.

`-maxwarns` *N*	Maximum number of times a particular type of warning message will be issued per analysis. You may use `-maxw` as an abbreviation of `-maxwarns`.
`-maxnotes` *N*	Maximum number of times a particular type of notice message will be issued per analysis. You can use `-maxn` as an abbreviation of `-maxnotes`.
`-maxwarnstolog` *N*	Maximum number of times a particular type of warning message will be printed to log file per analysis. You can use `-maxwtl` as an abbreviation of `-maxwarnstolog`.
`-maxnotestolog` *N*	Maximum number of times a particular type of notice message will be printed to log file per analysis. You may use `-maxntl` as an abbreviation of `-maxnotestolog`.
`+note`	Prints notices on the screen.
`-note`	Does not print notices on screen.
`+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 *cmd.*
`-slvhost <`*hostname*`>`	Runs the attached simulator on machine *hostname*. Defaults to local machine.
`-V`	Prints version information.
`-W`	Prints subversion information.
`-cmiversion`	Prints CMI version information.
`-cmiconfig` *file*	Reads *file* for information to modify the existing CMI configuration.
`-alias <`*name*`>`	Gives *name* to the license manager as the name of the simulator invoked.
`-E`	Runs the C preprocessor on an input file. In SPICE mode, the first line in the file must be a comment.
`-D<`*x*`>`	Defines string *x* and runs the C preprocessor.

© 2003-2014 All Rights Reserved.

`-D<x=y>`	Defines string *x* to be *y* and runs the C preprocessor.
`-U<x>`	Undefines string *x* and runs the C preprocessor.
`-I<dir>`	Runs the C preprocessor and searches the directory *dir* for include files.
`-spp`	Does not run the SPICE netlist reader on the input file.
`+spp`	Runs the Spice netlist reader on the input file.
`-sppbin path`	Specifies the location for SPICE netlist reader. Default provided.
`+sensdata <file>`	Sends the sensitivity analyses data to *file*.
`+mt`	Turns on the multithread capability. Virtuoso(R) Spectre automatically detects the number of processors and selects the proper number of threads to use. (See note on the `options` help page about using multithreading). `+multithread` can be used as an abbreviation of `+mt`.
+mt=*N*	Turns on the multithread capability. *N* is the specified number of threads. For the baseline mode, at most 4 threads are allowed. For APS mode, at most 16 threads are allowed. `+multithread` can be used as an abbreviation of `+mt`.
`-mt`	Turns off multithread capability. By default, multithreading is turned off for Spectre but turned on for APS. `-multithread` can be used as an abbreviation of `-mt`.
`-processor`	Sets the CPU affinity of a process similar to Linux `taskset` command. It specifies a numerical list of processors that may contain multiple items, separated by comma, for example, `-processor 0-3,5,7`. Specification of numerical value out of range for current system results in the process termination with *Invalid argument* error message. You can use `-proc` as an abbreviation of `-processor`.

© 2003-2014 All Rights Reserved.

`+dp[=rsh|ssh|lsf]`	This option enables distributed APS. It can only be used with the `+aps` option and is not supported in baseline Spectre. When distributing jobs on to other machines, three methods are supported: rsh, ssh, and lsf. When using the rsh or ssh methods, the `+hosts` option (see below) must be used to specify a list of machines, which are to be used. If no argument is provided to the `+dp` option, it will first check to see if the simulation is being controlled by LSF, and if so, use LSF to spawn any sub-processes. If LSF is not controlling the simulation, `+dp` will default to using rsh.
`+hosts` "*`host specification`*"	This option is used in conjunction with the `+dp` option above. When rsh or ssh are used to spawn sub-processes, the list of machines must be provided using this option. The format of the host specification string is the same for both rsh and ssh and contains a space-delimiter set of machine specifications. Each machine specification contains the name of the machine and number of cores to be used on the machine, separated by a colon, `:` . For example: `+hosts "hostA:4 hostB:4".` Specifies that 4 cores can be used for both hostA and hostB to perform simulation.
`-interactive`	Runs in the non-interactive mode, that is, process the input file and then return. You can use `-inter` as an abbreviation of `-interactive`.
`+interactive`	Runs in the default interactive mode. You can use `+inter` as an abbreviation of `+interactive`.
`+interactive=`*`type`*	Runs in the interactive mode of the type specified. You can use `+inter` as an abbreviation of `+interactive`. Possible values for *`type`* are `skill` or `mpsc`.
`+mpssession=`*`sessionName`*	The *`sessionName`* for an interactive session using multiprocess SKILL (MPS). This option is necessary for `+interactive=mpsc` and implies `+interactive=mpsc`.

© 2003-2014 All Rights Reserved.

+mpshost=*sessionHost*	The *sessionHost* for an interactive session using MPS.
-64	Run with 64 bit binary.
-mdlcontrol	Specifies 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.
-docheckl imit	Turns off the checklimit capability. You can use -docl as an abbreviation of -docheckl imit.
+docheckl imit	Turns on the checklimit capability. You can use +docl as an abbreviation of +docheckl imit.
+lqtimeout *value*	Turns on the queuing for license capability. Spectre will sleep and request the license again if no available license. You have to set how long to wait for a license (*value* is in seconds). Specifying *value* 0 means wait until license is available. You can use +lqt as an abbreviation of +lqtimeout.
+lqsleep *value*	Sleep time between two attempts 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.
+lqmmtoken	Turns on the queuing for token license capability. Spectre will register token request to license server and sleep to wait for the authorization. Given this option, Spectre will ignore all non-token licenses during waiting time since only token licenses are queued.

© 2003-2014 All Rights Reserved.

`+lsuspend`	Turns on the license suspend/resume capability. When Spectre receives SIGTSTP it will check in all the licenses before it gets suspended. The licenses will be checked out again when SIGCONT is received. You may use `+lsusp` as an abbreviation of `+lsuspend`.
`+lmode` *`value`*	Virtuoso(R) Spectre will check out the licenses required to run the specified license mode during initialization phase. Possible values are `#`, `RF` and `POWER`. String values are case insensitive. `#` is a numeric value, which means to check out the given token licenses (for example, 4 tokens for APS + multicore option) in one checkout step (such as, `+lmode 10`) and, similarly, `RF` and `POWER` means to check out enough licenses for `RF/power` option license in one step (for example, `+lmode rf`). `POWER` feature will be supported in future. If the simulation requires more licenses than that specified by `lmode`, the extra needed licenses will be checked out after the initialization phase.
`+lorder` *`value`*	Specifies value for license check out order. Possible values are `PRODUCT`, `MMSIM`, `PRODUCT:MMSIM` and `MMSIM:PRODUCT`. Values are case insensitive. `PRODUCT` will try to check out the product+options combination licenses only; `MMSIM` will try the `Virtuoso_Multi_mode_Simulation (MMSIM)` tokens only; `PRODUCT:MMSIM` will try product licenses first and then `MMSIM` tokens; `MMSIM:PRODUCT` will try `MMSIM` tokens first and then product licenses. Default is `PRODUCT:MMSIM`.
`-bsrccom` *`value`*	Determines the Bsource compiled flow. Use `0` for fast compilation but potentially slower simulation and `1` for slow compilation but potentially faster simulation. Default value is `1`. This option will be ignored in APS. You may use `-bc` as an abbreviation of `-bsrccom`.
`-ahdlcom` *`value`*	Determines the Ahdl compiled C flow. Use `0` for fast compilation but potentially slower simulation and `1` for slow compilation but potentially faster simulation. Default value is `1`. This option will be ignored in APS. You may use `-ac` as an abbreviation of `-ahdlcom`.

© 2003-2014 All Rights Reserved.

`-va,define MACRO[=`*`value`*`]`	The option defines a macro with priority higher than the one defined in Verilog-A files.
`+parasitics`	Enables reduction of netlist parasitics, with default bandwidth of 1 GHz.
`+parasitics=`*`value`*	Enables reduction of netlist parasitics and overwrites the default bandwidth. A value of 10 means 10 GHz. A value of `rf` is an alias of 30 GHz.
`++parasitics=value`	Enables reduction of netlist parasitics. Possible values are `liberal`, `moderate`, and `conservative` where `moderate` is the default value. This feature is available only in APS.
`+errpreset=`*`value`*	Selects a reasonable collection of parameter settings. Possible values are `liberal`, `moderate`, or `conservative`.
`+aps`	Enables APS mode.
`+aps=`*`value`*	Enables APS mode and overwrites errpreset in all transient analyses to the specified value. Possible values are `liberal`, `moderate`, or `conservative`.
`++aps`	Enables ++aps mode. Unlike the `+aps` mode, the ++aps mode uses a different time-step control algorithm as compared to Spectre. This results in improved performance, while satisfying error tolerances and constraints.
`++aps=`*`value`*	Enables the ++aps mode and overwrites the errpreset value in all transient analyses to apply the specified value. Possible values are `liberal`, `moderate`, or `conservative`.
`+xps`	Enables the XPS mode.
`+xps=s[value]`	Enables XPS SPICE accuracy mode. The optional value is the seed setting.
`+speed=value`	The simulation group speed setting. The lower number means conservative setting.

© 2003-2014 All Rights Reserved.

+query=*value*	This option queries the licenses that are required to run the simulation on the current machine or on one with similar configuration. Possible values are alllic and tokenlic. While alllic prints all possible license combinations for the design for simulation, tokenlic prints the number of MMSIM tokens required. Default is tokenlic.
+liclog	Writes the license check-in/check-out information in the log file.
+cktpreset=value	Sets a group of options for simulation, based on the circuit type.
-clearcache	Clear the Spectre cache directory /home/<*username*>/.cadence/mmsim. You can use -cc as an abbreviation for -clearcache.
-disableCPP	Disable CPP preprocesses in situations where Spectre implicitly calls CPP, such as -I and -D. However, if -E is specified, -disableCPP has no effect.

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 15, "Managing Files***,** in the *Virtuoso Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*.

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.

© 2003-2014 All Rights Reserved.

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 some defaults for different types of parameters from the Spectre online help:

`nf=1.0`	Forward emission coefficient.
`etchc`	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`.

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 Virtuoso Spectre Circuit Simulator and Accelerated Parallel SImulator User Guide.

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

3

Analysis Statements

This chapter discusses the following topics:

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

AC Analysis (ac)

Description

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

Spectre can perform AC 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 at each step. You can sweep the circuit temperature by giving the parameter name as `temp`, without a `dev` or `mod` parameter. In addition, you can sweep a netlist parameter by giving the parameter name without a `dev` or `mod` parameter. After the analysis is complete, the modified parameter returns to its original value.

Definition

```
Name ac parameter=value ...
```

Parameters

1	`prevoppoint=no`	Use the operating point computed in the previous analysis. Possible values are `no` and `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.

© 2003-2014 All Rights Reserved.

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 an estimate of DC solution (nodeset).
16	`write`	DC operating point output file at the first step of the sweep.
17	`writefinal`	DC operating point output file at the last step of the sweep.

Initial condition parameters

18	`force=none`	The set of initial conditions to use. Possible values are `none`, `node`, `dev`, and `all`.
19	`readforce`	File that contains initial conditions.
20	`skipdc=no`	Skip DC analysis. Possible values are `no` and `yes`.
21	`useprevic=no`	If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns. Possible values are `no`, `yes`, and `ns`.

Output parameters

22	`save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

© 2003-2014 All Rights Reserved.

23 `nestlvl` Levels of subcircuits to output.

24 `oppoint=no` Determine whether operating point information should be computed. If yes, where should it be printed (screen or file). Operating point information is not printed if the operating point computed in the previous analysis remains unchanged. Possible values are `no`, `screen`, `logfile`, and `rawfile`.

Convergence parameters

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

Annotation parameters

26 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, and `steps`.

27 `title` Analysis title.

28 `perturbation=linear`
The type of AC analysis. Default is linear for normal AC analysis. `im2ds` is for im2 distortion summary and `ds` is for distortion summary.
Possible values are `linear`, `ds`, `ip3`, `ip2`, `im2ds`, and `multiple_beat`.

29 `flin_out=0 Hz` Frequency of linear output signal.

30 `fim_out=0 Hz` Frequency of IM output signal.

31 `out1="NULL"` Output signal 1.

32 `out2="NULL"` Output signal 2.

33 `contriblist="NULL"` Array of device names for distortion summary. When `contriblist[""]`, distortion from each non-linear device is calculated.

34 `maxharm_nonlin=4` Maximum harmonics of input signal frequency induced by non-linear effect.

© 2003-2014 All Rights Reserved.

35 `rfmag=0` RF source magnitude.

36 `rfdbm=0` RF source dBm.

37 `rf1_src="NULL"` Array of RF1 source names for IP3/IP2/IM2DistortionSummary.

38 `rf2_src="NULL"` Array of RF2 source names for IP3/IP2/IM2DistortionSummary.

You can define sweep limits by specifying the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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 an operating point was computed during a previous analysis, you can set `prevoppoint=yes` to avoid recomputing it. For example, if `prevoppoint=yes` and the previous analysis was a transient analysis, the operating point is the state of the circuit at the final time point.

Nodesets help find the DC or initial transient solution. You can specify nodesets in the circuit description file with nodeset statements, or in a separate file by using the `readns` parameter. When nodesets are specified, Spectre computes an initial guess of the solution by performing DC analysis while forcing the specified values on to 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 required 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 required solution. 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 significantly speed up convergence.

When you simulate the same circuit multiple times, it is recommended that you use both `write` and `readns` parameters and assign the same file name to both parameters. DC analysis then converges quickly even if the circuit has changed since the last simulation, and the nodeset file is automatically updated.

During the initial operating point DC analysis, you may force some of the circuit variables to the values given in the `ic` file, `ic` statements, or `ic` parameter on the capacitors and inductors. The `ic` parameter controls the interaction of various methods of setting the force values. The effects of individual settings are as follows:

© 2003-2014 All Rights Reserved.

`force=none`: All initial conditions are ignored.

`force=node`: The `ic` statements are used, and the `ic` parameters 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 `ic` statements and `ic` parameters are used, with the `ic` parameters overriding the `ic` statements.

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

After you specify the initial conditions, Spectre computes the DC operating point 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 corresponding to each parameter name indicates where to find the description of that parameter.

© 2003-2014 All Rights Reserved.

```
lin  7              prevoppoint  1      span  5
```


© 2003-2014 All Rights Reserved.

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` without a `dev`, `mod`, or `sub` parameter.
- alter a top-level netlist parameter by giving the parameter name without a `dev`, `mod`, or `sub` parameter.
- 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` and `title`.
7	`reelaborate=yes`	Activates the re-elaboration process, which triggers a related expression evaluation throughout the circuit. To speed up the simulation, disable this parameter by setting it to `no` in

© 2003-2014 All Rights Reserved.

		successive `alter` statements. Possible values are `no` and `yes`.
8	`subckt`	Subcircuit definition.

Parameter Index

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

© 2003-2014 All Rights Reserved.

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, parameter statements and options statements (only supports temp, tnom, and scale). These statements should be bound within braces. The opening brace is required at the end of the line defining the altergroup. Altergroups cannot be specified within subcircuits. The following statements are not allowed within altergroups (analyses, `export`, `paramset`, `save`, and `sens`).

Within an altergroup, each device (instance or model) is first set to default and then the device parameters are updated. For netlist parameters, the expressions are updated and evaluated.

For subcircuit within altergroup, all instances of the subcircuits are modified when running altergroup. There are strict checks that do not allow changes to topology.

You can include files into the altergroup and can use the `simulator lang=spice` directive. See `spectre -h include` for more information. A model defined in the netlist should have the same model name and primitive type (`bsim2`, `bsim3`, `bjt`) in the altergroup. An instance defined in the netlist, should have the same instance name, terminal connections, and primitive type. For model groups, you can change the number of models in the group. However, 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` and `title`.

```
Example:
FastCorner altergroup {
        parameters p2=1 p3=p1+2
        myopt options temp=27
        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
```


© 2003-2014 All Rights Reserved.

```
}
```

The list of public devices supported by `altergroup` is as follows:

```
angelov  assert  bht  bht0
bjt  bjt301  bjt500  bjt500t
bjt503  bjt504  bjt504t  bjt3500
bjt3500t  bjtd504  bjtd504t  bjtd3500
bjtd3500t  bsim1  bsim2  bsim3
bsim3v3  bsim4  bsim6  bsimcmg
bsimimg  bsimsoi  capacitor  capq
cccs  ccvs  dcblock  dcfeed
delta_gate  dio500  diode  diode_cmc
ekv  ekv3  ekv3_nqs  ekv3_r4
ekv3_rf  ekv3_s  fracpole  gaas
hbt  hisim2  hisim_diode  hisim_hv
hisim_igbt  hvmos  igbt0  inductor
intcap  isource  jfet  jfet100
juncap  juncap200  juncap_eldo  ldmos
mos1  mos2  mos3  mos30
mos40  mos40t  mos705  mos902
mos903  mos903c  mos903e  mos903t
mos1000  mos1100  mos1100e  mos1101e
mos1101et  mos1102e  mos1102et  mos2001
mos2001e  mos2001et  mos2001t  mos2002
mos2002e  mos2002et  mos2002t  mos3002
mos3100  mos3100t  mos11010  mos11010t
mos11011  mos11011t  mos11020  mos11020t
mos11021  mos11021t  mosvar  msline
mutual_inductor  nodcap  ovcheck  pattern
pcccs  pccvs  phy_res  port
print  psitft  psp102  psp102e
psp103  psp103t  psp1020  psp1021
pspnqs102e  pspnqs103  pspnqs1020  pspnqs1021
pvccs  pvcvs  r2  r3
rcnet_opt  rdiff  resistor  rlck_matrix
spmos  tline  tom2  tom3
tom3v1  transformer  utsoi2  vbic
vccs  vcvs  vsource  wprobe
wsource
```

The list of public devices not supported by `altergroup` is as follows:

```
a2d  atft  b3soipd  bend
bend2  bit  cktrom  conductor
core  corner  cross  curve
d2a  delay  dielectric  ibis_buffer
iprobe  iswitch  loc  mos0
mos15  mtline  nport  rcnet_opt_C
rcnet_opt_R  relay  scccs  sccvs
stackup  step  svccs  svcvs
switch  tee  vswitch  winding
zcccs  zccvs  zvccs  zvcvs
```


© 2003-2014 All Rights Reserved.

Check Parameter Values (check)

Description

The `check` analysis checks the values of component parameters to ensure that 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`	The parameters that should be checked. Possible values are `none`, `inst`, `models`, `input`, `output`, `all`, and `oppoint`.

© 2003-2014 All Rights Reserved.

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 the `assert` statements in the netlist to perform checks on parameters of device instances, models, subcircuits or expressions.

Multiple `checklimit` analyses can be 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. The patterns can have wildcard symbols. Default is `all`.
2	`disable=[...]`	Array of checks to be disabled. The patterns can have wildcard symbols. Default is `none`.
3	`severity`	Severity of the checks. Possible values are `none`, `notice`, `warning`, `error`, and `fatal`.
4	`title`	Analysis title.
5	`checkallasserts=yes`	If all checks should be enabled or disabled. `checkallasserts=no` disables all checks. Possible values are `no` and `yes`.
6	`asserts=[...]`	Specifies a group of asserts whose parameters (specified using the `param` parameter) and values (specified using the `value` parameter) need to be changed.
7	`param`	Specifies the parameter setting for an individual assert or a group of asserts specified using the `asserts` parameter. Possible

© 2003-2014 All Rights Reserved.

values are `max`, `min`, `check_windows`, `maxvio_perinst` and `maxvio_all`.

8 `value` — Specifies the value of the parameter (specified using the `param` parameter) for the assert (specified using the `asserts` parameter) that needs to be changed.

Boundary parameters

9 `boundary_type=time` — Boundary type.
Possible values are `time` and `sweep`.

10 `start` — Start time or sweep boundary of the checks.

11 `stop` — Stop time or sweep boundary of the checks.

12 `check_windows=[...]` — Boundary time or sweep windows of the checks. Array should have an even number of values `[b_begin1 b_end1 b_begin2 b_end2 ...]`.

Parameter Index

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

`asserts` 6	`checkallasserts` 5	`param` 7	`stop` 11
`boundary_type` 9	`disable` 2	`severity` 3	`title` 4
`check_windows` 12	`enable` 1	`start` 10	`value` 8

© 2003-2014 All Rights Reserved.

Setting for Simulink-MATLAB co-simulation (cosim)

Description

Setting for Simulink-MATLAB co-simulation.

Definition

```
Name cosim parameter=value ...
```

Parameters

1	`server`	MATLAB/Simulink server name.
2	`port=38520`	Co-simulink listen port.
3	`timeout=60 s`	Socket timeout in seconds. Default is 60s.
4	`inputs=[...]`	Array of input names.
5	`outputs=[...]`	Array of output node names.
6	`design`	MATLAB/Simulink design name. If the design name is specified, MATLAB is launched automatically.
7	`silent=yes`	Launch MATLAB with parameter `-nodesktop`. Possible values are `no` and `yes`.
8	`ratio=0`	Hold time ratio between two sample points. Default is 0.

Parameter Index

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

`design 6`	`outputs 5`	`ratio 8`	`silent 7`
`inputs 4`	`port 2`	`server 1`	`timeout 3`

© 2003-2014 All Rights Reserved.

DC Analysis (dc)

Description

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` without a `dev`, `mod` or `sub` parameter.
- sweep a top-level netlist parameter by giving the parameter name without a `dev`, `mod` or `sub` parameter.
- 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 is complete, 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.

© 2003-2014 All Rights Reserved.

8	`log=50`	Number of steps, log sweep.
9	`values=[...]`	Array of sweep values.
10	`hysteresis=no`	Perform DC hysteresis sweep. When set to `yes`, a reverse sweep will automatically be added to the DC sweep. Possible values are `no` and `yes`.

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.

State-file parameters

14	`force=none`	Determine whether to force values for DC. Uses the values from the device and node ICs. Possible values are `none`, `node`, `dev`, and `all`.
15	`readns`	File that contains estimate of DC solution (nodeset).
16	`readforce`	File that contains force values.
17	`write`	File to which solution at first step in sweep is written.
18	`writefinal`	File to which solution at last step in sweep is written.
19	`useprevic=no`	If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns. Possible values are `no`, `yes`, and `ns`.

Output parameters

20	`save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.
21	`nestlvl`	Levels of subcircuits to output.

© 2003-2014 All Rights Reserved.

22 `print=no` — Print node voltages.
Possible values are `no` and `yes`.

23 `oppoint=no` — Should operating point information be computed; if yes, where should it be printed (screen or file). Operating point information is not printed if sweep parameter `param` is set.
Possible values are `no`, `screen`, `logfile`, and `rawfile`.

24 `check=yes` — Check operating point parameters against soft limits.
Possible values are `no` and `yes`.

Convergence parameters

25 `homotopy=all` — Method used when no convergence occurs on initial attempt of DC analysis. You can specify methods and their orders by specifying a vector setting such as `homotopy=[source ptran gmin]`.
Possible values are `none`, `gmin`, `source`, `dptran`, `ptran`, `arclength`, `all`, `fast_dptran`, `fast_ptran`, and `fast_gmin`.

26 `newton=normal` — You can specify newton methods such as `newton=none` and `newton=normal`.
Possible values are `none` and `normal`.

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

28 `maxiters=150` — Maximum number of iterations.

29 `maxsteps=10000` — Maximum number of steps used in homotopy method.

Annotation parameters

30 `annotate=sweep` — Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, `iters`, `detailed`, and `rejects`.

31 `title` — Analysis title.

© 2003-2014 All Rights Reserved.

Emir output parameters

32 `emirformat=none`	Format of the EM/IR database file. Possible values are `none` and `vavo`.
33 `emirfile`	Name of the EM/IR database file. The default is '%A_emir_vavo.db'. The file is printed to a raw directory.

You can define sweep limits by specifying the end points or the center value and span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. In addition, you can specify a step size parameter (`step`, `lin`, `log`, or `dec`) and determine whether the sweep is linear or logarithmic. If you do not specify 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 prints the linearized model for each nonlinear component.

Nodesets help find DC or initial transient solution. You can specify nodesets in the circuit description file with `nodeset` statements, or in a separate file using the `readns` parameter. When nodesets are specified, Spectre computes an initial guess of the solution by performing DC analysis while forcing the specified values on to 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 required 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 required solution. Second, these 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 significantly speed up convergence.

When you simulate the same circuit multiple times, it is recommended that you use both `write` and `readns` parameters and assign the same file name to both parameters. DC analysis then converges quickly even if the circuit has changed since the last simulation, and the nodeset file is automatically updated.

You can set the `force` parameter and specify the values to force the DC analysis. 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 as follows:

`force=none`: All initial conditions 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.

© 2003-2014 All Rights Reserved.

`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.

After you specify the force conditions, Spectre performs 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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

DC Device Matching Analysis (dcmatch)

Description

The DCMATCH analysis performs DC device mismatching analysis for a given output. It computes the deviation in the DC operating point of the circuit caused by mismatch in the devices. You 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 have zero mean and some variance. The variance of the current sources is computed according to mismatch models. Next, the 3-sigma variance of DC voltages or currents (due to the mismatch current sources) is computed at the outputs you specify. The simulation result displays the devices rank ordered by their contribution 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 that determine changes in accuracy with any changes in size. 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 is equal in magnitude but opposite in sign. Typical usage is to simulate the output offset voltage of operational amplifiers, estimate the variation in bandgap voltages, and predict the accuracy of current steering DACS.

DCMATCH analysis is available for BSIM3V3, BSIM4, BSIMSOI, EKV, PSP102, PSP103, BJT, VBIC, BHT, RESISTOR, PHY_RES, R3, and resistor-type bsource.

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`, and `rawfile`.
3	`file`	File name for results to be printed if where=file is used.

© 2003-2014 All Rights Reserved.

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.

© 2003-2014 All Rights Reserved.

State-file parameters

19 `readns`	File that contains estimate of DC solution (nodeset).
20 `useprevic=no`	If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns. Possible values are `no`, `yes`, and `ns`.

Output parameters

21 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.
22 `nestlvl`	Levels of subcircuits to output.
23 `oppoint=no`	Should operating point information be computed; if yes, where should it be printed (screen or file). Operating point information is not printed if (1) operating point is computed in the previous analysis and is unchanged, (2) sweep parameter `param` is set. Possible values are `no`, `screen`, `logfile`, and `rawfile`.

Convergence parameters

24 `prevoppoint=no`	Use operating point computed on the previous analysis. Possible values are `no` and `yes`.
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` and `yes`.

Annotation parameters

26 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, and `steps`.
27 `title`	Analysis title.

© 2003-2014 All Rights Reserved.

Miscellaneous parameters

28 `version=0`	Use BSIM short-channel mismatch equation for BSIM3 and BSIM4 devices, if the value is set to `1` and `3`. If set to `0` and `2`, do not use BSIM short-channel mismatch equation. Values `2` and `3` are compatible with Monte Carlo analysis while `0` and `1` are not. This option works only when `method=standard`. Possible values are `0` to `3`.
29 `method=standard`	Proceed with standard device mismatch models, or utilize the parameters defined in statistics blocks to compute output variation. Possible values are `standard` and `statistics`.
30 `nsigma=3`	Specifies the variation of each statistics parameter (in the number of sigma). This option works only if `method=statistics`.
31 `variations=all`	Selects the type of statistical parameters that are involved in `dcmatch`. The option is valid only if `method=statistics`. Possible values are `process`, `mismatch`, and `all`.

The dcmatch analysis will find a DC operating point first. If the DC analysis fails, the dcmatch analysis also fails. The parameter `mth` is a threshold value relative to maximum contribution. Any device contribution less than (`mth * maximum`) is not reported, where `maximum` is the maximum contribution among all the devices of a given type.

Examples:

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

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

© 2003-2014 All Rights Reserved.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Envelope Following Analysis (envlp)

Description

This analysis computes the envelope response of a circuit based on the specified analysis `clockname`, `period`, or `fund`. If `clockname` is specified, 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 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. The down conversion of two closely placed frequencies can also generate a slow-varying modulation envelope.

Envelope following 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. Therefore, autonomous circuits generate non-constant waveforms even though they are not driven by a time-varying stimulus. Driven circuits require time-varying stimulus to generate a time-varying response. The most common example of an autonomous circuit is an oscillator.

When applied to autonomous circuits, envelope following analysis requires you to specify a pair of nodes, `p` and `n`. In fact, this is how envelope following analysis determines whether it is being applied to an autonomous or a driven circuit. If the pair of nodes is supplied, envelope assumes the circuit is autonomous; if not, the circuit is assumed to be driven.

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 the specified harmonics of the clock fundamental.

Fast mode envelope analysis is used to simulate RF power amplifier (PA) with I/Q orthogonal modulation. Like normal envelope analysis, the time scale difference between I/Q signals and carrier is very large. Fast envelope analysis has larger speed up performance than normal envelope. Fast envelope has two modes, level1 and level2. Level1 is used to simulate the circuit without memory effect. Level2 is used to simulate the circuit with mild nonlinear-memory effect. If the circuit has strong nonlinear-memory effect, fast envelope can be inaccurate. In this situation, the only accurate way is to use regular envelope analysis. Fast envelope only outputs the fd result of specified nodes (assigned by the parameter output) at harmonic 1 of carrier.

© 2003-2014 All Rights Reserved.

Fast mode envelope analysis is not supported in the Shooting engine; its parameters apply only to the harmonic balance engine.

Definition

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

Parameters

Envelope fundamental parameters

1	`clockname`	Name of the clock fundamental.
2	`modulationbw (Hz)`	Modulation bandwidth.
3	`resolutionbw (Hz)`	Resolution bandwidth; if set, overwrites the stoptime to be at least `1/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 envelope starts shooting.
7	`period (s)`	Period of the clock fundamental; if set, `clockname` can be ignored. It is the estimated period for autonomous circuits.
8	`fund (Hz)`	Alternative to `period`. Frequency of the clock fundamental frequency.
9	`outputstart=start s`	Output is saved only after this time is reached.

Time-step parameters

10	`maxstep (s)`	Maximum time step for inner transient integration. Default is derived from `errpreset`.

© 2003-2014 All Rights Reserved.

11	`envmaxstep (s)`	Maximum outer envelope step size. Default is derived from `errpreset`.
12	`fixstepsize=no`	Use this option to fix envelope step size for speeding up envelope analysis. Possible values are `no` and `yes`.
13	`stepsize=4`	The number of cycles skipped between two steps when `fixstepsize` is `yes`. The time interval between the two steps will be `(stepsize+1)*Tc`, where `Tc` is the clock period. For shooting, autonomous, and fm envelope, it is rounded off to an integer.
14	`stepperiod`	The interval (in seconds of envelope following time) between two steps when `fixstepsize` is `yes`. Should be greater than period of clock. For autonomous, FM, or shooting envelope, it is rounded off to the nearest integer multiple of clock period.

Initial-condition parameters

15	`ic=all`	What should be used to set initial condition. Possible values are `dc`, `node`, `dev`, and `all`.
16	`skipdc=no`	If set to `yes`, there will be no DC analysis for initial transient. Possible values are `no` and `yes`.
17	`readic`	File that contains initial transient condition.
18	`useprevic=no`	If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns. Possible values are `no`, `yes` and `ns`.

Convergence parameters

19	`readns`	File that contains estimate of initial DC solution.
20	`cmin=0 F`	Minimum capacitance from each node to ground.

State-file parameters

21	`write`	File to which initial transient solution is to be written.

© 2003-2014 All Rights Reserved.

22 `writefinal` — File to which final transient solution is to be written.

23 `swapfile` — Temporary file that holds the matrix information used by Newton's method. It tells Spectre to use a regular file, rather than virtual memory, to hold the matrix information. Use this option if Spectre does not have enough memory to complete this analysis, This parameter is now valid only for shooting.

Envelope Integration method parameters

24 `envmethod=gear2only`
Envelope Integration method.
Possible values are `euler`, `trap`, `traponly`, `gear2`, `gear2only`, and `trapgear2`.

Integration method parameters

25 `method=gear2only` — Inner transient integration method.
Possible values are `euler`, `trap`, `traponly`, `gear2`, `gear2only`, and `trapgear2`.

26 `oscic=default` — Oscillator IC method. It determines how the starting values for the oscillator are determined.
Possible values are `default` and `lin`.

Accuracy parameters

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

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

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

30 `smoothcheck=yes` — When set to `no`, the smooth of envelope is not checked. The skip cycles are as many as possible. If `fixstepsize=yes`,

© 2003-2014 All Rights Reserved.

		`smoothcheck` is set to `no` automatically. Possible values are `no` and `yes`.
31	`itres=1e-2`	Relative tolerance for linear solver.
32	`lnsolver=gmres`	Linear solver. Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, and `gmres_cycle`.
33	`inexactNewton=no`	Inexact Newton method. Possible values are `no` and `yes`.
34	`steadyratio`	Ratio used to compute steady-state tolerances from LTE tolerance. Default is derived from `errpreset`.
35	`envlteratio`	Ratio used to compute envelope LTE tolerances. Default is derived from `errpreset`.

Annotation parameters

36	`annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, and `steps`.
37	`title`	Analysis title.

Output parameters

38	`harms`	If `harmonicbalance` is set to `no`, it is the number of clock harmonics to output and the default value is `1`. If `harmonicbalance` is set to `yes`, it is the `maxharm` of the clock fundamental and the default value is `3`.
39	`harmsvec=[...]`	Array of desired output clock harmonics. Alternative form of `harms` that allows selection of specific harmonics. For multi-carrier envelope, each group of elements with size equal to that of `funds` is a selection of specific harmonic combinations of fundamental frequencies.
40	`outputtype=both`	Output type. Possible values are `both`, `envelope` and `spectrum`.

© 2003-2014 All Rights Reserved.

41 `save` — Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

42 `nestlvl` — Levels of subcircuits to output.

43 `compression=no` — Perform data compression on output.
Possible values are `no` and `yes`.

44 `strobeperiod (s)` — The output strobe interval (in seconds) of envelope following time. For Shooting Envelope, the actual strobe interval is rounded off to an integer multiple of the clock period.

45 `transtrobeperiod (s)` — The output strobe interval (in seconds) of the envelope time. The value of the parameter must be less than the cycle period. Those strobe timepoints in all cycles will output when the parameter is working. It is valid for shooting and HB.

Newton parameters

46 `maxiters=5` — Maximum number of Newton iterations per transient integration time step.

47 `envmaxiters` — Maximum number of Newton iterations per envelope step. For time domain envelope, the default is 3. For Harmonic Balance Envelope, the default is 40.

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

Circuit age

49 `circuitage (Years)` — Stress time. Age of the circuit used to simulate hot-electron degradation of MOSFET and BSIM circuits.

50 `fmspeedup=0` — The level to speed up the envelope analysis for frequency modulated signal. Default is `0` for standard envelope following and `1` for fmmod sources speed up.

© 2003-2014 All Rights Reserved.

51 `saveinit=no` If set, the waveforms for the initial transient (tstab) before envelope are saved.
Possible values are `no` and `yes`.

Fast envelope parameters

52 `fastmode=off` This parameter controls the accuracy of envelope analysis. When fastmode=off, a rigorous transistor-level envelope algorithm is used . When fastmode=level1, a faster fast envelope algorithm is used. Fast envelope is more efficient but its usefulness is limited to (near)-memoryless systems. The level2 algorithm is obsolete and is automatically switched to level1 when set.
Possible values are `off`, `level1`, and `level2`.

53 `fastmethod=passband` This parameter applies only in fast envelope mode. The `passband` method models only magnitude-dependent (AM-AM and AM-PM) effects. The `baseband` method includes both magnitude- and phase- dependent (PM-AM and PM-PM) effects. Use the `passband` method when you model PAs. Use the `baseband` method when the circuit includes transistor-level modulators, demodulators, and generally whenever the output depends both on the magnitude and phase of the input signal.
Possible values are `passband` and `baseband`.

54 `writeenv` The file to which fast mode envelope data is written. It can be reused by readenv if the circuit remains unchanged, except for the decrease in srci or srcq scale. Can only be used in fast envelope.

55 `readenv` File from which fast mode envelope data is read. It can be used only if the circuit remains unchanged after the file is created, except for the decrease in srci or srcq scale. Can only be used in fast envelope.

56 `srci=[...]` I branch baseband modulation source, whose signal corresponds to the real part of baseband signal. If two sources are assigned, it means that the inputs are differential signals and the first source is positive. Only the pwl type of source is supported. Can only be used in fast envelope.

57 `srcq=[...]` Q branch baseband modulation source, whose signal corresponds to the imaginary part of baseband signal. If two

© 2003-2014 All Rights Reserved.

sources are assigned, it means that the inputs are differential signals and the first source is positive. Only the pwl type of source is supported and can be used only in fast envelope.

58 `srcw=[...]` Wireless source. Can be used only in fast envlp currently.

59 `sweepmethod=coarse` This parameter controls the sweep algorithm during the fast envelope circuit characterization. When sweepmethod is `coarse`, `fine`, or `userdefined`, it uses a fixed number of sweep points. When sweepmethod is `adaptive`, it determines the number of sweep points automatically. `coarse` uses 7 magnitude points (plus 12 phase points when fastmethod=baseband); `fine` uses 10 magnitude points (plus 16 phase points when fastmethod=baseband); when `sweepmethod=userdefined`, use the sweepnum parameter to define the grid. Possible values are `coarse`, `fine`, `userdefined`, and `adaptive`.

60 `sweepnum=[...]` This parameter controls the number of sweep points for fast envelope characterization when `sweepmethod=userdefined`. When fastmethod=passband, `sweepnum` is a one-element integer array which sets the number of magnitude sweep points. When fastmethod=baseband, it is a two-element integer array which sets the number of magnitude and phase sweep points. In passband mode, default `sweepnum`=[7]. In baseband mode, default `sweepnum`=[7 12]. In most cases, default values are adequate to capture the main and adjacent channel power accurately. Nevertheless, it is recommended to increase sweepnum gradually to ensure that the results do not change. A reasonable strategy is to start at sweepnum=[7 12] and increase to [10 16] until results converge. The upper limit of the magnitude and phase sweep points is [49 48].

61 `output=[...]` Fast mode envelope output nodes. Fast mode envelope only generates fd result (complex solution of a certain harmonic versus time) and can be used only in fast envelope.

62 `outputharmonic=1` Output harmonic in fast mode envelope analysis.

© 2003-2014 All Rights Reserved.

63 `outputallharms=no`

By default, fast mode envlp analysis outputs only one harmonic, as determined by the `outputharmonic` parameter. When `outputallharms` is set to `yes`, all harmonics are calculated and sent to output. In addition, when `outputallharms` is set to `yes` and the input excitation is single-tone, SpectreRF also calculates and stores the output time domain waveform. This parameter applies to `fast mode envlp` analysis only. Possible values are `no` and `yes`.

Fast envelope noise model parameters

64 `noisemethod=off` This parameter controls the noise model in fast envelope mode. `noisemethod` does not apply in regular envelope (when fastmode=off). By default, `noisemethod`=`off` and the circuit is treated as noiseless. If `noisemethod`=level1, the noise model is generated from linear noise analysis performed about the DC operating point. If `noisemethod`=level2, the noise model is calculated from a more rigorous periodic noise analysis. The root-mean-square value of the magnitude of the input signal and the mean of the phase of the input signal, are used as the operating point in periodic noise analysis. The level2 model is preferred in almost all the practical situations. The level1 model is faster to extract and may be useful in certain situations under nearly-linear operating condition. Possible values are `off`, `level1`, and `level2`.

65 `fstart=1k` This parameter sets the starting sweep frequency for the fast envelope noise sweep.

66 `dec=3` This parameter controls the noise frequency sweep for the fast envelope noise model. It is applicable when noisemethod=level1 or level2. When set, the simulator performs a linear or periodic noise analysis around output harmonic, using dec log steps per decade in the frequency interval given by [*`fstart, 1/(2*Tstep)`*]. If lin and dec are both specified, log sweep is used and the parameter lin is ignored.

67 `lin=10` This parameter controls the noise frequency sweep for the fast envelope noise model. It is applicable when `noisemethod`=level1 or level2. When set, the simulator performs a linear or periodic noise analysis around output

© 2003-2014 All Rights Reserved.

harmonic, using lin equal steps in the frequency interval given by [*fstart, 1/(2*Tstep)*].

Wireless fundamental parameters

68 `wsource` — Wireless source in envlp analysis. It must be selected within wsource instances.

69 `wprobe=[...]` — The wprobe vector in envlp analysis. It is designed to measure evm or ber, and so on.

Harmonic Balance Envelope parameters

70 `funds=[...]` — Array of fundamental frequency names for fundamentals that will be used for Harmonic Balance Envelope.

71 `maxharms=[...]` — Array of number of harmonics of each fundamental that will be considered for Harmonic Balance Envelope.

72 `freqdivide` — Large signal frequency division.

73 `fundfreqs=[...]` — Array of fundamental frequencies to use in multi-carrier envelope.

74 `harmonicbalance=no`
Use Harmonic Balance Envelope.
Possible values are `no` and `yes`.

75 `flexbalance=no` — The same parameter as `harmonicbalance`.
Possible values are `no` and `yes`.

76 `oversamplefactor=1`
Oversample sample device evaluations for Harmonic Balance Envelope.

77 `oversample=[...]` — Array of oversample factors for each tone for Harmonic Balance Envelope.

Tstab save/restart parameters

78 `saveperiod` — Save the tran analysis periodically on the simulation time.

© 2003-2014 All Rights Reserved.

79 `saveclock=1800 s` Save the tran analysis periodically on the wall clock time.

80 `savetime=[...]` Save the analysis states into files on the specified time points.

81 `savefile` Save the analysis states into the specified file.

82 `recover` Specify the file to be restored.

AMS-envlp co-sim parameters

83 `resetenv=no` Use this option to reset envelope data after D2A/A2D events for AMS-envlp co-simulation.
Possible values are `no` and `yes`.

84 `ignoredclk=no` Use this option to ignore digital clock if the clock rate is in the same order as envelope clock for AMS-envlp co-simulation.
Possible values are `no` and `yes`.

85 `trancycles=5` The number of transient cycles for AMS-envlp co-simulation. This is the number of cycles around D2A/A2D `events'` time point. Default value is `5`.

envlp-PAC parameters

86 `pacnames=[...]` Names of pac, pnoise, psp, or pxf analyses to be performed at each time point in the `pactimes` array. Not for AMS.

87 `pactimes=[...] s` Times when analyses specified in `pacnames` array are performed. Not for AMS.

If `period` or `fund` is not specified, 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 frequency is 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`. You provide an estimate of the modulation bandwidth. The simulator puts at least eight points within the modulation period. It is recommended that you use strobeperiod to get equally spaced envelope points, which will improve the noise floor in power spectrum density computation.

© 2003-2014 All Rights Reserved.

The `harms` and `harmsvec` parameters affect the simulation time in an 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. If `harmonicbalance` is no, `harms` is typically set to 1 or 2 because the high-order harmonics are not 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 certain envelope following analysis parameters is shown in the following table.

For `conservative` autonomous envelope, default values for `method` and `envmethod` are set to `traponly` to avoid numerical damping of the oscillator.

In this table, `T` is the period of the clock.

Table 3-1 Parameter defaults as a function of errpreset

errpreset	maxstep	envmaxstep	reltol	relref	steadyratio	envlteratio
liberal	T/20	Interval/10	0.01	siglobal	0.1	0.35
moderate	T/20	Interval/25	0.001	siglobal	0.1	3.5
conservative	T/50	Interval/50	0.0001	alllocal	1.0	35.0

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, because 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 two times the convergence criteria. 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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

© 2003-2014 All Rights Reserved.

Harmonic Balance Steady State Analysis (hb)

Description

This analysis uses harmonic balance (in the frequency domain) to compute the response of circuits with one fundamental frequency (periodic steady-state, PSS) or multiple fundamental frequencies (quasi-periodic steady-state, QPSS). The simulation time required for an HB analysis is independent of the time-constants of the circuit. This analysis also determines the circuit's periodic or quasi-periodic operating point, which can then be used during a periodic or quasi-periodic time-varying small-signal analysis, such as HBAC or HBNOISE.

Usually, harmonic balance (HB) analysis is a very efficient way to simulate weak nonlinear circuits. In addition, HB analysis works better than shooting analysis (in the time domain) for frequency-dependent components, such as delay, transmission line, and S-parameter data.

An HB analysis consists of two phases. The first phase calculates an initial solution, which the second phase then uses to compute the periodic or quasi-periodic steady-state solution, by using the Newton method.

The two most important parameters for HB analysis are `funds` and `maxharms`. The `funds` parameter accepts a list of names of fundamentals that are present in the sources. These names are specified in the sources by the `fundname` parameter. If only one name appears, the analysis is an HB PSS analysis. On the other hand, if more than one name appears, the analysis is an HB QPSS analysis. The `maxharms` parameter accepts a list of numbers of the harmonics that are required to adequately model the responses due to the different fundamentals.

The annotate parameter has two values specific to HB analysis: `detailed_hb` and `internal_hb`. When `annotate` is set to `detailed_hb` or `internal_hb`, additional analysis debug information is printed to the log files. In the case of `internal_hb`, encrypted debug information is stored in the internal log file. Both options are valid for pss and qpss analyses with `flexbalance=yes`.

Definition

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


© 2003-2014 All Rights Reserved.

Parameters

HB fundamental parameters

1	`funds=[...]`	Array of fundamental frequency names for fundamentals to use in analysis.
2	`fundfreqs=[...]`	Array of fundamental frequencies to use in analysis.
3	`maxharms=[...]`	Array of number of harmonics of each fundamental to consider for each fundamental.
4	`selectharm`	Name of harmonics selection methods. Default is `diamond` when `maximorder` or `boundary` is set; otherwise, default is `box`. Possible values are `box`, `diamond`, `funnel`, and `axis`.
5	`evenodd=[...]`	Array of even, odd, or all strings for moderate tones to select harmonics.
6	`maximorder`	Maximum intermodulation order (same parameter as `boundary`).
7	`freqdivide`	Large signal frequency division.

Simulation interval parameters

8	`tstab=0.0 s`	Extra stabilization time after the onset of periodicity for independent sources.
9	`autotstab=no`	Activates the automatic initial transient (tstab) in harmonic balance. If `yes`, the simulator decides whether to run tstab and for how long. Typically, the initial length of tstab is 50 periods, however, it may be longer depending on the type of circuit and its behavior. If steady-state is reached (or nearly reached), tstab terminates early. Possible values are `no` and `yes`.
10	`autosteady=no`	Activates the automatic steady state detection during initial transient (tstab) in harmonic balance. When steady state is reached (or nearly reached), tstab terminates early. This

© 2003-2014 All Rights Reserved.

parameter applies only when tstab>0 or when `autotstab=yes`. Possible values are `no` and `yes`.

11 `autoharms=no` — Activates automatic harmonic number calculation in harmonic balance. Applies only if tstab>0 or if `autotstab=yes`. If a steady-state is reached, Spectre does a spectrum analysis to calculate the optimal number of harmonics for HB. The minimum number of harmonics is specified by `maxharms`. If steady-state is not reached to sufficient tolerance, `autoharms` may be disabled.
Possible values are `no` and `yes`.

Time-step parameters

12 `maxstep (s)` — Maximum time step. The default is derived from `errpreset`.

Initial-condition parameters

13 `ic=all` — The value to be used to set initial condition.
Possible values are `dc`, `node`, `dev`, and `all`.

14 `skipdc=no` — If set to `yes`, there is no DC analysis for initial transient.
Possible values are `no`, `yes` and `sigrampup`.

15 `readic` — File that contains initial condition.

16 `oscic=default` — Oscillator IC method. It determines how the starting values for the oscillator are calculated. `oscic=lin` provides you an accurate initial value, but it takes time; `oscic=fastic` is fast, but it is less accurate. `oscic=skip` directly uses the user-provided frequency as the initial guess frequency. It is for two tier-method only.
Possible values are `default`, `lin`, `fastic`, and `skip`.

17 `useprevic=no` — If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns.
Possible values are `no`, `yes` and `ns`.

Convergence parameters

18 `readns` — File that contains an estimate of the initial transient solution.

© 2003-2014 All Rights Reserved.

19 `cmin=0 F` — Minimum capacitance from each node to ground.

Output parameters

20 `save` — Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

21 `nestlvl` — Levels of subcircuits to output.

22 `saveinit=no` — If set to `yes`, the waveforms for the initial transient before steady state are saved.
Possible values are `no` and `yes`.

Integration method parameters

23 `tstabmethod` — Integration method used in stabilization time. The default is `traponly` for autonomous circuits, or is derived from `errpreset` for driven circuits.
Possible values are `euler`, `trap`, `traponly`, `gear2`, and `gear2only`.

Accuracy parameters

24 `errpreset` — Selects a reasonable collection of parameter settings.
Possible values are `liberal`, `moderate`, and `conservative`.

25 `maxperiods` — Maximum number of iterations allowed before convergence is reached in shooting or harmonic balance Newton iteration. For PSS and QPSS, the default is 20 for driven circuits, and 50 for oscillators. For HB, the default is 100.

26 `itres=1e-4 for shooting, 0.9 for HB` — Controls the residual for iterative solution of linearized matrix equation at each Newton iteration. Tightening the parameter can help with the Newton convergence, but does not affect the result accuracy. The value should be between [0, 1].

27 `krylov_size=10` — The minimum iteration count of the linear matrix solver used in HB large-signal analysis. After reaching `krylov_size` iterations, the iteration is forced to terminate because of the poor

© 2003-2014 All Rights Reserved.

rate of convergence. Increase `krylov_size` if the simulation reports insufficient norm reduction errors in GMRES.

28 `pinnode` — Node to pin during autonomous HB simulation.

29 `pinnoderank` — Harmonic rank to pin during autonomous HB simulation.

30 `pinnodemag` — This parameter gives an estimate of the magnitude of the pin node voltage. Default value is 0.01.

31 `pinnodeminus` — Second node to pin during autonomous HB simulation. Needed only when differential nodes exist in oscillator.

32 `hbpartition_defs=[...]`
Define HB partitions.

33 `hbpartition_fundratios=[...]`
Specify HB partition fundamental frequency ratios.

34 `hbpartition_harms=[...]`
Specify HB partition harmonics.

35 `oversamplefactor=1`
Oversample device evaluations.

36 `oversample=[...]` — Array of oversample factors for each tone. This parameter overrides `oversamplefactor`.

37 `oscmethod` — Osc Newton method for autonomous HB.

38 `hbhomotopy=tone` — Name of Harmonic Balance homotopy selection methods. Possible values are `tstab`, `source`, `gsweep`, `tone`, and `inctone`.

39 `sweepic=none` — IC extrapolation method in sweep HB analysis. Possible values are `none`, `linear` and `log`.

40 `gstart=1.e-7` — Start conductance for hbhomotopy of gsweep.

41 `gstop=1.e-12` — Stop conductance for hbhomotopy of gsweep.

42 `glog=5` — Number of steps, log sweep for hbhomotopy of gsweep.

© 2003-2014 All Rights Reserved.

Annotation parameters

43 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `estimated`, `steps`, `iters`, `detailed`, `rejects`, `alliters`, `detailed_hb`, and `internal_hb`.

44 `title` Analysis title.

Newton parameters

45 `restart=no` Restart the DC/PSS/QPSS solution if set to `yes`; if set to `no`, reuse the previous solution as an initial guess; if set to `firstonly`, restart if it is the first point of sweep (supported only in HB). The default value is `no` for HB and `yes` for shooting.
Possible values are `no`, `yes` and `firstonly`.

Circuit age

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

47 `writehb` File to which final harmonic balance steady-state solution is to be written. Small-signal analyses, such as hbac and hbnoise can read in the steady-state solution from this file directly instead of running the hb analysis again.

48 `readhb` File from which final harmonic steady-state solution is to be read. Small signal analyses, such as hbac and hbnoise can read in the steady-state solution from this file directly instead of running the hb analysis again.

Tstab save/restart parameters

49 `saveperiod` Save the tran analysis periodically on the simulation time.

50 `saveperiodhistory=no`
Maintains the history of saved files. If `yes`, stores all the saved files. Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

51 `saveclock (s)` Save the tran analysis periodically on the wall clock time. The default is `1800s` for Spectre. This parameter is disabled in the APS mode by default.

52 `savetime=[...]` Save the analysis states into files on the specified time points.

53 `savefile` Save the analysis states into the specified file.

54 `recover` Specify the file to be restored.

55 `xdbcompression="no"` Sets the automatic gain compression analysis. In automatic gain compression analysis, Spectre automatically sweeps the input excitation until the gain, as defined by the analysis parameter xdbgain, compresses by the amount specified by the analysis parameter xdblevel. In gain compression analysis, Spectre ouputs the hb solution at the calculated compression point only. Dependent analyses, such as hbnoise and hbac, are supported and calculated about the calculated compression level. Auxiliary output includes the gain and voltage/power compression curves. These outputs are available for analysis and post-processing in ADE. The possible values are `yes` and `no`. Default is `no`.

56 `xdblevel=1.0` Sets the gain compression level for compression analysis. The reference point for gain compression is the small-signal gain of the circuits, or as specified by the analysis parameter `xdbref`. Default is `1`.

57 `xdbgain="power"` Chooses between the voltage gain or transducer power gain as the target for compression point calculation. When `xdbgain=power`, the gain is defined as *G (dB) = Pload (dBm) - Pavailable (dBm)*. When `xdbgain=voltage`, the gain is defined as *G (dB) = dB20(|Vload|/|Vsource|)*. In both cases, Spectre sweeps the excitation source until *xdbref - G = xdblevel*, where the analysis parameter xdbref defines the reference level for compression calculation. Possible values are `power` and `voltage`. Default is `power`.

58 `xdbref="linear"` Sets the reference point for gain compression calculations. When `xdbref=linear`, spectre uses the small-signal gain as the reference. When `xdbref=max`, spectre uses the maximum

© 2003-2014 All Rights Reserved.

observed gain as the reference. Possible values are `linear` and `max`. Default is `linear`.

59 `xdbsource` — The instance name of the excitation source, which is swept automatically to reach the compression level. When `xdbgain=power`, the excitation source must be a port instance. When `xdbgain=voltage`, the excitation source can be a vsource instance or a port instance. Note that in the voltage gain calculation, *`dB20(|Vload|/|Vsource|)`*, when `xdbsource` is a port and `xdbgain=voltage`, Spectre interprets `Vsource` as the voltage across a matched load, according to the Spectre usual port element conventions.

60 `xdbload` — The instance name of the load termination. When `xdbgain` is `power`, `xdbload` can be a port, a resistor, or a current probe.

61 `xdbnodep` — The output terminals for voltage gain calculation when `xdbgain=voltage`. If either is left unspecified, the terminal is assumed to be the global ground.

62 `xdbnoden` — The output terminals for voltage gain calculation when `xdbgain=voltage`. If either is left unspecified, the terminal is assumed to be the global ground.

63 `xdbrefnode` — The reference node when `xdbload` is a current probe. The default is the ground node.

64 `xdbharm=[...]` — The Integer array which specifies the harmonic indexes of the output voltage or power component.

65 `xdbsteps=100` — The maximum number of steps for the compression point search. The simulator terminates if `xdbsteps` exceeds before the compression point is found. The default is `100`.

66 `xdbmax` — The maximum input power (or voltage) for the compression point search. Default is `10 dBm` when `xdbgain=power`, and `0.1 V` when `xdbgain=voltage`.

67 `xdbstart` — The starting input power (or voltage) for the compression point search. Default is `(xdbmax-50) dBm` when `xdbgain=power`, and `xdbmax/1000` when `xdbgain=voltage`.

© 2003-2014 All Rights Reserved.

68 `xdbtol=0.01` Sets the tolerance for compression analysis. This tolerance is used in compression curve fitting and calculating the compression point.

69 `xdbrapid="no"` Sets the automatic gain compression analysis in rapid mode. In this mode, Spectre does not trace the compression curve and calculates only the compression point.

70 `xdbcpi` Sets the estimated input-referred compression point for rapid compression analysis.

71 `memoryestimate="no"`

Sets the memory usage estimate for Harmonic Balance. If `yes`, a memory estimate is printed in the log file. You can use this memory estimate to plan the computing resources before submitting harmonic balance runs. In memory estimate mode, a short simulation is performed first, and the engine exits after printing the estimate in the log file without saving harmonic balance results. You must turn it off to perform an actual simulation. Memory estimation is not recommended for simulations that require less than 500MB approximately. For PSS analysis, memory estimate mode does not apply unless `flexbalance=yes`. The estimate applies only to large-signal analysis, and does not include subsequent noise or other small-signal simulations.

72 `tuneparam` When set, `tuneparam` enables the tuning mode oscillator analysis. In the tuning mode analysis, a circuit parameter is automatically varied to reach the oscillation frequency specified by the fundfreqs parameter. The tuning parameter can be a device instance parameter (as determined by the parameters `tunedev`) or a netlist parameter. This mode applies only to autonomous circuits (oscillators).

73 `tunedev` Sets the instance name of a device whose parameter (identified by `tuneparam)` will be varied such that the circuit oscillates at the specified frequency. Applies only in tuning mode autonomous analysis. `Tunedev` must be used with `tuneparam`.

74 `tunerange=[...]` The tuning range of the parameter identified by `tuneparam`. Although `tunerange` is not required, it can aid in convergence, if set.

© 2003-2014 All Rights Reserved.

75 `lsspports=[...]` Specifies the list of ports on which the large-signal 2-port S-parameters are calculated.

76 `lsspharms=[...]` Specifies the output harmonic for large-signal S-parameter calculations. The input harmonic is defined by the frequency parameters on the input port instance.

77 `lsspfile` Identifies the file name for large-signal S-parameter output.

78 `lsspdatafmt=touchtone`
Sets the file format of the large-signal S-parameter output. Possible values are `spectre` and `touchstone`. Default is `touchstone`.

79 `lsspdatatype=magphase`
Sets the data format or the large-signal S-parameter output. Possible values are `realimag`, `magphase`, and `phase`. Default is `magphase`.

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. The initial transient simulation also helps convergence by eliminating the large but fast decaying modes that are present in many circuits.

In some circuits, the linearity of the relationship between the initial and final states depends on when HB analysis begins. In practice, starting at a good point can improve convergence, and starting at a bad point can degrade convergence and slow down the analysis.

When HB analysis simulates oscillators, initialization is performed to obtain an initial guess of the steady-state solution and of the oscillating frequency. Two initialization methods are implemented, based on transient and linear analysis. When `oscic=default` is specified, transient initialization is used and the length of the transient is specified by `tstab`. You must start the oscillator by using initial conditions or by using a brief impulsive stimulus, just as you would if you were simulating the turn-on transient of the oscillator by using transient analysis. Initial conditions would be provided for the components of the oscillator's 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. The Designers Guide to Spice and Spectre [K. S. Kundert, Kluwer Academic Publishers, 1995] describes in depth some techniques for starting oscillators. When `oscic=lin` is specified, linear initialization is used. In this method both oscillation frequency and amplitude are estimated based on linear analysis at DC solution. No impulsive stimulus or initial conditions are needed. Linear initialization is suitable for linear type of oscillators, such as LC and crystal oscillators. Note that tstab transient is still performed after

© 2003-2014 All Rights Reserved.

linear initialization, though it can be significantly shortened or skipped. Either way, specifying a non-zero `tstab` parameter can improve convergence.

For the funds parameter, 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 source. However, these sources must have the same fundamental name. For each fundamental name, the fundamental frequency is the greatest common factor of all frequencies associated with the name. Omitting a fundamental name in the `funds` parameter is an error that stops the simulation. If `maxharms` is not given, a warning message is issued, and the number of harmonics defaults to `1` for each of the fundamentals in multi-tone simulation and `10` in single-tone simulation.

HB signal partition is a method of decomposing a circuit so that multi-rate behavior can be exploited to increase simulation performance. If every part of a circuit has the same spectrum structure, such as fundamental frequency and bandwidth, there is no need to apply signal partition. However, if the RF circuit has multiple tones, the signals in different parts can have various spectrum structures, such as different fundamental frequency and number of harmonics. With HB signal partition, you can divide the circuit into several parts based on the signals contained in them. The parameter `hbpartition_defs` defines the partitions. Each partition can be made up of one or more instances. For example,

```
hbpartition_defs = ["I9 I10" "I11 I12" "I13 I14"]
```

defines three partitions. The first partition consists of instance "`I9`" and "`I10`" while the second partition consists of instances "`I11`" and "`I12`".The third one has "`I13`" and "`I14`".

The number of instances for each partition should not be less than 1 and there is no upper limit for the number.

The principle for dividing a circuit is that the subcircuits or instances with the same spectrum properties should be put into one partition. The parameter `hbpartition_harms` specifies the maximum number of positive harmonics of each tone for every partition. For example:

```
hbpartition_harms=["10 0 0" "5 3 3" "3 3 3"]
```

So, the maximum number of positive harmonics for the first partition is `10`, `0` and `0`, respectively. For the second partition, it is `5`, `3` and `3`. And, for the last partition, it is `3`, `3` and `3`.

The parameter `hbpartition_fundratios` indicates the fundamental frequency ratio of each tone for each partition. With these ratios, it is easy to know the fundamental frequencies of each tone of the partitions. For example:

```
hbpartition_fundratios=["2 1 1" "1 1 1" "1 1 1"]
```

If three global fundamental frequencies are defined as: `LO=1GHz`, `RF1=1.1GHz` and `RF2=1.13GHz`, it indicates the fundamental frequencies of each tone of the first partition is `2*LO`, `1*RF1`, and `1*RF2`, respectively. The second and third partition has the same

© 2003-2014 All Rights Reserved.

frequencies for their each tone: `1*LO`, `1*RF1` and `1*RF2`. However, you have to make sure that the global fundamental frequencies for each tone are the smallest among all the partitions and the ratios are integers.

The parameter `maxperiods` default value is set to 50 for HB.

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. If you want a fast simulation with reasonable accuracy, you can set `errpreset` to `liberal` (it is not recommended to use `liberal` on RF circuit). If you have some concern for accuracy, you can set `errpreset` to `moderate`. If accuracy is your main interest, you can set errpreset to `conservative.`

The following table shows the effect of `errpreset` on other parameters in HB with One-tone Driven Circuits, Multi-tone Driven Circuits and Autonomous Circuits:

```
-------------------------------------------------------------------------------
Parameter defaults as a function of errpreset with different circuits
-------------------------------------------------------------------------------
| One-tone Driven Circuits | Multi-tone Driven Circuits and Autonomous Circuits
-------------------------------------------------------------------------------
errpreset    |  reltol(max)  lteratio   |  reltol(max)  lteratio
-------------------------------------------------------------------------------
----------
liberal      |  1e-3         3.5        |  1e-3         3.5
moderate     |  1e-3         3.5        |  1e-4         3.5
conservative |  1e-4         *          |  1e-5         *
-------------------------------------------------------------------------------
```

* : lteratio=10.0 for conservative errpreset by default. However, when the option reltol <= 1e-4*10.0/3.5, lteratio is set to 3.5.

The values of `reltol` are usually different in the tstab interval and in the hb interval. During tstab, reltol is set to the option reltol, whose default value is 1e-3. This part is not impacted by errpreset. If you want to change the value, set reltol in Spectre options. Any value more than 1e-3 is ignored.

The value of `reltol` in the hb interval is affected by both errpreset and the option reltol. errpreset sets the maximum value of reltol (as shown in the table above). If the option reltol is less than the maximum value, it is set to the option reltol. Otherwise, the maximum value is used. reltol value that is more than 1e-3 is ignored.

If errpreset is not specified in the netlist, moderate settingsis used.

© 2003-2014 All Rights Reserved.

If the circuit you are simulating has 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 can 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 as follows:

`ic=dc`: All initial conditions are ignored, and the DC solution is used.

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

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

`ic=all`: Both `ic` statements and `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.

After 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 because Spectre can have convergence problems while trying to make the jump.

You can skip DC analysis entirely by using the parameter `skipdc`. If DC analysis is skipped, the initial solution is trivial, or is given in the file you specified by using the `readic` parameter, or if the `readic` parameter is not given, by 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 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

© 2003-2014 All Rights Reserved.

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 descriptions are as follows:

`skipdc=no`: Initial solution is calculated using 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 ramp-up phase. The ramp-up 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 the simulator find the DC or initial transient solution. You can specify nodesets in the circuit description file with `nodeset` statements, or in a separate file using the `readns` parameter. When nodesets are specified, Spectre computes an initial guess of the solution by performing DC analysis, while forcing the specified values on to 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 solution. 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 significantly speed up convergence.

Nodesets and initial conditions have similar implementation, but produce different effects. Initial conditions 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. Because differential equations have an infinite number of solutions, a complete set of initial conditions must be specified to identify the required solution. Any initial conditions that 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.

With parameter `hbhomotopy`, you can specify harmonic balance homotopy selection methods. The possible values of parameter `hbhomotopy` and their descriptions are as follows:

© 2003-2014 All Rights Reserved.

`hbhomotopy=tstab`: Simulator runs a transient analysis and generates an initial guess for harmonic balance analysis; it is recommended for nonlinear circuits or circuits with frequency dividers.

`hbhomotopy=source`: For driven circuit, simulator ignores tstab and accordingly increases the source power level; for oscillators, the simulator accordingly adjusts the probe magnitude until probe has no effect on the oscillators. It is recommended for strongly nonlinear or high Q circuits.

`hbhomotopy=tone`: This method is valid only for multi-tone circuit. The simulator first solves a single-tone circuit by turning off all the tones, except the first one, and then solves the multi-tone circuit by restoring all the tones and using the single-tone solution as its initial guess. It is recommended for multi-tone simulation with a strong first tone.

`hbhomotopy=inctone`: The simulator first solves a single tone, then turns on moderate tones incrementally till all tones are enabled. It is recommended for circuits with one strong large tone.

`hbhomotopy=gsweep`: A resistor, whose conductance is g, is connected with each node, and the sweep of g is controlled by gstart, gstop, and glog. It is recommended for circuits containing high-impedance or quasi-floating nodes.

Parameter Index

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

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

HB AC Analysis (hbac)

Description

The harmonic balance AC (HBAC) analysis computes transfer functions for circuits that exhibit single or multi-tone frequency translation. Such circuits include mixers, switched-capacitor filters, samplers, phase-locked loops, and so on. HBAC is a small-signal analysis like AC analysis, except that the circuit is first linearized about a periodically or quasi-periodically varying operating point, rather than about a simple DC operating point. Linearizing about a periodically or quasi-periodically time-varying operating point allows transfer-functions that include frequency translation, which is not the case when linearizing about a DC operating point because linear time-invariant circuits do not exhibit frequency translation. In addition, 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 or quasi-periodically varying circuit is a two-step process. First, the small stimulus is ignored and the periodic or quasi-periodic steady-state response of the circuit to possibly large periodic stimulus is computed using HB analysis. As part of the HB analysis, the periodically or quasi-periodically 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 or quasi-periodically varying linear representation to compute the small signal response. This is done using the HBAC analysis. An HBAC analysis cannot be used independently; it must follow an HB analysis. However, any number of periodic or quasi-periodic small-signal analyses, such as HBAC or HBNOISE, can follow an HB analysis.

Modulated small signal measurements are possible using the Analog Design Environment(ADE). The `modulated` option for HBAC and other modulated parameters are set by ADE. HBAC analyses with this option produce results that can have limited use outside ADE. 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, see the *Virtuoso® Spectre® Circuit Simulator and Accelerated Parallel Simulator RF Analysis User Guide*.

Note: Unlike other analyses in Spectre, the HBAC analysis can only sweep frequency.

Definition

```
Name ... hbac parameter=value ...
```


© 2003-2014 All Rights Reserved.

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=unspecified`

Specifies if the sweep frequency range is the absolute frequency of input or if it is relative to the port harmonics. When the unspecified value is used, Spectre RF sweeps the absolute input source for non-PSP-driven cases; for other cases, Spectre RF sweeps relative to the port harmonics.
Possible values are `absolute`, `relative`, and `unspecified`.

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

Sampled analysis parameters

12 `ptvtype=timeaveraged`

Specifies if the PTV analysis will be traditional or sampled under certain conditions.
Possible values are `timeaveraged` and `sampled`.

© 2003-2014 All Rights Reserved.

13 `sampleprobe` The crossing event at this port triggers the sampled small signal computation.

14 `thresholdvalue=0` Sampled measurement is done when the signal crosses this value.

15 `crossingdirection=all`

Specifies the transitions for which sampling must be done. Possible values are `all`, `rise`, `fall`, and `ignore`.

16 `maxsamples=16` Maximum number of sampled events to be processed during the sampled analysis.

17 `extrasampletimepoints=[...]`

Additional time points for sampled PTV analysis.

Output parameters

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

19 `maxsideband=7` An alternative to the `sidebands` array specification, which automatically generates the array: [-maxsideband ... 0 ... +maxsideband]. For shooting analysis, the default value is `7`. For HB small-signal analysis, the default value is the `harms/maxharms` setting in the HB large signal analysis. It is ignored in HB small signal when it is more than the `harms/maxharms` value of large signal.

20 `freqaxis` Specifies whether the results should be printed as per the input frequency, the output frequency, or the absolute value of the output frequency. Default is `absout`.
Possible values are `absout`, `out` and `in`.

21 `save` Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

22 `nestlvl` Levels of subcircuits to output.

23 `oscout=total` The type of output for oscillator simulation. Default value is total for the output of total modulation response from oscillator

© 2003-2014 All Rights Reserved.

simulation. Other values are `pm` for the output of phase-modulation response and `am` for the output of amplitude-modulation response.
Possible values are `total`, `pm` and `am`.

Convergence parameters

24 `relativeTol` Relative tolerance for harmonic balance-based linear solver. The default value is 1.0e-2.

25 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, and `gmres_cycle`.

26 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, and `recyclelong`.

27 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is `autoset`. With `autoset`, the simulator automatically selects the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when `autoset` is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulator's instructions may speed up subsequent analyses.
Possible values are `basicsolver`,`blocksolver`, and `autoset`.

28 `krylov_size=200` The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching `krylov_size` iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase the `krylov_size` if the simulation reports insufficient norm reduction errors in GMRES.

© 2003-2014 All Rights Reserved.

Annotation parameters

29 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, and `detailed_hb`.

30 `title` Analysis title.

Modulation conversion parameters

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

32 `inmodharmnum=1` Harmonic value for the PAC input source modulation.

33 `outmodharmvec=[...]` Harmonic list for the PAC output modulations.

34 `moduppersideband=1`
Index of the upper sideband included in the modulation of an output for PAC or an input for PXF.

35 `modsource` Refer the output noise to this component.

36 `perturbation=linear`
The type of PAC analysis. Default is linear for normal PAC analysis. im2ds stands for im2 distortion summary and ds stands for distortion summary.
Possible values are `linear`, `ds`, `ip3`, `ip2`, `im2ds`, and `multiple_beat`.

37 `flin_out=0 Hz` Frequency of linear output signal.

38 `fim_out=0 Hz` Frequency of IM output signal.

39 `out1="NULL"` Output signal 1.

40 `out2="NULL"` Output signal 2.

41 `contriblist="NULL"` Array of device names for distortion summary. When `contriblist[""]`, distortion from each non-linear devices is calculated.

© 2003-2014 All Rights Reserved.

42	`maxharm_nonlin=4`	Maximum harmonics of input signal frequency induced by non-linear effect.
43	`rfmag=0`	RF source magnitude.
44	`rfdbm=0`	RF source dBm.
45	`rf1_src="NULL"`	Array of RF1 source names for IP3/IP2/IM2.
46	`rf2_src="NULL"`	Array of RF2 source names for IP3/IP2/IM2.
47	`rf_src="NULL"`	Array of RF source names for triple beat analysis.
48	`freqs=NULL`	Array of RF source frequencies for triple beat analysis.
49	`rfampls=NULL`	RF source amplitudes; the units are dBm for ports, Voltage for v-sources and Ampere for i-sources.

You can select the set of periodic small-signal output frequencies of interest by setting either the `maxsideband` or the `sidevec` parameter. When there is only one tone in HB analysis, sidebands are n integer numbers, K1, K2, ..., Kn, and the output frequency at each sideband is computed as follows:

```
f(out)= f(in) + Ki * fund(hb)
```

where `f(in)` represents the (possibly swept) input frequency and `fund(hb)` represents the fundamental frequency used in the corresponding HB analysis. Thus, when analyzing a down-converting mixer, while sweeping the RF input frequency, the most relevant sideband for IF output is Ki= -1. When simulating an up-converting mixer, while sweeping IF input frequency, the most relevant sideband for RF output is Ki= 1. By setting the `maxsideband` value to Kmax, all 2 * Kmax + 1 sidebands from -Kmax to +Kmax are generated.

When there are multiple tones in HB analysis, sidebands are vectors. Consider that you have one large tone and one moderate tone in HB. A sideband, K1, is represented as [K1_1 K1_2]. Corresponding frequency is as follows:

`K1_1 * fund(large tone of HB) + K1_2 * fund` (moderate tone of HB)

The assumption is that there are L large and moderate tones in HB 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 follows:

```
f(out)= f(in) + SUM_j=1_to_L{Ki_j * fund_j(hb)},
```


© 2003-2014 All Rights Reserved.

where `f(in)` represents the (possibly swept) input frequency, and `fund_j(hb)` represents the fundamental frequency used in the corresponding HB analysis. Therefore, 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}. You 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 `maxsideband`, only the large tone, which is the first fundamental, is affected by this entry. All the other tones, which are the moderate tones, are limited by `maxharms` specified for an HB analysis. Given `maxharms=[k1max k2max ... knmax]` in HB and `maxsideband=Kmax`, all `(2*Kmax + 1)*(2*k2max+1)*(2*k3max+1) *...*(2*knmax+1)` sidebands are generated.

The number of requested sidebands changes the simulation time substantially.

With HBAC, the frequency of the stimulus and of the response are usually different (this is an important area in which HBAC 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 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 specify a step size parameter (`step`, `lin`, `log`, or `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 can use the `values` parameter to specify the values that the `sweep` parameter should take. If you provide 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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

HB Noise Analysis (hbnoise)

Description

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

HBNOISE analysis linearizes the circuit about the periodic or quasi-periodic operating point computed in the prerequisite HB analysis. It is the periodically or quasi-periodically time-varying nature of the linearized circuit that accounts for the frequency conversion. In addition, the effect of a periodically or 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 `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` can be used as an input probe. If the input source is noisy, as is a `port`, the noise analysis computes 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, 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(input)| = |f(out) + refsideband frequency shift|.`

For periodic noise (only one tone in HB analysis), `refsideband` is a number. Use `refsideband=0` when the input and output of the circuit are at the same frequency, such as

© 2003-2014 All Rights Reserved.

with amplifiers and filters. When `refsideband` differs from 0, the single side-band noise figure is computed.

While for quasi-periodic noise (multiple tones in HB analysis), reference sidebands are vectors. Assume that there is one large tone and one moderate tone in HB. A sideband Ki is a vector [Ki_1 Ki_2]. It gives the frequency at:

```
Ki_1 * fund(large tone of HB) + Ki_2 * fund(moderate tone of HB)
```

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

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.

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. Therefore, 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

© 2003-2014 All Rights Reserved.

NFieee = IEEE single sideband noise figure

Then:

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

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

NF = 10*log10(F)

Fdsb = (No^2 - Nl^2)/(Ns^2+Nsi^2)

NFdsb = 10*log10(Fdsb)

Fieee = (No^2 - Nl^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.

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

When option `xfonly` is set to `yes`, only XF analysis is done. In other words, HBNOISE analysis does only a conventional transfer function analysis which computes the transfer function from every source in the circuit to a single output. This analysis differs from a conventional AC analysis in that the AC analysis computes the response from a single stimulus to every node in the circuit. It computes the transfer functions from any source at any frequency to a single output at a single frequency. Therefore, similar to HBAC analysis, it includes frequency conversion effects. It 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).

An HBNOISE analysis must follow an HB analysis.

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

Definition

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


© 2003-2014 All Rights Reserved.

The optional terminals (p and n) specify the output of the circuit. If you do not specify the terminals, 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=unspecified`
Specifies if the sweep frequency range is the absolute frequency of input or if it is relative to the port harmonics. When the unspecified value is used, Spectre RF sweeps the absolute input source for non-PSP-driven cases; for other cases, Spectre RF sweeps relative to the port harmonics.
Possible values are `absolute`, `relative`, and `unspecified`.

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.

© 2003-2014 All Rights Reserved.

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` and `individual`.

Sampled analysis parameters

16 `ptvtype=timeaveraged`
Specifies if the PTV analysis will be traditional or sampled under certain conditions.
Possible values are `timeaveraged`, and `sampled`.

17 `extrasampletimepoints=[...]`
Additional time points for sampled PTV analysis.

18 `sampleprobe` The crossing event at this port triggers the sampled small signal computation.

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

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

21 `numberofpoints=5` Number of time points of interest in the period where the time domain PSD is calculated. 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.

22 `thresholdvalue=0` Sampled measurement is done when the signal crosses this value.

23 `crossingdirection=all`
Specifies the transitions for which sampling needs to be done.
Possible values are `all`, `rise`, `fall`, and `ignore`.

© 2003-2014 All Rights Reserved.

24 `maxsamples=16` Maximum number of sampled events to be processed during the sampled analysis.

Output parameters

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

26 `maxsideband=7` In shooting pnoise, the parameter determines the maximum sideband to be included when computing noise that is either up-converted or down-converted to the output by the periodic drive signal. In HB pnoise, the parameter determines the size of the small signal system when the HB pnoise is performed. This parameter is critical for the accuracy of the HB pnoise analysis. Using a small value for `maxsideband` might cause accuracy loss.

The default value for shooting pnoise is 7. And, for HB pnoise, the default is the `harms/maxharms` setting in the HB large signal analysis.

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

28 `save` Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

29 `nestlvl` Levels of subcircuits to output.

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

31 `saveallsidebands=no`
Save noise contributors by sideband.
Possible values are `no` and `yes`.

32 `xfonly=no` Perform XF analysis only.
Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

33 `stimuli=sources` Stimuli used for XF analysis in hbnoise.
Possible values are `sources` and `nodes_and_terminals`.

34 `separatenoise=no` Separate noise into sources and transfer functions.
Possible values are `no` and `yes`.

35 `cyclo2txtfile=no` Output cyclo-stationary noise to text file as input source of next stage.
Possible values are `no` and `yes`.

36 `oscout=total` The type of output for oscillator simulation. Default value is total for the output of total modulation response from oscillator simulation. Other values are `pm` for the output of phase-modulation response and `am` for the output of amplitude-modulation response.
Possible values are `total`, `pm` and `am`.

Convergence parameters

37 relative`Tol` Relative tolerance for harmonic balance-based linear solver. The default value is 1.0e-2.

38 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, and `gmres_cycle`.

39 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, and `recyclelong`.

40 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is `autoset`. With `autoset`, the simulator automatically selects the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when `autoset` is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulator's

© 2003-2014 All Rights Reserved.

instructions may speedup subsequent analyses.
Possible values are `basicsolver`,`blocksolver`, and `autoset`.

41 `ppv=no` — If set to `yes`, save the oscillator PPV after performing noise analysis.
Possible values are `no` and `yes`.

42 `augmented=yes` — If set to `yes`, the frequency-aware PPV method is used to calculate the total noise of the oscillator; if set to `pmonly`, only the PM part of the oscillator noise is calculated; if set to `amonly`, only the AM part of the oscillator noise is calculated.
Possible values are `no`, `yes`, `pmonly`, and `amonly`.

43 `lorentzian=cornerfreqonly`
This option determines if the Lorentzian plot is used in the oscillator noise analysis.
Possible values are `no`, `cornerfreqonly` and `yes`.

44 `krylov_size=200` — The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching `krylov_size` iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase the `krylov_size` if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

45 `annotate=sweep` — Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, and `detailed_hb`.

46 `title` — Analysis title.

In practice, noise can mix with each of the harmonics of the periodic drive signal applied in the HB analysis and end up at the output frequency. However, the HBNOISE analysis includes only the noise that mixes with a finite set of harmonics that are typically specified using the `maxsideband` parameter.

If Ki represents sideband i, then for periodic noise:

```
f(noise_source) = f(out) + Ki * fund(hb)
```


© 2003-2014 All Rights Reserved.

For quasi-periodic noise with multi-tone in HB analysis, assuming that there is one large tone and one moderate tone, Ki is represented as [Ki_1 Ki_2]. Corresponding frequency shift is as follows:

```
Ki_1 * fund(large tone of HB) + Ki_2 * fund(moderate tone of HB)
```

If there are L large and moderate tones in HB analysis and a set of n integer vectors representing the sidebands:

```
K1 = { K1_1,...K1_j..., K1_L}, K2, ..., Kn
```

Then:

```
f(noise_source) = f(out) +  SUM_j=1_to_L{ Ki_j * fund_j(hb) }
```

The `maxsideband` parameter specifies the maximum |Ki| included in the HBNOISE calculation. For quasi-periodic noise, only the large tone, which is the first fundamental, is affected by this entry. All the other tones, which are the moderate tones, are limited by `maxharms` specified for an HB analysis.

The number of requested sidebands changes the simulation time substantially.

When HBNOISE analysis does only an xf analysis (`xfonly=yes`), 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 the `maxsideband` parameter. With this analysis, the frequency of the stimulus and of the response are usually different (this is an important area in which this analysis differs from XF).

You can designate a voltage to be the output by specifying a pair of nodes on the HBNOISE analysis statement or by using the `oprobe` parameter. 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.

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. You must not specify both `portv` and `porti`. If you specify neither, the probe component provides a reasonable default.

You can use the `stimuli` parameter to specify what serves as the inputs for the transfer functions. There are two choices: `stimuli=sources` or `stimuli=nodes_and_terminals`.

`stimuli=sources` indicates that the sources present in the circuit are to be used. You can use the `xfmag` parameters provided by the sources to adjust the computed gain to

© 2003-2014 All Rights Reserved.

compensate for gains or losses in a test fixture. You 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 are to 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 you want transfer functions from specific terminals, specify the terminals in the save statement. You must use the `:probe` modifier, (for example, Rout:1:probe), or specify `useprobes=yes` on the options statement. If you want transfer functions from all terminals, specify `currents=all` and `useprobes=yes` on the options statement.

You can specify sweep limits by providing the end points or the center value and 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`) to determine whether the sweep is linear or logarithmic. If you do not provide 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 can use the values parameter to specify the values that the sweep parameter should take. If you provide 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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

HB S-Parameter Analysis (hbsp)

Description

The periodic or quasi-periodic SP (HBSP) analysis is used to compute scattering and noise parameters for n-port circuits such as mixers that exhibit frequency translation. It is a small-signal analysis similar to SP analysis, except that in HBAC and HBNOISE, the circuit is first linearized about a periodically varying operating point as opposed to a simple DC operating point. Linearizing about a periodically or quasi-periodically time-varying operating point allows the computation of S-parameters between circuit ports that convert signals from one frequency band to another. HBSP can also calculate noise parameters in frequency-converting circuits. In addition, HBSP computes noise figure (both single-sideband and double-sideband), input referred noise, equivalent noise parameters, and noise correlation matrices. Similar to HBNOISE, but unlike SP, the noise features of the HBSP analysis include noise folding effects due to the periodic 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 or quasi-periodic steady-state response of the circuit to possibly large periodic stimulus is computed using HB analysis. As a part of the HB 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 HBSP analysis. HBSP analysis cannot be used independently; it must follow HB analysis. However, any number of periodic small-signal analyses such as HBAC, HBSP, HBNOISE, can follow an HB analysis.

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

Definition

```
Name hbsp parameter=value ...
```

Parameters

Sweep interval parameters

1	`start=0`	Start sweep limit.
2	`stop`	Stop sweep limit.
3	`center`	Center of sweep.

© 2003-2014 All Rights Reserved.

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=unspecified`
Specifies if the sweep frequency range is absolute frequency of input or if it is relative to the port harmonics. When the `unspecified` value is used, Spectre RF sweeps the absolute input source for non-PSP-driven cases; for other cases, Spectre RF sweeps relative to the port harmonics.
Possible values are `absolute`, `relative`, and `unspecified`.

Port parameters

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

12 `portharmsvec=[...]` List of harmonics that are 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 the ones associated with specific ports by `portharmsvec`, that are active.

Output parameters

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

© 2003-2014 All Rights Reserved.

Noise parameters

15 `donoise=yes`

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

Probe parameters

16 `maxsideband=7`

In shooting pnoise, this parameter determines the maximum sideband to be included when computing noise that is either up-converted or down-converted to the output by the periodic drive signal. When running HB pnoise, the parameter determines the size of the small signal system when HB pnoise is performed. This parameter is critical for the accuracy of the HB pnoise analysis. Using a small value for `maxsideband` might cause accuracy loss.

The default value for the shooting pnoise is 7. And, for the HB pnoise, the default is the `harms/maxharms` setting in the HB large signal analysis.

Convergence parameters

17 `relativeTol`

Relative tolerance for harmonic balance-based linear solver. Default value is 1.0e-2.

18 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is `autoset`. With `autoset`, the simulator automatically selects the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when `autoset` is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulator's instructions may speed up subsequent analyses.
Possible values are `basicsolver`,`blocksolver` and `autoset`.

© 2003-2014 All Rights Reserved.

19 `krylov_size=200` The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching `krylov_size` iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase the `krylov_size` if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

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

21 `title` Analysis title.

To specify the HBSP analysis, the port and port harmonic relations must be specified. You can select the ports of interest by setting the `port` parameter, and select the set of periodic small-signal output frequencies of interest by setting `portharmsvec` or `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 following frequencies:

For periodic SP in one-tone HB analysis, frequency is:

```
f(scattered)= f(rel) + Ki * fund(HB)
```

For quasi-periodic noise with multi-tone in HB analysis, sidebands are vectors. Consider that you have one large tone and one moderate tone in HB. Then, the above sideband K1 will be represented as [K1_1 K1_2]. In this case, the corresponding frequency is:

```
K1_1 * fund(large tone of HB) + K1_2 * fund(moderate tone of HB)=
SUM_j=1_to_L{Ki_j * fund_j(HB)}
```

If there are L (1 large and L-1 moderate) tones in HB analysis and a given set of n integer vectors representing the sidebands:

```
K1 = { K1_1,...K1_j..., K1_L } , K2, ... Kn
```

If you specify the relative frequency, the scattering parameters at each port are computed at the frequencies:

```
f(scattered)= f(rel) + SUM_j=1_to_L{Ki_j * fund_j(hb)},
```

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

© 2003-2014 All Rights Reserved.

represents the conversion, and `fund`(one-tone HB) or `fund_j`(multi-tone HB) represents the fundamental frequency used in the corresponding HB analysis.

During 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. In case of periodic SP with one-tone HB, the most relevant harmonic for RF input is Ki= 1 and for IF output Ki= 0. Therefore, we can associate K2=0 with the IF port and K1=1 with the RF port. S21 represents the transmission of signal from the RF to IF, and S11 represents the reflection of signal back to the RF port. If the signal was in the lower sideband, a choice of K1=-1 is more appropriate. For quasi-periodic SP with multi-tone HB, the most relevant sideband for this input is Ki= {1 , 0} and for IF output Ki= {0 , 0}. Therefore, we can associate K1={1 , 0} with the RF port and K2={0 , 0} with the IF port. If the signal was in the lower sideband, then a choice of K1={-1 , 0} is more appropriate.

`portharmsvec` or `harmsvec` parameters can be used to specify the harmonics of interest. If `portharmsvec` is specified, 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, all possible frequency-translating scattering parameters associated with the specified harmonics are computed.

With HBSP the frequency of the input and of the response are usually different (this is an important area in which HBSP differs from SP). Because the HBSP computation involves inputs and outputs at frequencies that are relative to multiple harmonics or sidebands, the `freqaxis` and `sweeptype` parameters behave differently in HPSP than in HBAC and HBNOISE.

The `sweeptype` parameter controls the way the frequencies in the HBSP analysis are swept. Specifying a `relative` sweep indicates the sweep to be relative to the analysis harmonics or port sideband (not the HB fundamental) and specifying an `absolute` sweep indicates the sweep of the absolute input source frequency.

For example, in case of periodic SP with one-tone HB and HB fundamental of 100MHz, `portharmsvec` set to [9 1] to examine a downconverting mixer, `sweeptype=relative`, and a sweep range of f(rel)=0->50MHz, S21 represents the strength of signal transmitted from the input port in the range 900->950MHz to the output port at frequencies 100->150MHz. Using `sweeptype=absolute` and sweeping the frequency from 900->950MHz would calculate the same quantities, because f(abs)=900->950MHz, f(rel) = f(abs) - K1 * fund(hb) = 0->50MHz, and K1=9 and fund(hb) = 100MHz.

For quasi-periodic noise with multi-tone HB, and HB fundamentals of 1000MHz (LO) and 966MHz (blocker in RF channel), `portharmsvec` could be set to [0 1 -1 1] to examine a downconverting mixer. Consider setting `sweeptype=relative` and a sweep range of f(rel)=-10MHz<->10MHz. Then, S21 will represent the strength of the signal transmitted from the input port in the range 956->976MHz to the output port at frequencies 24<->44MHz.

© 2003-2014 All Rights Reserved.

Using `sweeptype=absolute` and sweeping the frequency from 966<->976MHz will calculate the same quantities, because f(abs)=956<->976MHz, f(rel) = f(abs) - (K1_1 * fund_1(hb) + K1_2 * fund_2(hb) = -10MHz<->10MHz, andK1_1=0, K1_2=1 and fund_1(hb) = 1000MHz, fund_2(hb) = 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`).

HBSP 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:

No = total output noise at frequency f

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 (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(No^2/G^2)

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

NF = 10*log10(F)

© 2003-2014 All Rights Reserved.

Fdsb = (No^2 - Nl^2)/(Ns^2+Nsi^2)

NFdsb = 10*log10(Fdsb)

Fieee = (No^2 - Nl^2 - Nso^2)/Ns^2

NFieee = 10*log10(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 HBSP 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 HBSP.

You can specify sweep limits by giving the end points or by providing the center value and span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. In addition, you can specify 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 use the `values` parameter to specify the values that the sweep parameter should take. If you use 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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

```
freqaxis  14        maxsideband  16     step  5

harmsvec  13        portharmsvec  12    stop  2
```


© 2003-2014 All Rights Reserved.

Circuit Information (info)

Description

The circuit information analysis outputs several types 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 provides 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`	The parameters that should be printed. Possible values are `none`, `inst`, `models`, `input`, `output`, `nodes`, `all`, `terminals`, `oppoint`, `captab`, `parameters`, `primitives`, `subckts`, `assert`, `allparameters`, `netlist`, `options`, and `dumpall`.
2	`where=logfile`	Where the parameters should be printed. Asserts can only be written to rawfile. Possible values are `nowhere`, `screen`, `file`, `logfile`, and `rawfile`.

© 2003-2014 All Rights Reserved.

3	`file="%C:r.info.what"`	File name when `where=file`.
4	`save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.
5	`nestlvl`	Levels of subcircuits to output.
6	`extremes=yes`	Print minimum and maximum values. Possible values are `no`, `yes`, and `only`.
7	`title`	Analysis title.
8	`descriptions=no`	Print descriptions. Possible values are `no` and `yes`.

Captab parameters

9	`detail=node`	How detailed should the capacitance table be. Possible values are `node`, `nodetoground`, and `nodetonode`.
10	`sort=name`	How to sort the capacitance table. Possible values are `name` and `value`.
11	`threshold=0 F`	Threshold value for printing capacitances (ignore capacitances smaller than this value).

Parameter Index

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

© 2003-2014 All Rights Reserved.

Load Pull Analysis (loadpull)

Description

The `loadpull` analysis.

Definition

```
Name loadpull parameter=value ...
```

Parameters

1	`rho`	Name of parameter to rho sweep.
2	`rhostart=0`	Start sweep limit of rho.
3	`rhostop`	Stop sweep limit of rho.
4	`rhostep`	Step size, linear sweep of rho.
5	`rholin=50`	Number of steps, linear sweep of rho.
6	`rhovalues=[...]`	Array of sweep values of rho.
7	`phi`	Name of parameter to phi sweep.
8	`phistart=0`	Start sweep limit of phi.
9	`phistop`	Stop sweep limit of phi.
10	`phistep`	Step size, linear sweep of phi.
11	`philin=50`	Number of steps, linear sweep of phi.
12	`phivalues=[...]`	Array of sweep values of phi.
13	`inst`	Port instance of load.
14	`z0=50`	the Z0 of the load port.

© 2003-2014 All Rights Reserved.

Parameter Index

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

© 2003-2014
All Rights Reserved.

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 about 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 you constructed as scalar calculator expressions during Monte Carlo analysis setup, can be used to measure circuit metrics, such as the slew-rate of an op-amp. For each iteration during Monte Carlo analysis, the expression results vary with the netlist parameters. Therefore, Monte Carlo analysis 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. 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 {
```


© 2003-2014 All Rights Reserved.

```
    run child analysis

      evaluate expressions

  }

}
```

Definition

```
Name montecarlo parameter=value ...
```

Parameters

Analysis parameters

1	`numruns=100`	Number of Monte Carlo iterations to perform (does not include the nominal run).
2	`firstrun=1`	Starting iteration number.
3	`variations=process`	Level of statistical variation to apply. Possible values are `process`, `mismatch`, and `all`.
4	`sampling=standard`	Method of statistical sampling to apply. Possible values are `standard`, `lhs`, `orthogonal`, and `lds`.
5	`numbins=0`	Number of bins for latin-hypercube(lhs) and orthogonal method. The number is checked against numruns + firstrun - 1, and Max(numbins, numruns + firstrun -1) is used.
6	`seed`	Optional starting seed for random number generator.
7	`scalarfile`	Output file that contains output scalar data.
8	`paramfile`	Output file that contains output scalar data labels.
9	`dut=[...]`	If set, the specified subcircuit instance have process and mismatch variations applied and the unspecified instance only have process variations applied. All subcircuits instantiated

© 2003-2014 All Rights Reserved.

under this instance also have process and mismatch enabled. By default, mismatch is applied to all subcircuit instances in the design and process is applied globally. This parameter allows the test-bench to change and not affect the variations seen by the actual design.

10 `ignore=[...]` If set, no variation is applied to specified subcircuit instances. In addition, all subcircuits instantiated under this instance do not have variation enabled. By default, the mismatch is applied to all subcircuit instances in the design and the process is applied globally.

11 `dutparams=[...]` If set, only the specified statistical parameters have process and mismatch variations applied.

12 `ignoreparams=[...]` If set, the specified statistical parameters are excluded from applying process and mismatch variation.

13 `accuracyaware=summary`
Specifies the mode of runtime monitoring and early termination of a montecarlo simulation. In `summary` mode, the statistics of measurement is only listed after the Monte-Carlo analysis finishes. In `iteration` mode, the statistics is printed after each step of Monte-Carlo analysis. In `autostop` mode, analysis is terminated based on the criteria specified by the options `minmaxpairs`, and `smoothhthresh`.
Possible values are `summary`, `iteration`, and `autostop`.

14 `minmaxpairs=[...]` Pairs of values that are used to specify the min and max of each measurement defined in ocean expressions. Simulation is terminated when the current iteration generates a measurement that resides outside the region of [min, max]. It's not necessary that the number of pairs equals the number of measurement. However, each defined pair must be aligned with the corresponding ocean measurement. Extra number of pairs or measurements are ignored when deciding upon early termination. This option is active only when `accuracyaware=autostop`.

15 `smoothhthresh=0.0` Specifies the smoothness threshold of an averaged ocean measurement. The average takes place within consecutive non-overlapping 200-iteration windows. Recommended value is 1e-4

© 2003-2014 All Rights Reserved.

for a reasonably converged signal-average. This test of smoothness is active only when `accuracyaware=autostop`.

16 `method=standard` Method used to run montecarlo analysis. Standard is the regular montecarlo analysis with varying netlist parameters. VADE is variation aware design with directly varying device parameters. Possible values are `standard` and `vade`.

Saving Process Parameters

17 `saveprocessparams`
Whether to save scalar data for statistically varying process parameters that are subject to process variation.
Possible values are `no` and `yes`.

18 `processscalarfile`
Output file that contains process parameter scalar data.

19 `processparamfile`
Output file that contains process parameter scalar data labels.

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

21 `savemismatchparams=no`
Whether to save scalar data for statistically varying mismatch parameters that are subject to mismatch variation.
Possible values are `no` or `yes`.

22 `mismatchscalarfile`
Output file that contains mismatch parameter scalar data.

23 `mismatchparamfile`
Output file that contains mismatch parameter scalar data labels.

Flags

24 `donominal=yes` Whether to perform nominal run.
Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

25 `addnominalresults=no`
Whether to add nominal run results to MC run results.
Possible values are `no` and `yes`.

26 `paramdumpmode=no` Whether to dump process/mismatch parameters information.
Possible values are `no` and `yes`.

27 `dumpseed=no` Whether to dump seed parameters information.
Possible values are `no` and `yes`.

28 `nullmfactorcorrelation=no`
Whether to set 0% correlation mismatch devices with m-factor.
Possible values are `no` and `yes`.

29 `appendsd=no` Whether to append scalar data.
Possible values are `no` and `yes`.

30 `savefamilyplots=no`
Whether to save data for family plots. If set to `yes`, this could require considerable disk space.
Possible values are `no` and `yes`.

31 `savedatainseparatedir=no`
Whether to save data for each plot in a separate directory. If set to `yes`, this could require considerable disk space.
Possible values are `no` and `yes`.

Annotation parameters

32 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, and `status`.

33 `title` Analysis title.

34 `usesamesequence=no`
If set to `yes`, the random number sequence is maintained for a seed, even if there is a non-empty dut/ignore list. Possible values are `no` and `yes`.

Detailed Description and Examples

`sampling=[standard | lhs]`

© 2003-2014 All Rights Reserved.

Determines the sampling behavior. This parameter can be set to `standard`, the default value, or `lhs`. `lhs` invokes the latin-hypercube (LHS) method, while `standard` defaults to the existing standard sampling behavior.

`numbins=value`

Controls the number of subdivisions used in the LHS method.

If `numbins` is not specified, the number of subdivisions of the sampling space in LHS will be `numruns + firstrun - 1`. This parameter is active only when sampling is `lhs`. If `numbins` is set to a non-zero integer, the number of subdivisions will be assigned to the greater of the two values `numbins` or `numruns + firstrun -1`.

`numruns:(default=100)`

Specifies the number of Monte Carlo iterations to perform. The simulator performs a loop, running the specified child analyses, and evaluating any expressions `numruns` times.

`seed:(no default)`

Specifies the 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, each time you run the analysis, you will get different results, that is, a different stream of pseudo-random numbers is generated.

`scalarfile="filename"`

Allows you to specify an ASCII file in which scalar data (results of expressions that resolve to scalar values) is written. The data from this file can be read and plotted in histograms by ADE. For each iteration of each Monte Carlo child analyses, Spectre (through Artil) writes a line to this ASCII file, which contains scalar data (one scalar expression per column, for example, slewrate or bandwidth). The default name for this file is of the form *`name`*`.mcdata`, where *`name`* is the name of the Monte Carlo analysis instance. This file contains only the matrix of numeric values. If you are an ADE Monte Carlo user, you will be more familiar with the term `mcdata` file for the scalar file. Additionally, when the ADE Monte Carlo tool is used to generate the Spectre netlist file, Spectre merges the values of the statistically varying process parameters into this file that contains the scalar data (results of expressions). This means that ADE 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, you can see correlations between process parameter variations and circuit performance variations. This data merging occurs whenever the `scalarfile` and `processscalarfile` are written in the same directory.

`paramfile="filename"`

© 2003-2014 All Rights Reserved.

Contains the titles, sweep variable values, and the full expression for each of the columns in the scalarfile. If you are an ADE Monte Carlo user, you will be more familiar with the term `mcparam` file for the paramfile. This file is created in the psf directory by default, unless you specify an alternative path with the file name.

`processscalarfile="filename"`

If `saveprocessparams` is set to `yes`, the process (batch-to-batch) values of all statistically varying parameters are saved to this scalar data file. You can use `saveprocessvec` to filter out a subset of parameters in which case Spectre will save only the parameters specified in `saveprocessvec` to the `processscalarfile`). `processscalarfile` is equivalent to `scalarfile`, except that the data in the `scalarfile` contains the values of the scalar expressions, whereas the data in `processscalarfile` contains the corresponding process parameter values. The default name for this file is of the form *`instname`*`.`*`process`*`.mcdata`, where *`instname`* is the name of the Monte Carlo analysis instance. This file is created in the psf directory by default, unless you specify an alternative path with the filename. You can load processscalarfile and processparamfile into the ADE statistical post-processing 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 ADE statistical postprocessing environment.

`processparamfile="filename"`

Contains the titles and sweep variable values for each of the columns in `processscalarfile`. These titles are the names of the process parameters.

`processparamfile` is equivalent to the `paramfile`, except that `paramfile` contains the name of the expressions, whereas `processparamfile` contains the names of the process parameters. The default name for this file is of the form *`instname`*`.`*`process`*`.mcparam`, where *`instname`* is the name of the Monte Carlo analysis instance. This file is created in the psf directory by default, unless you specify an alternative location with *`filename`*.

`firstrun:(default=1)`

Specifies the index of the first iteration. If the first iteration is specified as some number n greater than one, then the beginning n-1 iterations are `skipped`, that is, 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)`

© 2003-2014 All Rights Reserved.

Determines whether to apply only process (batch-to-batch) variations, or only mismatch (per-instance) variations, or both. This parameter 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.

`saveprocessvec=[rshsp TOX ...]`

If `saveprocessparams` is set to `yes`, this parameter saves the process (batch-to-batch) values of only those parameters that are listed in saveprocessvec to 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, do not specify this parameter.

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

Controls whether Spectre should perform a nominal run before starting the main Monte Carlo loop of iterations. If any errors are encountered during the nominal run (for example, convergence problems, incorrect expressions, and so on) then Spectre issues an appropriate error message and immediately abandons the Monte Carlo analysis.

If set to `no`, Spectre runs only the Monte Carlo iteration, and does not perform nominal analysis. If any errors are encountered during the Monte Carlo iterations, Spectre issues a warning and continues with the next iteration of the Monte Carlo loop.

`addnominalresults={yes,no}` (defaults to `no`).

Controls whether Spectre should append nominal run results after the Monte Carlo run results in the data files.

`paramdumpmode={yes,no}` (defaults to `no`).

Controls whether Spectre should dump out each process/mismatch parameter distribution type, mean value, standard value, and correlation information into additional parameter files. The names for these files are `insname.process_full.param`, `insname.mismatch_full.param`, `insname.process.correlate.param` and `insname.mismatch.correlate.param`, where *`instname`* is the name of the Monte Carlo analysis instance.

`dumpseed={yes,no}` (defaults to `no`)

Controls whether Spectre should dump out seed parameter and iteration number into the `insname.seed` file, where *`instname`* is the name of the Monte Carlo analysis instance.

`nullmfactorcorrelation={yes,no}` (defaults to `no`)

© 2003-2014 All Rights Reserved.

Controls whether Spectre should emulate a 0% correlation for mismatch devices with m-factor.

`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 set to `yes`, a data file (for example, psf) is saved for each analysis for each Monte Carlo iteration, in addition to the expressions scalar results that 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 ADE. It also enables you to define/evaluate new calculator measurements after the simulation has been run using the Wavescan 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 saved data to a minimum. If this parameter is set to `no`, data files are overwritten by each Monte Carlo iteration.

`savedatainseparatedir={yes,no}`

If set to `yes`, a data file (for example, psf) is saved for each analysis for each Monte Carlo iteration in a separate directory, in addition to the expressions scalar results that are saved to the ASCII scalar data file at the end of each iteration. This feature can result in a huge amount of data being stored to the disk. Therefore, it is recommended that you use this feature with care and keep the saved data to a minimum. If this parameter is set to `no`, data files are overwritten by each Monte Carlo iteration.

`annotate={no,title,sweep,status}`

Specifies the degree of annotation. Use the maximum value of `status` to print a summary of the runs that did not converge, had problems evaluating expressions, and so on.

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
```


© 2003-2014 All Rights Reserved.

```
// 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 represent 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 that may be used for generating truncated distributions. The distributions specified in the process block are 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, the sampled process value becomes the mean for the mismatch random number generator for that particular parameter.

Note: Multiple statistics blocks can exist and in that case the blocks either accumulate or overlay. Typically, process variations, mismatch variations, and correlations between process parameters are specified in one statistics block. A second statistics block should be specified where actual device instance correlations are specified (that is, 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 statistics blocks, which are discussed below along with syntax requirements.

```
// define some netlist parameters to represent process parameters
```


© 2003-2014 All Rights Reserved.

```
// 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
    truncate tr=2.0 // +/- 2 sigma
        ...
    }
    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
        truncate tr=7.0 // +/- 7 sigma
    }
    // 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
}
```

Specifying Distributions

Parameter variations are specified using the following syntax:

```
vary PAR_NAME dist=<type> {std=<value> | N=<value>} {percent=yes|no}
```

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

Gaussian Distribution

© 2003-2014 All Rights Reserved.

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) = Normal( log(mean), 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, the process variation is according to:

```
log(xxx) = Normal( log(20000), 12 )
```

Uniform Distribution

The uniform distribution for parameter x is generated according to:

```
x = unif(mean-N, mean+N)
```

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 by using the formula `std=N/sqrt(3)`.

Values as Percentages

The `percent` flag indicates whether the standard deviation `std` or uniform range `N` are specified in absolute terms (`percent=no`) or as a percentage of the mean value (`percent=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)), that is, Normal (200, 24). It is recommended that you do not 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.

© 2003-2014 All Rights Reserved.

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:

```
correlate param=[list of parameters] cc=<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 are randomly generated according to the specified distributions and correlations.

Mismatch Correlation (Matched Devices)

The syntax of the instance or mismatch correlation statement is:

```
correlate dev=[list of subcircuit instances] {param=[list of parameters]}
     cc=<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 (that is, 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.

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 that 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:

© 2003-2014 All Rights Reserved.

```
truncate tr=<value>
```

The following conditions should be considered while setting the truncate factor:

- The value of the truncation factor can be a constant or an expression.
- Parameter correlations can be affected by using small truncation factors.
- There are different truncate for process and mismatch blocks. If a truncate for process or mismatch block is not given, it will be set to the value of truncate in statistic block.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Noise Analysis (noise)

Description

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 are computed. If the input source is noisy, the noise figure is also 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 by 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 computes 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, 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 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. Therefore, if:

No = total output noise

Ns = noise at the output due to the input probe (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

© 2003-2014 All Rights Reserved.

Then:

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

F = (No^2 - Nl^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 AC 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 at each step. You can sweep the circuit temperature by giving the parameter name as `temp`, without a `dev` or `mod` parameter. In addition, you can sweep a netlist parameter by giving the parameter name without a `dev`, or `mod` parameter. After the analysis is complete, 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 specify the terminals, you must specify the output with a probe component.

Parameters

1	`prevoppoint=no`	Use the operating point computed in the previous analysis. Possible values are `no` and `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.

© 2003-2014 All Rights Reserved.

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 an estimate of DC solution (nodeset).
18	`write`	DC operating point output file at the first step of the sweep.
19	`writefinal`	DC operating point output file at the last step of the sweep.

Initial condition parameters

20	`force=none`	The set of initial conditions to use. Possible values are `none`, `node`, `dev`, and `all`.
21	`readforce`	File that contains initial conditions.

© 2003-2014 All Rights Reserved.

22 `skipdc=no` — Skip DC analysis.
Possible values are `no` and `yes`.

23 `useprevic=no` — If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns.
Possible values are `no`, `yes` and `ns`.

Output parameters

24 `save` — Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, and `selected`.

25 `nestlvl` — Levels of subcircuits to output.

26 `oppoint=no` — Determine whether operating point information should be computed; if yes, where should it be printed (screen or file). Operating point information is not printed if the operating point computed in the previous analysis remains unchanged.
Possible values are `no`, `screen`, `logfile`, and `rawfile`.

Convergence parameters

27 `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` and `yes`.

Annotation parameters

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

29 `title` — Analysis title.

You can define sweep limits by specifying the end points or by providing the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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.

© 2003-2014 All Rights Reserved.

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 an operating point was computed during a previous analysis, you can set `prevoppoint=yes` to avoid recomputing it. For example, if `prevoppoint=yes` when the previous analysis was a transient analysis, the operating point is the state of the circuit at the final time point.

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 by using the `readns` parameter. When nodesets are given, Spectre computes an initial guess of the solution by performing DC analysis, while forcing the specified values on to 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 required 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 required solution. 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 significantly speed up convergence.

When you simulate the same circuit multiple times, it is recommended that you use both `write` and `readns` parameters and assign the same file name to both parameters. DC analysis then converges quickly even if the circuit has changed since the last simulation, and the nodeset file is automatically updated.

During the initial operating point DC analysis, you may force certain circuit variables to use the values given in the `ic` file, `ic` statements, or `ic` parameter on the capacitors and inductors. The `ic` parameter controls the interaction of various methods of setting the force values. The effects of individual settings are as follows:

`force=none`: All initial conditions 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 `ic` statements and `ic` parameters are used, with the `ic` parameters overriding the `ic` statements.

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

© 2003-2014 All Rights Reserved.

After you specify the initial conditions, Spectre computes the DC operating point 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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

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 more options, see the description of individual analyses.

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

In many cases, a particular option can be controlled by either a command-line or netlist specification. In the situation where both are used, the command-line option takes priority over the setting in the netlist options statement.

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=1.0e-6 V`	Voltage absolute tolerance convergence criterion.
4 `iabstol=1.0e-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

© 2003-2014 All Rights Reserved.

dependent on temperature; and if `tempeffect=all`, all built-in temperature models are enabled.
Possible values are `vt`, `tc`, and `all`.

Output parameters

8 `save=selected` — Signals to output public.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

9 `autosavecurvolt=no`
Automatically save terminal currents and node voltage.
Possible values are `no` and `yes`.

10 `nestlvl=INT_MAX` — Levels of subcircuits to output.

11 `subcktprobelvl=0` — Level up to which the subcircuit terminal currents are to be computed.

12 `subcktiprobes=yes`
Insert iprobes when computing subcircuit terminal currents.
Possible values are `no` and `yes`.

13 `amsiprobes=no` — Insert iprobes when computing MDL related terminal currents.
Possible values are `no` and `yes`.

14 `termcur_method=default`
Select methods to compute terminal currents. Set `termcur_method=complete` to get correct currents when devices have no `termcurs` output. However, this may negatively impact performance.
Possible values are `default` and `complete`.

15 `currents=selected`
Terminal currents to output. (See important note in the description below the parameter descriptions about saving currents by using probes).
Possible values are `all`, `nonlinear`, `selected`, and `none`.

16 `useprobes=no` — Use current probes when measuring terminal currents. (See important in the description below the parameter descriptions about saving currents by using probes).
Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

17 `useterms=index` Output terminal currents by specified option.
Possible values are `name` and `index`.

18 `redundant_currents=no`
If set to `yes`, save both currents through two terminal devices.
Possible values are `no` and `yes`.

19 `pwr=none` Power signals to create.
Possible values are `all`, `subckts`, `devices`, `total`, and `none`.

20 `saveahdlvars=selected`
AHDL variables to output.
Possible values are `all`, `selected`, and `allwithnodes`.

21 `ahdldomainerror=warning`
AHDL domain error handling selector. If `ahdldomainerror=error`, incorrect AHDL domain input is treated as an error; if `ahdldomainerror=warning`, incorrect AHDL domain input is treated as warning; if `ahdldomainerror=none`, AHDL domain error is ignored.
Possible values are `error`, `warning`, `none`, `erroriter`, and `warniter`.

22 `rawfmt=psfbin` Output raw data file format.
Possible values are `nutbin`, `nutascii`, `wsfbin`, `wsfascii`, `psfbin`, `psfascii`, `psfbinf`, `psfxl`, `awb`, `sst2`, `fsdb`, `fsdb5`, `wdf`, `uwi`, and `tr0ascii`.

23 `rawfile="%C:r.raw"` Output raw data file name.

24 `mdl_keep_cache_dir=0`
Flag to indicate whether or not to keep the MDL cache dir.

25 `xps_va_flow=0` Flag to indicate which xps variation analysis flow to use.

26 `precision="%g"` Format specification of double for psfascii. Example: "%15.12g" outputs 12 decimal digits in mantissa. The default value "%g" prints 6 decimal digits.

27 `uwifmt` User-defined output format. To specify multiple formats, use `:` as a delimiter. The option is valid only when waveform format is defined as uwi.

© 2003-2014 All Rights Reserved.

28 `uwilib` — Absolute path to the user-defined output format library. This option is used along with `uwifmt`. Use `:` as a delimiter to specify more than one library.

29 `exportfile` — oceanEval output file name.

30 `colonasdelimiter=none` — Determines whether to treat colon as a hierarchy delimiter. If `yes`, sst2 and uwi will treat colon as the hierarchy delimiter. If `no`, all formats will not treat colon as the hierarchy delimiter. Possible values are `no`, `yes`, and `none`.

31 `wfmaxsize=DBL_MAX` — Limits the maximum size of the output waveform. This option is valid when the uwi format is set.

32 `wfmaxtime=` — Limits the maximum period time of the output waveform. This option is valid when uwi format is set.

Convergence parameters

33 `homotopy=all` — Method used when there is no convergence on initial attempt of DC analysis. You can specify methods and their orders by defining vector setting such as `homotopy=[source ptran gmin]`.
Possible values are `none`, `gmin`, `source`, `dptran`, `ptran`, `arclength`, `all`, `fast_dptran`, `fast_ptran`, and `fast_gmin`.

34 `newton=normal` — Method used on DC analysis. You can specify methods `newton=[none | normal]`, and the default value is `normal`.
Possible values are `none` and `normal`.

35 `limit=dev` — Limiting algorithms to aid DC convergence.
Possible values are `delta`, `log`, `dev`, and `l2`.

36 `maxdeltav=0.3 V` — Maximum change allowed in voltage per Newton iteration when `limit=delta` (default `0.3 V`) or `limit=l2` (default `10 V`).

37 `gmethod=dev` — Stamp `gdev`, `gnode`, or `both` in the homotopy methods (other than `dptran`). See the detailed description below the parameter descriptions for more information.
Possible values are `dev`, `node`, and `both`.

© 2003-2014 All Rights Reserved.

38 `dptran_gmethod=node`
Stamp `gdev`, `gnode` or `both` in the `dptran` (homotopy) methods.
Possible values are `dev`, `node`, and `both`.

39 `gmin_start=1.0` Initial `gmin` in gmin-stepping (homotopy) methods.

40 `try_fast_op=yes` Speeds up the DC solution. For hard to converge designs, this feature fails and other methods are applied. In corner cases, this feature may have negative effects. If the DC analysis is unusually slow, the memory usage of the processes keeps increasing, or if DC analysis gets stuck even before homotopy methods start, try setting this option to `no`.
Possible values are `no` and `yes`.

41 `icversion=1` Convert initial condition to initial guess, when `.ic` statements exist in the netlist and there are no other options to set `IC` or `nodeset`.

Multithreading parameters

42 `multithread=off` Enable or disable multithreading capability. When multithreading is enabled 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 about using multithreading in the detailed description below the parameter descriptions).
Possible values are `off` and `on`.

43 `nthreads` Specifies the number of threads for multithreading.

Component parameters

44 `scalem=1.0` Model scaling factor.

45 `scale=1.0` Device instance scaling factor.

46 `scalefactor=1.0` ScaleFactor for Device Model Technology Scaling. The options parameter `scalefactor` enables device model providers to scale device technology independent of the design dimension scaling done by circuit designers. The resulting device instance scaling is defined by `scale * scalefactor`. If the foundry

© 2003-2014 All Rights Reserved.

uses a technology scale factor of 0.9 (`scalefactor=0.9`), and the circuit designer uses a design scale factor of 1e-6 (`scale=1e-6`), the compounded scaling of the device instance dimension is 0.9e-6. Unlike the options parameter `scale`, `scalefactor` cannot be used as a netlist parameter, and cannot be altered or used in `sweep` statements.

47 `ishrink=1` — Ishrink factor.

48 `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`, `spiceplus`, and `eldo`.

49 `approx=no` — Use approximate models. Difference between approximate and exact models is generally less public.
Possible values are `no` and `yes`.

50 `macromodels=no` — Determine whether circuit contains macromodels; at times, setting this parameter to `yes` helps improve performance.
Possible values are `no` and `yes`.

51 `auto_minductor=no` — Automatic insertion of missing mutual inductor coupling. For more information, see detailed description below the parameter descriptions.
Possible values are `no` and `yes`.

52 `GENK=no` — Automatic insertion of missing mutual inductor coupling. For more information, see detailed description below the parameter descriptions.
Possible values are `no` and `yes`.

53 `kmax=no` — Sets 1.0 as a maximum absolute value for coupling co-efficient of mutual inductors.
Possible values are `no` and `yes`.

54 `lcut=0 H` — Threshold inductance value for parasitic inductor reduction.

55 `kcut=0.0` — Threshold coupling co-efficient value for mutual inductor reduction, `0<=kcut<=1`.

© 2003-2014 All Rights Reserved.

56 `nportirreuse=yes` Reuse impulse responses data for all nport instances. If `no`, disable this feature.
Possible values are `no` and `yes`.

57 `nportirfiledir` The directory to which the nport impulse response file will be written. If an absolute path is not specified, the file will be written to `/home/<username>/.cadence/mmsim/`.

58 `nportcompress=yes`
Nport compression improves the efficiency of S-parameter simulation of large nport files when a certain percentage of the ports is unused, i.e., open or short circuited. Nport compression does not impact simulation accuracy. This option turns off compression if set to no and attempts to force compression if set to yes. If left unspecified, compression is on if N>=10 and the ratio of used ports is less than or equal to 0.8. Possible values are no, yes and unspecified..
Possible values are `no` and `yes`.

59 `nportunusedportgmin=0.0`
Default is 0, which leaves the port open-circuited. A small value loads open-circuited ports with a finite but large resistance. This introduces a small error in the response, but it induces losses which help obtain a passive response.

60 `nportunusedportrmin=0.0`
Default is 0, which leaves the port short-circuited. A small value will insert a small resistance in place of short circuited ports. This introduces a small error in the response, but it induces losses which help obtain a passive response.

61 `nportcompressfiledir`
The directory where the compressed nport sparameter file is written to. If unspecified, it is stored in outdir.

Noise parameters

62 `noiseon_inst=[...]` The list of instances to be considered as noisy throughout noise analysis, which include, noise, sp noise, pnoise and tran noise.

63 `noiseoff_inst=[...]` The list of instances not to be considered as noisy throughout noise analyses, which include noise, sp noise, pnoise and tran noise.

© 2003-2014 All Rights Reserved.

Error-checking parameters

64 `convdbg=none` — Debug convergence problems.
Possible values are `none`, `status`, and `detailed`.

65 `spice_montecarlo=no` — Monte Carlo analysis for SPICE public.
Possible values are `no` and `yes`.

66 `topcheck=full` — Check circuit topology for errors.
Possible values are `no`, `min`, `full`, `fixall`, `errmin`, and `errfull`.

67 `iccapcheck=yes` — Check if nodes with initial conditions have capacitive path to ground. IC for the node without capacitance is treated as nodeset.
Possible values are `no` and `yes`.

68 `ignshorts=no` — Silently ignore shorted components.
Possible values are `no` and `yes`.

69 `ndbrshort=0.0 Ω` — If the value of the resistor is less than the value of this option, the resistor will not be created in NDB period.

70 `vabsshort (V)` — When `vabsshort=value` is specified, all instance vsources with absolute `DC<=value` are shorted. The default value in APS is 1e-9 V.

71 `diagnose=no` — Print additional information that might help diagnose accuracy and convergence problems.
Possible values are `no`, `yes`, and `detailed`.

72 `diagnose_start=0 s` — Start time to enable the detailed diagnosis mode.

73 `diagnose_end (s)` — End time to enable the detailed diagnosis mode. Default is transient stop time.

74 `debugstepdrop=no` — Generates warnings on dramatic step size drop and notices when step size is recovered.
Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

75 `stepdropsize=100` The threshold size of the step drop to report a dramatical change warning.

76 `checklimitfile` File to which assert violations are written.

77 `dochecklimit=yes` Check asserts in the netlist.
Possible values are `no` and `yes`.

78 `checklimitdest=file`
Destination(s) where violations are written.
Possible values are `file`, `psf`, and `both`.

79 `checklimitdetails=no` Print detailed information of expression. Possible values are no and yes.

80 `probe_compatible=spectre`
Flag to enable or disable optimization for probe wildcard statement.
Possible values are `hspice` and `spectre`.

81 `rampsource=yes`
When performing dc hysteresis sweep, `rampsource` should be set to `no`.
Possible values are `no` and `yes`.

82 `devcheck_stat=yes`
Enable or disable output device-checking statistics.
Possible values are `no` and `yes`

83 `opptcheck=yes` Check operating point parameters against soft limits.
Possible values are `no` and `yes`.

Resistance parameters

84 `gmin=1e-12 S` Minimum conductance across each nonlinear device.

85 `gmindc=1e-12 S` Minimum conductance across each non-linear device in DC analysis. If `gmindc` is not specified, the value of `gmindc` will be equal to `gmin`. Default value is `1.0e-12`.

86 `gminallnode=0.0 S` Minimum conductance added for all nodes in DC analysis..

© 2003-2014 All Rights Reserved.

87 `gmin_check=max_v_only`
Specifies that effect of `gmin` should be reported if significant. Possible values are `no`, `max_v_only`, `max_only`, and `all`.

88 `fix_singular_matrix=no`
Fix matrix singular issue by inserting gmin. Possible values are `no`, `yes`, `print`, and `both`.

89 `fix_dc_floating_singular=no`
Fix dc floating node singular issue by inserting gmin. Possible values are `no` and `yes`.

90 `rforce=1` Ohm — Resistance used when forcing nodesets and node-based initial conditions.

91 `rthresh=0.001` Ohm — All instance resistors that have resistance smaller than global `rthresh` will use resistance form, unless their instance parameter or model parameter overwrites it. Note that resistance form of any resistor is set at the beginning of simulation and cannot be changed later, so t altering the value of `rthresh` is of no use. You will have to start a new run if you want a different `rthresh` for your circuit.

92 `rclamp (`Ohm`)` — When `rclamp=value` is given, all instance resistors with `R<value` are clamped to `value`. The default value is 0.001 ohm for bsource resistor.

93 `rcut=DBL_MAX (`Ohm`)`

Limit the resistor. Default is `DBL_MAX` for Spectre. For bsource resistor, the default is 1e12 for Spectre, and 1e18 for APS.

94 `ndbccut=0.0 F` — if the absolute value of parasitics capacitor is not greater than the absolute value of this option, the capacitor will be ignored in NDB period.

95 `rabsclamp (`Ohm`)` — When `rabsclamp=value` is specified, all instance resistors with absolute `R<value` are clamped to value.

96 `rabsshort (`Ohm`)` — When `rabsshort=value` is specified, all instance resistors with absolute `R<=value` are shorted. The default value in APS is `0.001` Ohm.

© 2003-2014 All Rights Reserved.

Quantity parameters

97 `value="V"` Default value quantity.

98 `flow="I"` Default flow quantity.

99 `quantities=no` Print quantities. If `quantities=min`, Spectre will print all defined quantities; if `quantities=full`, Spectre will also print a list of nodes and their quantities.
Possible values are `no`, `min`, and `full`.

Annotation parameters

100 `audit=detailed` Print time required by various parts of the simulator.
Possible values are `no`, `brief`, `detailed`, and `full`.

101 `inventory=detailed`
Print summary of components used.
Possible values are `no`, `brief`, `detailed`, and `full`.

102 `narrate=yes` Narrate the simulation.
Possible values are `no` and `yes`.

103 `debug=no` Give debugging messages.
Possible values are `no` or `yes`.

104 `info=yes` Give informational messages.
Possible values are `no` or `yes`.

105 `note=yes` Give notice messages.
Possible values are `no` or `yes`.

106 `maxnotes=5` Maximum number of times any notice is issued per analysis. Note that this option has no effect on notices issued as part of parsing the netlist. Use the `-maxnotes` command-line option to control the number of notices issued.

107 `emir_cmi_solver=1`
EMIR cmi solver selector. If set to 0, short the internal node. If set to 1, consider the internal nodes..

108 `stver_note=no` Give -stver suggestion warning for invalidate parameter.
Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

109 `warn=yes` — Display warning messages.
Possible values are `no` and `yes`.

110 `maxwarns=5` — Maximum number of times any warning message is issued per analysis. Note that this option has no effect on warnings issued as part of parsing the netlist. Use the `-maxwarns` command-line option to control the number of warnings issued.

111 `maxwarnstologfile=5`
Maximum number of times any warning message is printed to the log file per analysis. Note that this option has no effect on warnings printed as part of parsing the netlist. Use the `-maxwarnstolog` command-line option to control the number of warnings printed to the log file.

112 `maxnotestologfile=5`
Maximum number of times any notice message is printed to the log file per analysis. Note that this option has no effect on notices printed as part of parsing the netlist. Use the `-maxnotestolog` command-line option to control the number of all notices printed to the log file.

113 `error=yes` — Generate error messages.
Possible values are `no` and `yes`.

114 `digits=5` — Number of digits used when printing numbers.

115 `measdgt=0` — Number of decimal digits (in floating point numbers) in measurement output in mt0 format.

116 `notation=eng` — The notation to be used when printing real numbers to the screen. Possible values are `eng`, `sci`, and `float`.

117 `cols=80` — Width of screen in characters.

118 `colslog=80` — Width of log-file in characters.

119 `title` — Circuit title.

120 `simstat=basic` — Print simulation phase statistics report.
Possible values are `basic` and `detailed`.

© 2003-2014 All Rights Reserved.

Matrix parameters

121 `pivotdc=no` Use numeric pivoting on every iteration of DC analysis. Possible values are `no` and `yes`.

122 `dc_pivot_check=no` During DC analysis, the numeric pivoting is only performed when bad pivot is detected. Possible values are `no` and `yes`.

123 `checklimit_skip_subs=[...]`
Array of subcircuit masters or subcircuit master patterns to be skipped in device checking. Patterns can have any wildcard symbols.

124 `checklimit_skip_file`
File which contains the subcircuit masters or subcircuit master patterns to be skipped in device checking. Patterns can have any wildcard symbols.

125 `pivrel=0.001` Relative pivot threshold.

126 `pivabs=0` Absolute pivot threshold.

127 `preorder=partial` Try this option when simulation runs out of memory or if the simulation is unreasonably slow for the size of your design. It controls the amount of matrix pre-ordering that is done and may lead to much fewer matrix fill-ins in some cases. Known cases include designs with very large number of small resistors or large number of behavioral instances containing voltage based equations.
Possible values are `partial` and `full`.

128 `ignore_device_fpe=no`
Ignore Floating Point Exception in device loading for DC simulation.
Possible values are `no` and `yes`.

129 `limit_diag_pivot=yes`
If set to `yes`, there is a limit on the number of matrix fill-ins when selecting a pivot from a diagonal. For backward compatibility, set this option to `no`.
Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

130 `rebuild_matrix=no`
If set to `yes`, rebuild circuit matrix at the beginning of ac, dc, dcmatch, montecarlo, pz, stb, sweep, tdr, and tran analyses. This is to ensure consistent matrix ordering at the beginning of the analyses for consistent results. Notice that rebuild circuit matrix can result in performance overhead.
Possible values are `no` and `yes`.

Miscellaneous parameters

131 `ckptclock (s)` Clock time checkpoint period. Default is `1800s` for Spectre.

132 `param_topchange=yes`
If set to `yes`, Spectre will support parametric topology change caused by device internal node changes when the device, model, or netlist parameter value is altered.
Possible values are `no` and `yes`.

133 `redefinedparams=error`
Specify whether parameters can be redefined in the netlist. When set to `warning` or `ignore`, the simulator allows you to redefine parameters in the netlist. However, it honors only the last definition of the redefined parameter. Depending on the value that is set, the simulator displays warning messages for the redefined parameters or does not display any message. When set to `error`, the simulator does not allow you to redefine parameters in the netlist and displays an error message.
Possible values are `error`, `ignore`, `warning`, or `warn`.

134 `duplicateports=error`
Specify whether duplicate ports are allowed in the definition of the subcircuit. When set to `warning` or `ignore`, the duplicate ports are shorted. Depending on the value that is set, the simulator displays warning messages for the duplicate ports or does not display any message. When set to `error`, the simulator does not allow duplicate ports in the definition of the subcircuit and displays an error message. Possible values are `error`, `ignore`, and `warning`.

135 `duplicate_subckt=error`
Specify whether duplicate subcircuit definitions are allowed. When set to `warning` or `ignore`, the simulator allows duplicate subcircuit definitions. However, it honors only the last subcircuit

© 2003-2014 All Rights Reserved.

definition. Depending on the value that is set, the simulator displays warning messages for the duplicate subcircuit definitions or does not display any message. When set to `error`, the simulator does not allow duplicate subcircuit definitions and displays an error message. Possible values are `error`, `ignore`, and `warning`.

136 `duplicateinstance=error`

Specify whether duplicate instance definitions are allowed. When set to `ignore` or `warning`, the simulator allows duplicate instance definitions. However, it honors only the last instance definition. Depending on the value that is set, the simulator displays warning messages for the duplicate instance definitions or does not display any message. When set to `error`, the simulator does not allow duplicate instance definitions and displays an error message. Possible values are `error`, `ignore`, and `warning`.

137 `duplicatemodel=error`

Specify whether duplicate model definitions are allowed. When set to `ignore` or `warning`, the simulator allows duplicate model definitions. However, it honors only the last model definition. Depending on the value that is set, the simulator displays warning messages for the duplicate model definitions or does not display any message. When set to `error`, the simulator does not allow duplicate model definitions and displays an error message. Possible values are `error`, `ignore`, and `warning`.

138 `warning_limit=5`

The maximum number of times the warnings specified in immediately following the `warning_id` parameter should be displayed.

139 `use_veriloga=0` Determine whether to take Verilog-A as high priority. If set to `1`, the Verilog-A model takes priority over the other subckt/models. If set to `0`, the Verilog-A model does not take priority over the other subckt/models.

140 `warning_change_severity=warning`

Change the severity of warning messages specified in the immediately following `warning_id`.
Possible values are `error`, `warning`, and `notice`.

© 2003-2014 All Rights Reserved.

141 `warning_id=[...]` Vector of warning message identifiers, such as [SPECTRE-16462 SPECTRE-16684]. Used in conjunction with warning_limit or warning_change_severity.

142 `preserve_master=[...]` Preserve masters while running APS public.

143 `preserve_inst` Preserve instance while running APS.

144 `preserve_subckt=[...]` Preserve subcircuits while running APS.

145 `preserve_assert=lazy` Whether to execute the preservation for voltage check for assert statements. If the value is `lazy`, only execute the preservation for current checking; otherwise it will execute complete preservation. Possible values are `lazy` and `force`.

146 `enable_pre_ver=no` If `yes`, the old model versions are activated. Possible values are `no` and `yes`.

147 `soa_warn=yes` If `soa_warn` is `no`, SOA warnings will be turned off. Possible values are `no` and `yes`

148 `cmi_mos_cap=no` If `yes`, Mos_cap is activated if d/s/b are connected together. Possible values are `no` and `yes`.

149 `cmi_opts=[...]` The option for customer cmi.

150 `parasitics` Set parasitics for RC reduction. To use it as scoped option, command-line option `+mts` should be specified, and the option has higher priority than the value from command-line option `+parasitics`. If you use it as global option, the command-line option `+parasitics` has higher priority.

Sensitivity parameters

151 `sensfile` Output sensitivity data file name.

152 `sensformat=tabular` Format of sensitivity data. Possible values are `tabular` and `list`.

153 `senstype=partial` Type of sensitivity being calculated. Possible values are `partial` and `normalized`.

© 2003-2014 All Rights Reserved.

154 `sensfileonly=no` Enable or disable raw output of sensitivity results.
Possible values are `no` and `yes`.

155 `sensbinparam=no` Sensitivity for binning models.
Possible values are `no`, `uncorrelated`, and `correlated`.

156 `paramrangefile` Parameter range file.

Performance parameters

157 `minr=0.0` All parasitic resistors inside devices less than global `minr` will be removed. The order of checking devices is the follows: 1. Check if resistors are smaller than local `minr`. If yes, check if it is a MOSFET or BJT. If it is a MOSFET, drop the resistor, if it is BJT, clamp to the `minr` value, and give a warning message for both cases.
2. Check global `minr`. All Parasitic resistors less than global `minr` are removed and a warning message is issued.
3. If the resistor is not removed and is smaller than 0.001, issue a warning.

158 `short_bjtr=0.0` All parasitic resistors inside bipolar devices less than global `short_bjtr` will be removed. This option is applied to all bipolar junction transistors.

159 `verilogalang=relax`
AHDL Verilog-A language syntax check mode selector. If `verilogalang=strict`, show archaic syntax input as an error during Verilog-A parsing; if `verilogalang=relax`, show archaic syntax input as a warning during Verilog-A parsing.
Possible values are `strict` and `relax`.

160 `minstepversion=1` Minstep algorithm flow control. If `minstepversion=0`, a big jump is taken when minstep is reached. If `minstepversion=1`, a forward/backward search algorithm is taken to find a converged solution at small step size. By default, `minstepversion=1`.

161`try_hard_in_disaster=yes`
When `minstepversion=1`, this option means whether or not the simulator should try hard to search for a converged solution in disaster stage.
Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

162 `max_minstep_nonconv=100`
The maximum number of convergence failures allowed below minstep within 5% of stop time.

163 `max_approach_minstep=100`
The maximum number of times allowed to approach minstep within 5% of stop time.

164 `max_consecutive_minstep=1000`
The maximum number of consecutive steps allowed to be less than 10*minstep.

165 `autostop=no` If `yes`, tran analysis is terminated when all event-type measurement expressions have been evaluated. Event-type expressions use thresholding, event, or delay type functions. If the value is `spice`, autostop is consistent with spice simulator. Possible values are `no`, `yes`, and `spice`.

Model parameters

166 `soft_bin=singlemodels`
If set to `singlemodels`, it is used only on non-binned models. Possible values are `singlemodels`, `no`, and `allmodels`.

167 `ignore_unsupported_altergroup_constructs=no`
If set to `yes`, ignore unsupported statements in altergroup, such as save and export statements.
Possible values are `no` and `yes`.

168 `tmipath` Location of TMI shared object libraries to be used by tmiBsim4 models.

169 `tmiflag=0` Activate TMI flow. By default, TMI flow is not activated.

170 `tmevthmod=0` TSMC constant vth calculation. By default it is not activated.

171 `tmiage=0` Activate TMI Aging model flow. By default, it is not activated.

172 `tmiinput` Input file name, including the full path to read back TMI information for aging model.

173`tmioutput` Output file name about aging model.

© 2003-2014 All Rights Reserved.

174 `tmisave=1` Enable or disable aging information to be saved.

175 `etmiusrinput` This option can be set in model files or netlists.

176 `degfile` Output file name about aging model with text format.

177 `tmisort` Sort aging information.

178 `print_including=no`
Enable Spectre/UltraSim to print including information in `include` statement.
Possible values are `no` and `yes`.

179 `annotateonalter=yes`
Annotate on all alter statements.
Possible values are `no` and `yes`.

180 `reelaborateonalter=force`
Whether re-elaborate for every `alter` statement.
Possible values are `lazy` and `force`.

181 `scale_redefined=ignore`
Disallow redefining of the option `scale` in netlist.
Possible values are `error`, `ignore`, and `warning`.

182 `printstep=no` Enable Spectre to print results by equal step defined in `.tran` statement.
Possible values are `no` and `yes`.

183 `inlinesubcktcurrent=device`
Save inline subcircuit terminal current as inline device current or subcircuit current. By default, inline device current is saved.
Possible values are `device` and `subckt`.

184 `nonconv_topnum=10`
Top number of non-convergence nodes to be printed.

185 `dotprobefmt=flat` Print `.probe` signal with original name or hierarchical name.
Possible values are `flat` and `hier`.

186 `dcmmod=0` DCMismatch analysis version selector. If set to 1 or 3, uses Bsim short-channel mismatch equation for BSIM3 and BSIM4 devices; if set to 2 or 3, provides compatibility with Monte Carlo analysis.

© 2003-2014 All Rights Reserved.

187 `vthmod=std` Vth output selector. `std` outputs model equation Vth; `vthcc` outputs constant current Vth and may impact simulation performance.
Possible values are `std` and `vthcc`.

188 `ivthn=0.0 A` NMOS Vth current parameter.

189 `ivthp=0.0 A` PMOS Vth current parameter.

190 `ivthw=0.0 m` Width offset for constant current Vth.

191 `ivthl=0.0 m` Length offset for constant current Vth.

192 `ivth_vdsmin=0.05 V`
Minimum Vds in constant current Vth calculation.

193 `gform_vcr` Use Gform for VCR device. Specify `1` for yes and `0` for no.
90 `mdlthresholds=exact`
When set to `exact`, certain functions in MDL, for example, `cross()`, control the transient time-step to place a time-point at a threshold crossing. This can improve accuracy. However, for applications such as cell characterization an interpolated value can give the required accuracy with better performance.
Possible values are `exact` and `interpolated`.

194 `mdlthresholds=exact`
When set to `exact`, certain functions in MDL, for example, cross(), control the transient time-step to place a time-point at a threshold crossing. This can improve accuracy. However for applications, such as cell characterization an interpolated value can give the required accuracy with better performance.
Possible values are `exact` and `interpolated`.

195 `dio_allow_scaling=no`
Use this flag to enable the `SCALE` parameter for diode mode.
Possible values are `no` and `yes`.

196 `flashcell_model=atmel`
Used to choose the flash core cell model.
Possible values are `atmel`, `dallas`, `micron`, and `spansion`.

© 2003-2014 All Rights Reserved.

Stitching parameters

197 `spf` — This option specifies the DSPF file to be stitched and its stitching scope. The syntax is `spf="scope filename"`. The scope can be a subcircuit or an instance. Use "`spectre -h stitch`" for more information.

198 `dpf` — This option specifies the DPF file to be stitched and its stitching scope. The syntax is `dpf="scope filename"`. The scope can be a subcircuit or an instance. Use "`spectre -h stitch`" for more information.

199 `spef` — This option specifies the SPEF file to be stitched and its stitching scope. The syntax is `spef="scope filename"`. The scope can be a subcircuit or an instance. Use "`spectre -h stitch`" for more information.

200 `spfscale=1.0` — This option specifies the scaling factor of all the elements in the parasitic files (DSPF/SPEF/DPF). Use the `spectre -h stitch` command-line option for more details.

201 `spfxtorprefix` — This option specifies the prefix in the device or net names in the DSPF/SPEF/DPF file. The syntax is `spfxtorprefix="substring [replace_substring]"`. Use "spectre -h stitch" for more information.

202 `speftriplet=2` — This option specifies the value to be used for stitching in the SPEF file. It is effective only when the values in the SPEF file are represented by triplets (for example, `0.325:0.41:0.495`). Possible values are `1`, `2`, and `3`.

203 `spfswapterm` — This option specifies the swappable terminals of a subcircuit macro-model. The syntax is `spfswapterm="terminal1 terminal2 subcktname"`. Use "`spectre -h stitch`" for more information.

204 `spfaliasterm` — At times, the terminal names of devices in DSPF/SPEF/DPF files are different from those in the simulation model library. This happens in technology nodes that use subcircuits to model devices. The syntax is `spfaliasterm="<model|subckt> <prelayout_term1>=<spf_alias1> <prelayout_term2>=<spf_alias2> ... <prelayout_termN>=<spf_aliasN>"`. Multiple statements

© 2003-2014 All Rights Reserved.

	are supported. Use "`spectre -h stitch`" for more information.
205 `spfcnet`	This option specifies the net that has its total capacitance stitched. All other parasitic components such as parasitic resistors that are associated with this net are ignored. The complete hierarchical names are required and multiple statements are supported. Wildcards are supported. Use "`spectre -h stitch`" for more information.
206 `spfrcnet`	This option specifies the name of the net to be stitched with parasitic resistors and capacitors. The other nets are stitched with lumped total capacitances. Multiple statements are supported. Wildcards are supported and you can specify as many nets as needed. Complete hierarchical net names are required. Use "`spectre -h stitch`" for more information.
207 `spfnetcmin=0`	This option allows you to select the net for stitching by the value of its total node capacitance. If a net's total node capacitance exceeds `spfnetcmin`, all parasitics associated with the net are stitched correctly; otherwise, only the total capacitance is added to the net node. Use "`spectre -h stitch`" for more information.
208 `spfskipnetfile`	This option allows you to specify the nets to be skipped as a list in a text file. The syntax is `spfskipnetfile="filename"`. Only one file can be specified. The format in the file is one line per net. Use "`spectre -h stitch`" for more information.
209 `spfmsglimit`	This option specifies the maximum number of messages to be printed in the `spfrpt` file. The syntax is `spfmsglimit="number STITCH-ID_1 STITCH-ID_2"`. Use "`spectre -h stitch`" for more details.
210 `spfskipnet`	This option specifies the net to be skipped for stitching, that is, all parasitic components of the net are not stitched. Wildcards are supported. You can specify multiple `spfskipnet` statements . Use "`spectre -h stitch`" for more information.
211 `spfcnetfile`	This option has the same functionality as `spfcnet`, except that it accepts a text file in which all the C-only stitched nets are listed. Only one file can be specified. The syntax is `spfcnetfile="filename"`. The format of the file requires

© 2003-2014 All Rights Reserved.

you to specify one line per net. Use "`spectre -h stitch`" for more information.

212 `spfrcnetfile` This option has the same functionality as `spfrcnet`, except that it accepts a text file in which all the RC stitched nets are specified. Only one file can be specified. The syntax is `spfrcnetfile="filename"`. The format of the file requires you to specify one line per net. Use "`spectre -h stitch`" for more details.

213 `portmismatch_severity=error`
If set, ignore/warn/error out when ports number mismatch for subckt.
Possible values are `error`, `ignore`, and `warning`.

214 `implicit_subckt_param=no` Allows the implicit parameter to be declared for the spectre format. Possible values are `no` and `yes`.

215 `mdltrigtargmode=cross` This option specifies how to evaluate the trig and targ of the hspice and the output format. If mdltrigtargmode = `deltax`, we will use the deltax function to evaluate the trig and targ function. If mdltrigtargmode = `cross`, we will use the cross function to evaluate the trig and targ function. If mdltrigtargmode = `details`, we will ouput the trig and targ middle results. Possible values are `deltax`, `cross`, and `details`.

216 `rcnet_short_port=0`
Short RCNet's ports. Default is no.

217 `macro_mos=[...]` The list of subcircuit names, which is a macro model.

218 `cmi_mosfet=[...]` The list of device names that are treated as mosfet.

219 `hier_delimiter="."` Used to set hierarchical delimiter. Length of `hier_delimiter` should not be longer than 1, except the leader escape character.

220 `hier_ambiguity=strict`
The default value is `strict`. If set to `lower`, the simulator will partially flatten the hier name from the back to check if an object can be found. `Lower` is not recommended.
Possible values are `strict` and `lower`.

© 2003-2014 All Rights Reserved.

221 `skip=none` Skip simulating the specified subckt instances. For now, `cut` is supported, which will leave the subckt ports disconnected. Possible values are `none`, `load`, and `cut`.

222 `etmiflag=0` Indicate whether the TMI package includes the fastAge flow. By default, this parameter is not enabled. The version after 2.0104 will have fastAge.

223 `savefilter=none` Flag to enable filter internal RC nodes for probe/save statement. Possible values are `none` and `rc`.

224 `dspf_nodename=original`

When `original`, the current behavior, .connect, is created for nodes cut by DSPF translator. When `net_only`, it is the only net name in DSPF file that is used to match the wild card in .probe/ .ic statements. The hierarchical structure of the net name is restored like a pre-layout netlist. When `all`, the net name in DSPF file is stored as pre-layout node name, and the nodes in DSPF file are preserved like the current behavior. The wild card in .probe/.ic will match all the nodes and net names. Possible values are `original`, `net_only`, and `all`.

225 `measoutput=value` When set to `detail`, some additional parameters in the measure statement will be output to the .measure file. Possible values are `value` and `detail`.

226 `mc_scalarfile_sortby=analysis`
This option decides how scalarfile was printed. Default value is analysis.
Possible values are `analysis` and `iterator`.

227 `meassort=yes` Sorts the output value in the .measure file. Possible values are `no` and `yes`.

228 `mdlpost_signal_insensitive=no`
If set, the signal is found to be insensitive. Possible values are `no` and `yes`.

229 `detect_delta_gate=yes`Whether the delta gate resistors were detected and converted to a delta_gate device. Default value is `yes`. Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

230 `wfdebug=none` Flag to store the measurement signal in a raw file. Possible values are `none` and `measure`.

231 `saveports=no` If set to `yes`, the wildcard will match the subckt port for .probe and save statement. Possible values are `no` and `yes`.

232 `duplicate_module=error`

Specifies whether the duplicate module definitions are allowed. When set to `warning` or `ignore`, the simulator allows duplicate module definitions. However, it honors only the last module definition. Depending on the value that is set, the simulator decides whether to display warning messages for the duplicate module definitions or not. When set to `error`, the simulator does not allow the duplicate module definitions and displays an error message. Possible values are `error`, `ignore`, and `warning`.

233 `modmonte=0` Monte Carlo analysis for SPICE public.
Possible values are `0` and `1`.

234 `sx_factor=1` New scale factor.

235 `memcell_subckt=0` NULL. Possible values are `0` and `1`.

236 `memcell_gnode=0` NULL.
Possible values are `0` and `1`.

237 `ms_vpn` The virtual power node names, globally.

238 `ms_vpnv=0 V` The voltage value for the virtual power network sources in scope.

239 `ms_vgnd` The virtual ground node names, globally.

240 `ms_vgndv=0 V` The voltage value for the virtual power network grounds in scope.

241 `mtlinereuse=yes` This parameter defines whether to reuse RLGC data for all mtline instances. If `no`, this feature is disabled. Possible values are `no` and `yes`.

242 `mtlinecachedir` The directory in which the mtline RLGC data file will be written. If an absolute path is not given, the file will be written to /home/<username>/.cadence/mmsim/.

© 2003-2014 All Rights Reserved.

243 `disable_save_preserve=no`
Disable preserve by save statements. no - Do not disable save preserve. yes - Disable save preserve.
Possible values are `no` and `yes`.

244 `sim_stop_mode=0` If 1, when there is .print in the net list, the autostop will be ignored. If the value is 0, autostop will take effect, the default value is 0.
Possible values are `0` and `1`.

245 `separatemeasfile=0`
If 1, each run have separated measure file, if 2, measure file will have simple format.
Possible values are `0`, `1`, and `2`.

246 `keep_cmi=0` If set to 1, the cmi data will be reserved even when the specified instance and its children are converted to table models.
Possible values are `0` and `1`.

247 `saveExprProbeMeas=yes`
When saveProbeMeas=no, Do not save expression probe into measure file.
Possible values are `no` and `yes`.

248 `checkfoundpointaccuracy=yes`
In MDL buildin function, if the N point meet the conditions, if set yes, check next point have better accuracy .
Possible values are `no` and `yes`.

249 `check_format=xml` Determine the format of the checker violation report.
Possible values are `xml` and `text`.

Important note about accuracy related to useprobes or subcktiprobes

An iprobe should be inserted to get the accurate current of subcircuit or device terminal. For the subcircuit terminal, if you specify `save sub1:1`, iprobe is inserted automatically. On the other hand, for the device terminal `M1:g`, setting `useprobes=yes` triggers iprobe insertion to guarantee the current accuracy.

© 2003-2014 All Rights Reserved.

Important considerations for `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 the value of `reltol`.
- The following devices always use probes to save currents (even with `useprobes=no`): port, delay, switch, hbt, transformer, core, winding, fourier, d2a, a2d, a2ao, and 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`:

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

This definition is not scale free. When `senstype` is set to `normalized`, the sensitivity being calculated is the normalized sensitivity:

$$S(F \text{ w.r.t. } p) = \frac{d \ln F}{d \ln p} = \frac{p}{F}\frac{dF}{dp} = \frac{p}{F} D(F \text{ w.r.t. } p)$$

When either `F` or `p` take a zero value, the above normalized definition no longer provides a useful measure. In this case, the following two semi-normalized sensitivities are used:

$$S(F \text{ w.r.t. } p) = \frac{dF}{d \ln p} = p\frac{dF}{dp} = p\, D(F \text{ w.r.t. } p) \quad \text{if } F = 0$$

And:

$$S(F \text{ w.r.t. } p) = \frac{d \ln F}{dp} = \frac{1}{F}\frac{dF}{dp} = \frac{1}{F} D(F \text{ w.r.t. } p) \quad \text{if } p = 0$$

When both `F` and `p` are zero, partial sensitivity is used.

topcheck parameter

- When `topcheck=full`, the topology check is performed and `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.

© 2003-2014 All Rights Reserved.

- Use `topcheck=fixall` to attach `gmin` to all types of isolated nodes, including the ones detected by the heuristic topology check.
- When `topcheck=min`, the topology check is performed and `gmin` is inserted between isolated nodes and ground. A heuristic topology check is not performed.
- When `topcheck=no`, the topology check is not performed.
- `topcheck=errmin` (`topcheck=errfull`) is similar to `topcheck=min` (`topcheck=full`) but the simulation will stop if floating nodes are found.

Important considerations for using multithreading

- 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 is not applied to all instances of that primitive.
- Multithreading can be turned on/off using the command-line option, environment variable `SPECTRE_DEFAULTS` setting, or by using the `multithread` parameter in the options statement from the input file. The command-line option takes priority over the `SPECTRE_DEFAULTS` environment variable setting. In addition, the latter takes priority over the setting in the netlist options statement.
- 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. Because 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 the same result may not be reproducible when multithreading is used.
- Multithreading works best when the following capabilities are not used: `useprobes=yes`, save-current/SOA/alarm for multithreaded devices.
- When using device multithreading on hyperthreading enabled system:
  - allows one threads for each physical processor.
  - Because the device evaluation is almost exclusively floating point computation and each physical processor still has one floating point unit, each can handle one device evaluation at a time. Allowing additional thread(s) for device evaluation on the same physical processor will not have any benefit.
  - On a multi-processor system with hyperthreading enabled, device multithreading would allow an extra thread for each additional physical processor. Multithreading performance not only depends on the simulator but also on how well the operating system manages multiple threads and multiple processes.

© 2003-2014 All Rights Reserved.

gmethod parameter

- The parameter controls how conductance is stamped in the homotopy methods (other than `dptran`).
- If `gmethod=gnode` the conductance is added from node to ground. In case of `gmethod=dev` the conductance is stamped in the devices.

dptran_gmethod parameter

This parameter controls how conductance is stamped in the `dptran` (homotopy) method. See `gmethod` for more information on how this option affects circuits.

auto_minductor parameter

When this parameter is set to `yes`, the simulator automatically calculates the missing second-order coupling by multiplying the two first-order coefficients. This calculation is only an estimation and may not be correct for many geometries.

For example, consider two mutual inductors `K1` and `K2`:

```
K1 mutual_inductor coupling=.65 ind1=L1 ind2=L2
K2 mutual_inductor coupling=.65 ind1=L2 ind2=L3
```

In this example, Spectre automatically inserts the coupling between `L1` and `L3`,if missing, and the coupling co-efficient is `0.65*0.65=0.4225`.

save parameter

If `save=nooutput` is specified, Spectre does not print any simulation results other than measurements. For `save=nooutput` to have effect, it should be defined globally.

If `save=selected` is specified, and there is no save statement, `save=allpub` is used instead.

© 2003-2014 All Rights Reserved.

Parameter Index

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

© 2003-2014
All Rights Reserved.

© 2003-2014
All Rights Reserved.

© 2003-2014
All Rights Reserved.

© 2003-2014
All Rights Reserved.

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. PAC is a small-signal analysis similar to AC analysis, except that 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, which is not the case when linearizing about a DC operating point because linear time-invariant circuits do not exhibit frequency translation. In addition, 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 part of the PSS analysis, the periodically 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 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, and PNoise, can follow a PSS analysis.

Modulated small signal measurements are possible by using the Analog Design Environment (ADE). The `modulated` option for PAC and the other modulated parameters are set by Artist. PAC analyses with this option produces results that could have limited use outside such an 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, see the *Spectre RF User Guide*.

Unlike AC analysis, PAC analysis can print the time-domain simulation results by specifying the `outputperiod` parameter. In addition, unlike other analyses in Spectre, this analysis can only sweep frequency.

Definition

```
Name ... pac parameter=value ...
```


© 2003-2014 All Rights Reserved.

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=unspecified`
Specifies if the sweep frequency range is absolute frequency of input or if it is relative to the port harmonics. When the `unspecified` value is used, Spectre RF sweeps the absolute input source for non-PSP-driven cases; for other cases, Spectre RF sweeps relative to the port harmonics.
Possible values are `absolute`, `relative` and `unspecified`.

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

Sampled analysis parameters

12 `ptvtype=timeaveraged`
Specifies if the PTV analysis will be traditional or sampled under certain conditions.
Possible values are `timeaveraged` and `sampled`.

13 `sampleprobe` The crossing event at this port triggers the sampled small signal computation.

© 2003-2014 All Rights Reserved.

14 `thresholdvalue=0` Sampled measurement is done when the signal crosses this value.

15 `crossingdirection=all`
Specifies the transitions for which sampling must be done. Possible values are `all`, `rise`, `fall`, and `ignore`.

16 `maxsamples=16` Maximum number of sampled events to be processed during the sampled analysis.

17 `extrasampletimepoints=[...]`
Additional time points for sampled PTV analysis.

Output parameters

18 `sidebands=[...]` Array of relevant sidebands for the analysis.

19 `maxsideband=7` An alternative to the `sidebands` array specification, which automatically generates the array: [-maxsideband ... 0 ... +maxsideband]. For the shooting analysis, the default value is `7`. For HB small signal analysis, the default value is the `harms/maxharms` setting in the HB large signal analysis. It is ignored in HB small signal when it is larger than the `harms/maxharms` of large signal.

20 `freqaxis` Specifies whether the results should be printed as per the input frequency, the output frequency, or the absolute value of the output frequency. Default is `absout`.
Possible values are `absout`, `out`, and `in`.

21 `save` Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

22 `nestlvl` Levels of subcircuits to output.

23 `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.

24 `oscout=total` The type of output for oscillator simulation. The default value is `total` for the output of total modulation response from oscillator

© 2003-2014 All Rights Reserved.

simulation. Other values are `pm` for the output of phase-modulation response and `am` for the output of amplitude-modulation response.
Possible values are `total`, `pm`, and `am`.

Convergence parameters

25 `tolerance` — Relative tolerance for linear solver. The default value is `1.0e-9`.

26 `relativeTol` — Relative tolerance for harmonic balance-based linear solver. The default value is 1.0e-2.

27 `gear_order=2` — Gear order used for small-signal integration.

28 `solver=turbo` — Solver type.
Possible values are `std` and `turbo`.

29 `oscsolver=turbo` — Oscillator solver type. It is recommended to use `ira` for huge circuits.
Possible values are `std`, `turbo`, `ira`, and `direct`.

30 `lnsolver=gmres` — Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, and `gmres_cycle`.

31 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, and `recyclelong`.

32 `hbprecond_solver=autoset`

.Select a linear solver for the GMRES preconditioner. Default is `autoset`. With `autoset`, the simulator will automatically select the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when `autoset` is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulator's instructions may speedup subsequent analyses.

© 2003-2014 All Rights Reserved.

Possible values are `basicsolver`,`blocksolver` and `autoset`.

33 `krylov_size=200` The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching krylov_size iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase krylov_size if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

34 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, and `detailed_hb`.

35 `title` Analysis title.

Modulation conversion parameters

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

37 `inmodharmnum=1` Harmonic value for the PAC input source modulation.

38 `outmodharmvec=[...]` Harmonic list for the PAC output modulations.

39 `moduppersideband=1`
Index of the upper sideband included in the modulation of an output for PAC, or an input for PXF.

40 `modsource` Refer the output noise to this component.

41 `perturbation=linear`
The type of PAC analysis. Default is `linear` for normal PAC analysis. `im2ds` stands for im2 distortion summary and `ds` stands for distortion summary.
Possible values are `linear`, `ds`, `ip3`, `ip2`, `im2ds`, and `multiple_beat`.

42 `flin_out=0 Hz` Frequency of linear output signal.

© 2003-2014 All Rights Reserved.

43 `fim_out=0 Hz` Frequency of IM output signal.

44 `out1="NULL"` Output signal 1.

45 `out2="NULL"` Output signal 2.

46 `contriblist="NULL"` Array of device names for distortion summary. When `contriblist[""]`, distortion from each non-linear device is calculated.

47 `maxharm_nonlin=4` Maximum harmonics of input signal frequency induced by non-linear effect.

48 `rfmag=0` RF source magnitude.

49 `rfdbm=0` RF source dBm.

50 `rf1_src="NULL"` Array of RF1 source names for IP3/IP2/IM2.

51 `rf2_src="NULL"` Array of RF2 source names for IP3/IP2/IM2.

52 `rf_src="NULL"` Array of RF source names for triple beat analysis.

53 `freqs=NULL` Array of RF source frequencies for triple beat analysis.

54 `rfampls=NULL` RF source amplitudes; the units are dBm for ports, Voltage for v-sources and Ampere for i-sources.

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 sidebands K1, K2, ... Kn, the output frequency at each sideband is computed as `f(out)= f(in) + Ki * fund(pss)`, where `f(in)` represent the (possibly swept) input frequency, and `fund(pss)` represents the fundamental frequency used in the corresponding PSS analysis. Therefore, when analyzing a down-converting mixer, while sweeping the RF input frequency, the most relevant sideband for IF output is Ki= -1. When simulating an up-converting mixer, while sweeping IF input frequency, the most relevant sideband for RF output is Ki= 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

© 2003-2014 All Rights Reserved.

maxacfreq should be set in the PSS analysis. In 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 area 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 define sweep limits by specifying the end points or by providing the center value and span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. In addition, you can give a step size parameter (step, lin, log, or 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 corresponding to each parameter name indicates where to find the description of that parameter.

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

Periodic Distortion Analysis (pdisto)

Description

Quasi-periodic steady-state (QPSS) analysis computes circuit response with multiple fundamental frequencies using harmonic balance (in frequency domain) or shooting. QPSS can compute circuits' responses with closely spaced or incommensurate fundamentals, which cannot be resolved by PSS efficiently. The simulation time of QPSS analysis is independent of the time constants of the circuit. Also, QPSS analysis sets the circuit quasi-periodic operating point, which can then be used during a quasi-periodic time-varying small-signal analysis, such as QPAC, QPXF, QPSP, and QPNOISE.

Generally, harmonic balance (HB) is very efficient in simulating weak non-linear circuits while shooting is more suitable for computing a circuit response to several moderate input signals, in addition to a large signal. The large signal, which represents a LO or clock signal, is usually the one that causes the most nonlinearity or the largest response. A typical example is the intermodulation distortion measurements of a mixer with two closely spaced moderate input signals. HB is more efficient than shooting in handling frequency dependent components, such as delay, transmission line, and S-parameter data.

QPSS consists of three phases. First, an initial transient analysis with all moderate input signals suppressed is carried out. Second, a number of (at least 2) stabilizing iterations are run with all signals activated. Finally, the Newton method is followed.

When the shooting method is used, QPSS employs the Mixed Frequency Time (MFT) algorithm extended to multiple fundamental frequencies. For details of MFT algorithm, see `Steady-State Methods for Simulating Analog and Microwave Circuits,` by K. S. Kundert, J.K. White, and A. Sangiovanni-Vincentelli, Kluwer, Boston, 1990.

Similar to shooting in PSS, shooting in QPSS uses 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 `maxharms`. Given `maxharms=[k1 k2 ... kn]`, QPSS always treats k1 as the maximum harmonic of the large signal and the total number of integrations is `(2*k2+1)*(2*k3+1)*...*(2*kn+1)`. One consequence is that the efficiency of the algorithm depends significantly on the number of harmonics required to model the responses of moderate fundamentals. Another consequence is that 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.

© 2003-2014 All Rights Reserved.

QPSS inherits most of the PSS parameters and adds a few new ones. The most important ones are `funds` and `maxharms`. They replace the 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 the `fundname` parameter. In both shooting and HB QPSS analysis, the first fundamental is considered as the large signal. A few heuristics can be used for picking the large fundamental.

1. Pick the fundamental that is not a sinusoidal.
2. Pick the fundamental that causes the most nonlinearity.
3. Pick the fundamental that 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.

The semi-autonomous simulation is a special QPSS analysis combining the autonomous simulation and the QPSS. To perform the semi-autonomous simulation, you need to specify an initial frequency guess for the oscillator inside the circuit, and two oscillator terminals, similar to the autonomous simulation in PSS. For example:

```
myqpss (op on) qpss funds=[1.1GHz frf] maxharms=[5 5] tstab=1u flexbalance=yes
```

Note: The semi-autonomous simulation is only available in the frequency domain.

Definition

```
Name pdisto parameter=value ...
```

Parameters

QPSS fundamental parameters

1	`funds=[...]`	Array of fundamental frequency names for fundamentals to use in analysis.
2	`maxharms=[...]`	Array of number of harmonics of each fundamental to consider for each fundamental.
3	`selectharm`	Name of harmonics selection methods. The default is `diamond` when `maximorder` or `boundary` is set; otherwise, default is `box`. Possible values are `box`, `diamond`, `funnel`, and `axis`.

© 2003-2014 All Rights Reserved.

4	`evenodd=[...]`	Array of even, odd, or all strings for moderate tones to select harmonics.
5	`boundary`	Harmonic selection boundary.
6	`maximorder`	Maximum intermodulation order (same parameter as `boundary`).
7	`harmlist=[...]`	Array of harmonics indices.
8	`freqdivide`	Large signal frequency division.

Simulation interval parameters

9	`tstab=0.0 s`	Extra stabilization time after the onset of periodicity for independent sources.
10	`autotstab=no`	Activates automatic initial transient (tstab) in harmonic balance. When set to `yes`, the simulator decides whether to run tstab and for how long. Typically, the initial length of tstab is 50 periods, but may be longer depending on the type of circuit and its behavior. If steady-state is reached (or nearly reached), tstab will terminate early. Possible values are `no` and `yes`.
11	`autosteady=no`	Activates automatic steady state detection during initial transient (tstab) in harmonic balance. When steady state is reached (or nearly reached), tstab will terminate early. Applies only when tstab>0 or when `autotstab=yes`. Possible values are `no` and `yes`.
12	`autoharms=no`	Activates automatic harmonic number calculation in harmonic balance. Applies only if tstab>0 or when `autotstab=yes`. If steady-state is reached, Spectre does a spectrum analysis to calculate the optimal number of harmonics for HB. The minimum number of harmonics is specified by `maxharms`. If steady-state is not reached to sufficient tolerance, autoharms may be disabled. Possible values are `no` and `yes`.
13	`stabcycles=2`	Stabilization cycles with both large and moderate sources enabled.

© 2003-2014 All Rights Reserved.

14 `tstart=0.0 s` Initial transient analysis start time.

Time-step parameters

15 `maxstep (s)` Maximum time step. Default is derived from `errpreset`.

16 `maxacfreq` Maximum frequency requested in a subsequent periodic small-signal analysis. The default is derived from `errpreset` and `harms`. This parameter is valid only for shooting.

17 `step=0.001 period s`
Minimum time step that would be used solely to maintain the aesthetics of the results. This parameter is valid only for shooting.

Initial-condition parameters

18 `ic=all` The value to be used to set the initial condition.
Possible values are `dc`, `node`, `dev`, and `all`.

19 `skipdc=no` If set to `yes`, there is no DC analysis for initial transient.
Possible values are `no`, `yes`, and `sigrampup`.

20 `readic` File that contains initial condition.

21 `useprevic=no` If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns.
Possible values are `no`, `yes`, and `ns`.

Convergence parameters

22 `readns` File that contains an estimate of the initial transient solution.

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

Output parameters

24 `save` Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

© 2003-2014 All Rights Reserved.

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 printed (screen or file). Possible values are `no`, `screen`, `logfile`, and `rawfile`.
27 `skipstart=0 s`	The time to start skipping output data.
28 `skipstop=stop s`	The time to stop skipping output data.
29 `skipcount=1`	Save only one of every skipcount points.
30 `strobeperiod=0 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 `saveinit=no`	If set to `yes`, the waveforms for the initial transient before steady state are saved. Possible values are `no` and `yes`.

State-file parameters

33 `write`	File to which initial transient solution (before steady-state) is to be written.
34 `writefinal`	File to which final transient solution in steady-state is to be written. This parameter is now valid only for shooting.
35 `swapfile`	Temporary file to hold steady-state information. It 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 the analysis. This parameter is now valid only for shooting.

Integration method parameters

36 `method`	Integration method. Default is derived from `errpreset`. This parameter is valid only for shooting. Possible values are `euler`, `trap`, `traponly`, `gear2`, and `gear2only`.

© 2003-2014 All Rights Reserved.

Emir output parameters

37 `emirformat=none` — Format of the EM/IR database file.
Possible values are `none` and `vavo`.

38 `emirstart (s)` — EM/IR start time.

39 `emirstop (s)` — EM/IR stop time.

40 `emirfile` — Name of the EM/IR database file. Default is `%A_emir_vavo.db`. The file is output to raw directory.

Accuracy parameters

41 `errpreset` — Selects a reasonable collection of parameter settings.
Possible values are `liberal`, `moderate`, and `conservative`.

42 `relref` — Reference used for the relative convergence criteria. Default is derived from `errpreset`.
Possible values are `pointlocal`, `alllocal`, `sigglobal`, and `allglobal`.

43 `lteratio` — Ratio used to compute LTE tolerances from Newton tolerance. Default is derived from `errpreset`.

44 `lteminstep=0.0 s` — Local truncation error will be ignored if the step size is less than `lteminstep`.

45 `steadyratio` — Ratio used to compute steady state tolerances from LTE tolerance. Default is derived from `errpreset`.

46 `maxperiods` — Maximum number of iterations allowed before convergence is reached in shooting or harmonic balance Newton iteration. For PSS and QPSS, the default is 20 for driven circuits, and 50 for oscillators; For HB, the default is 100.

47 `lnsolver=gmres` — Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, and `gmres_cycle`.

48 `itres=1e-4 for shooting, 0.9 for HB` — Controls the residual for iterative solution of linearized matrix equation at each Newton iteration. Tightening the parameter can

© 2003-2014 All Rights Reserved.

help with the Newton convergence, but does not affect the result accuracy. The value should be between [0, 1].

49 `inexactNewton=no` Inexact Newton method.
Possible values are `no` and `yes`.

50 `finitediff` Options for finite difference method refinement after quasi-periodic shooting method. `finitediff` is changed from no to same grid automatically when `readqpss` and `writeqpss` are used to re-use QPSS results.
Possible values are `no`, `yes`, and `refine`.

Harmonic Balance parameters

51 `harmonicbalance=no`
Use Harmonic Balance engine instead of time-domain shooting.
Possible values are `no` and `yes`.

52 `flexbalance=no` Same parameter as `harmonicbalance`.
Possible values are `no` and `yes`.

53 `hbpartition_defs=[...]`
Define HB partitions.

54 `hbpartition_fundratios=[...]`
Specify HB partition fundamental frequency ratios.

55 `hbpartition_harms=[...]`
Specify HB partition harmonics.

56 `oversamplefactor=1`
Oversample device evaluations.

57 `oversample=[...]` Array of oversample factors for each tone. It overrides `oversamplefactor`.

58 `hbhomotopy=tone` Name of Harmonic Balance homotopy selection methods.
Possible values are `tstab`, `source`, `gsweep`, `tone`, and `inctone`.

59 `sweepic=none` IC extrapolation method in sweep HB analysis.
Possible values are `none`, `linear`, and `log`.

© 2003-2014 All Rights Reserved.

60 `gstart=1.e-7` Start conductance for hbhomotopy of `gsweep`.

61 `gstop=1.e-12` Stop conductance for hbhomotopy of `gsweep`.

62 `glog=5` Number of steps, log sweep for hbhomotopy of `gsweep`.

63 `backtracking=yes` This parameter is used to activate the backtracing utility of Newtons method. Default is `yes`.
Possible values are `no`, `yes`, and `forced`.

64 `krylov_size=10` The minimum iteration count of the linear matrix solver used in HB large-signal analysis. After reaching `krylov_size` iterations, the iteration is forced to terminate because of poor rate of convergence. Increase `krylov_size` if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

65 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `estimated`, `steps`, `iters`, `detailed`, `rejects`, `alliters`, `detailed_hb`, and `internal_hb`..

66 `annotateic=no` Degree of annotation for initial condition.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, `iters`, `detailed`, and `rejects`.

67 `title` Analysis title.

Newton parameters

68 `maxiters=5` Maximum number of iterations per time step.

69 `restart=no` Restart the DC/PSS/QPSS solution from scratch, if set to `yes`; if set to `no`, reuse the previous solution as initial guess; if set to `firstonly`, restart from scratch when it is first point of sweep (only supported in HB). The default value is `no` for HB and `yes` for shooting.
Possible values are `no`, `yes`, and `firstonly`.

© 2003-2014 All Rights Reserved.

Circuit age

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

71 `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. The results from shooting QPSS cannot be used in HB QPSS analysis and vice-versa.

72 `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. The results from shooting QPSS cannot be used in HB QPSS analysis and vice-versa.

Tstab save/restart parameters

73 `ckptperiod` Checkpoint the analysis periodically by using the specified period.

74 `saveperiod` Save the tran analysis periodically on the simulation time.

75 `saveperiodhistory=no` Maintains the history of saved files. If `yes`, stores all the saved files. Possible values are `no` and `yes`.

76 `saveclock (s)` Save the tran analysis periodically on the wall clock time. The default is 1800s for Spectre. This parameter is disabled in the APS mode by default.

77 `savetime=[...]` Save the analysis states into files on the specified time points.

78 `savefile` Save the analysis states into the specified file.

79 `recover` Specify the file to be restored.

80 `oscic=default` Oscillator IC method. It determines how the starting values for the oscillator are calculated. `oscic=lin` gives you an accurate initial value, but it takes time; `fastic` is fast, but it is less accurate. `oscic=skip` directly uses the frequency provided by you as the initial guess frequency. It is only for the two-tier

© 2003-2014 All Rights Reserved.

method.
Possible values are `default`, `lin`, `fastic`, and `skip`.

81 `xdbcompression="no"` Sets the automatic gain compression analysis. In automatic gain compression analysis, Spectre automatically sweeps the input excitation until the gain, as defined by the analysis parameter xdbgain, compresses by the amount specified by the analysis parameter xdblevel. In gain compression analysis, Spectre ouputs the hb solution at the calculated compression point only. Dependent analyses, such as hbnoise and hbac, are supported and calculated about the calculated compression level. Auxiliary output includes the gain and voltage/power compression curves. These outputs are available for analysis and post-processing in ADE. The possible values are `yes` and `no`. Default is `no`.

82 `xdblevel=1.0` Sets the gain compression level for compression analysis. The reference point for gain compression is the small-signal gain of the circuits, or as specified by the analysis parameter `xdbref`. Default is `1`.

83 `xdbgain="power"` Chooses between the voltage gain or transducer power gain as the target for compression point calculation. When `xdbgain=power`, the gain is defined as *G (dB) = Pload (dBm) - Pavailable (dBm)*. When `xdbgain=voltage`, the gain is defined as *G (dB) = dB20(|Vload|/|Vsource|)*. In both cases, Spectre sweeps the excitation source until *xdbref - G = xdblevel*, where the analysis parameter xdbref defines the reference level for compression calculation. Possible values are `power` and `voltage`. Default is `power`.

84 `xdbref="linear"` Sets the reference point for gain compression calculations. When `xdbref=linear`, spectre uses the small-signal gain as the reference. When `xdbref=max`, spectre uses the maximum observed gain as the reference. Possible values are `linear` and `max`. Default is `linear`.

85 `xdbsource` The instance name of the excitation source, which is swept automatically to reach the compression level. When `xdbgain=power`, the excitation source must be a port instance. When `xdbgain=voltage`, the excitation source can be a vsource instance or a port instance. Note that in the voltage gain

© 2003-2014 All Rights Reserved.

	calculation, *dB20(|Vload|/|Vsource|)*, when `xdbsource` is a port and `xdbgain=voltage`, Spectre interprets `Vsource` as the voltage across a matched load, according to the Spectre usual port element conventions.
86 `xdbload`	The instance name of the load termination. When `xdbgain` is `power`, `xdbload` can be a port, a resistor, or a current probe.
87 `xdbnodep`	The output terminals for voltage gain calculation when `xdbgain=voltage`. If either is left unspecified, the terminal is assumed to be the global ground.
88 `xdbnoden`	The output terminals for voltage gain calculation when `xdbgain=voltage`. If either is left unspecified, the terminal is assumed to be the global ground.
89 `xdbrefnode`	The reference node when `xdbload` is a current probe. The default is the ground node.
90 `xdbharm=[...]`	The Integer array which specifies the harmonic indexes of the output voltage or power component.
91 `xdbsteps=100`	The maximum number of steps for the compression point search. The simulator terminates if `xdbsteps` exceeds before the compression point is found. The default is `100`.
92 `xdbmax`	The maximum input power (or voltage) for the compression point search. Default is `30 dBm` when `xdbgain=power` and `10V` when `xdbgain=voltage`.
93 `xdbstart`	The starting input power (or voltage) for the compression point search. Default is (xdbmax-50) dBm when `xdbgain=power`, and xdbmax/1000 when `xdbgain=voltage`.
94 `xdbtol=0.01`	Sets the tolerance for compression analysis. This tolerance is used in compression curve fitting and calculating the compression point.
95 `xdbrapid="no"`	Sets the automatic gain compression analysis in rapid mode. In this mode, Spectre does not trace the compression curve and calculates only the compression point.

© 2003-2014 All Rights Reserved.

96 `xdbcpi` Sets the estimated input-referred compression point for rapid compression analysis.

97 `memoryestimate="no"` Sets the memory usage estimate for Harmonic Balance. If yes, a memory estimate is printed in the log file. You can use this memory estimate to plan the computing resources before submitting harmonic balance runs. In memory estimate mode, a short simulation is performed first, and the engine exits after printing the estimate in the log file without saving harmonic balance results. You must turn it off to perform an actual simulation. Memory estimation is not recommended for simulations that require less than 500MB approximately. For PSS analysis, memory estimate mode does not apply unless flexbalance=yes. The estimate applies only to large-signal analysis, and does not include subsequent noise or other small-signal simulations.

Most 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 determined automatically by the simulator. An important feature is that each input signal can be a composition of more than one source. 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. Omitting fundamental name in the `funds` parameter is an error that stops the simulation. If `maxharms` is not given, a warning message is issued, and the number of harmonics defaults to 1 for each of the fundamentals.

For QPSS analyses, the role of some PSS parameters is extended compared to their role in PSS analysis. In QPSS, 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 shooting or HB 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 set errpreset to `liberal`. If you want more accurate results, set `errpreset` to `moderate`. For most accurate results, set `errpreset` to `conservative`.

© 2003-2014 All Rights Reserved.

If you 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 in shooting.

Table 3-2 Parameter defaults as a function of errpreset

errpreset	reltol	relref	method	lteratio	maxstep
liberal	1e-3	sigglobal	gear2only	3.5	clock period/80
moderate	1e-4	sigglobal	gear2only	3.5	clock period/100
conservative	1e-5	sigglobal	gear2only	*	clock period/200

* : `lteratio=10.0` for conservative `errpreset` by default. However, when the specified reltol `<= 1e-4*10.0/3.5`, `lteratio` is set to `3.5`.

The new `errpreset` settings include a new default `reltol`, which is an enforced upper limit for appropriate setting. An increase in `reltol` above the default value is ignored by the simulator. You 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 not 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 are used

For HB, only `reltol` is affected by `errpreset` and this effect is the same as that in shooting. However, `lteratio` remains 3.5 and `steadyratio` remains 1 with all values of `errpreset`.

With parameter `hbhomotopy`, you can specify harmonic balance homotopy selection methods. The possible values of parameter `hbhomotopy` are as follows:

- `hbhomotopy=tstab`: Simulator runs a transient analysis and generates an initial guess for the harmonic balance analysis; it is recommended for nonlinear circuits or circuits with frequency dividers.
- `hbhomotopy=source`: For driven circuits, simulator ignores tstab and accordingly increases the source power level; for oscillators, simulator accordingly adjusts the probe magnitude until the probe has no effect on the oscillators. It is recommended for strongly nonlinear or high Q circuits
- `hbhomotopy=tone`: This method is only valid for multi-tone circuit. Simulator first solves a single-tone circuit by turning off all the tones except the first one, and then solves the multi-tone circuit by restoring all the tones and using the single-tone solution as its initial guess; it is recommended for multitone simulation with a strong first tone.

© 2003-2014 All Rights Reserved.

- `hbhomotopy=inctone`: Simulator firstly solves a single tone, then turns on moderate tones incrementally till all tones are enabled. It is recommended for circuits with one strong large tone.
- `hbhomotopy=gsweep`: A resistor, whose conductance is g, is connected with each node, the sweep of g is controlled by gstart, gstop, and glog; it is recommended for circuits containing high-impedance or quasi-floating nodes.

Parameter Index

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

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

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 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 by using the `iprobe` parameter. For input-referred noise, use 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 computes 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, 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(input)| = |f(out) + refsideband * 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.

© 2003-2014 All Rights Reserved.

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. Therefore, 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(No^2/G^2)

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

NF = 10*log10(F)

Fdsb = (No^2 - Nl^2)/(Ns^2+Nsi^2)

NFdsb = 10*log10(Fdsb)

© 2003-2014 All Rights Reserved.

Fieee = (No^2 - Nl^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.

In a phase noise analysis for an oscillator, the line width, which is also known as the corner frequency, is defined as either the full width at half maximum (FWHM), or as twice the half power (-3dB) width (HW). In the absence of 1/f noise and ignoring any noise floor, the phase noise spectrum satisfies the Lorentzian equation:

```
L(f) = (1/pi) * [ pi * c * fosc^2 ] / [(pi * c * fosc^2)^2 + f^2],
```

Where, c is a constant that defines the phase noise characteristics of the oscillator, fosc is the fundamental frequency of the oscillator, and f is the offset frequency of the oscillator. Therefore:

```
line width := FWHM = 2* HW = 2 * pi * c * fosc^2.
```

Note: 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 specify the terminals, 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.

© 2003-2014 All Rights Reserved.

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=unspecified`
Specifies if the sweep frequency range is absolute frequency of input or if it is relative to the port harmonics. When the `unspecified` value is used, Spectre RF sweeps the absolute input source for non-PSP-driven cases; for other cases, Spectre RF sweeps relative to the port harmonics.
Possible values are `absolute`, `relative`, and `unspecified`.

11 `relharmnum=1` Harmonic 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.

Sampled analysis parameters

15 `thresholdvalue=0` Sampled measurement is done when the signal crosses this value.

16 `crossingdirection=all`
Specifies the transitions for which sampling must be done.
Possible values are `all`, `rise`, `fall`, and `ignore`.

17 `maxsamples=16` Maximum number of sampled events to be processed during the sampled analysis.

18 `sampleratio=1` The ratio between sampled frequency and fund frequency (sampled frequency/fund frequency).

© 2003-2014 All Rights Reserved.

19 `externalsourcedata`
Name of PXF analysis that provides information to compute the contribution of external jitter sources.

Output parameters

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

21 `maxsideband=7`
In shooting pnoise, the parameter determines the maximum sideband included when computing noise that is either up-converted or down-converted to the output by the periodic drive signal. In HB pnoise, the parameter determines the size of the small signal system when the HB pnoise is performed. This parameter is critical for the accuracy of the HB pnoise analysis; using a small maxsideband may cause accuracy loss.

The default value for the shooting pnoise is 7. For HB pnoise, the default is the `harms/maxharms` setting in the HB large signal analysis.

22 `sidebands=[...]`
Array of relevant sidebands for the analysis.

23 `save`
Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

24 `nestlvl`
Levels of subcircuits to output.

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

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

27 `noiseskipcount=-1`
Calculate time-domain noise on only one of every noiseskipcount time points. When < 0, the parameter is ignored.

© 2003-2014 All Rights Reserved.

When >=0, simulator uses this parameter and ignores numberofpoints.

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

29 `numberofpoints=5` Number of time points of interest in the period where time domain PSD is calculated. 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.

30 `saveallsidebands=no`
Save noise contributors by sideband.
Possible values are `no` and `yes`.

31 `separatenoise=no` Separate noise into sources and transfer functions.
Possible values are `no` or `yes`.

32 `cyclo2txtfile=no` Output cyclo-stationary noise to text file as input source of next stage.
Possible values are `no` or `yes`.

33 `oscout=total` The type of output for oscillator simulation. The default value is total for the output of total modulation response from oscillator simulation. Other values are `pm` for the output of phase-modulation response and `am` for the output of amplitude-modulation response.
Possible values are `total`, `pm`, and `am`.

Convergence parameters

34 `tolerance` Relative tolerance for linear solver. The default value is 1.0e-9.

35 `relativeTol` Relative tolerance for harmonic balance-based linear solver. The default value is 1.0e-2.

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

37 `solver=turbo` Solver type.
Possible values are `std` and `turbo`.

© 2003-2014 All Rights Reserved.

38 `oscsolver=turbo` Oscillator solver type. It is recommended to use `ira` for huge circuit.
Possible values are `std`, `turbo`, `ira`, and `direct`.

39 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, and `gmres_cycle`.

40 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, and `recyclelong`.

41 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is `autoset`. With `autoset`, the simulator will automatically select the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when `autoset` is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulator's instructions may speedup subsequent analyses.
Possible values are `basicsolver, blocksolver`, and `autoset`.

42 `ppv=no` If set to `yes`, save the oscillator PPV after performing noise analysis.
Possible values are `no` and `yes`.

43 `ppvfile` File to which the PPV of oscillator is written.

44 `augmented=yes` If set to `yes`, the frequency-aware PPV method is used to calculate the total noise of the oscillator; if set to `pmonly`, only the PM part of the oscillator noise is calculated; if set to `amonly`, only the AM part of the oscillator noise is calculated.
Possible values are `no`, `yes`, `pmonly`, and `amonly`.

45 `lorentzian=cornerfreqonly`
This option determines if the Lorentzian plot is used in the

© 2003-2014 All Rights Reserved.

oscillator noise analysis.
Possible values are `no`, `cornerfreqonly`, and `yes`.

46 `pnoisemethod=default`
This option selects the shooting pnoise method.
Possible values are `default` and `fullspectrum`.

47 `krylov_size=200` The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching krylov_size iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase krylov_size if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

48 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, and `detailed_hb`.

49 `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 includes only the noise that mixes with a finite set of harmonics that are typically specified using the `maxsideband` parameter; however, in special circumstances, the harmonics may be specified with the `sidebands` parameter. If Ki represents sideband i, then:

```
f(noise_source) = f(out) + Ki * fund(pss)
```

The `maxsideband` parameter specifies the maximum |Ki| included in the PNoise calculation. Therefore, noise at frequencies less than f(out)-maxsideband*fund(pss) and greater than f(out)+maxsideband*fund(pss) are ignored. If selected sidebands are specified using the `sidebands` parameter, then only those specified are included in the calculation. When specifying the sidebands ensure that you include a sideband that contributes significant noise to the output; otherwise, the results will be erroneous.

The number of requested sidebands does not change the simulation time substantially. However, the `maxacfreq` of the corresponding PSS analysis should be set to guarantee that |max{f(noise_source)}| is less than `maxacfreq`; otherwise, the computed solution might be contaminated by aliasing effects. The PNoise simulation is not executed for |f(out)| greater than `maxacfreq`. Diagnostic messages are printed for those extreme cases, indicating which `maxacfreq` should be set in the PSS analysis. In majority of simulations, however,

© 2003-2014 All Rights Reserved.

this is not an issue, because `maxacfreq` is never allowed to be smaller than 40x the PSS fundamental.

Phase Noise measurements are possible by using the Analog Artist (ADE) environment. Two pnoise analyses are pre-configured for this simulation and most of the parameters are set by Artist. The first pnoise analysis named `mod1` is a regular noise analysis and can be used independently. The second pnoise correlation analysis named `mod2` has 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, see the Spectre RF User Guide.

You can define sweep limits by specifying the end points or by providing the center value and span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. In addition, you can give a step size parameter (`step`, `lin`, `log`, or `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 corresponding to each parameter name indicates where to find the description of that parameter.

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

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. Similar to 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 part of the PSS analysis, the periodically time-varying representation of the circuit is computed and saved for later use. The second step is to apply small-signal excitations to compute the n-port S-parameters and noise parameters. This is done using PSP analysis. PSP analysis cannot be used independently, 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.

Note: 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.

© 2003-2014 All Rights Reserved.

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=unspecified`
Specifies if the sweep frequency range is absolute frequency of input or if it is relative to the port harmonics. When the `unspecified` value is used, Spectre RF sweeps the absolute input source for non-PSP-driven cases; for other cases, Spectre RF sweeps relative to the port harmonics.
Possible values are `absolute`, `relative`, and `unspecified`.

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 printed as per the input frequency, the output frequency, or the absolute value of the input frequency. The default is `in`.
Possible values are `absin`, `in`, and `out`.

Noise parameters

15 `donoise=yes` Perform noise analysis. If `oprobe` is specified as a valid port, this parameter is set to `yes`, and a detailed noise output is

© 2003-2014 All Rights Reserved.

generated.
Possible values are `no` and `yes`.

Probe parameters

16 `maxsideband=7` In shooting pnoise, the parameter determines the maximum sideband included when computing noise that is either up-converted or down-converted to the output by the periodic drive signal. In HB pnoise, the parameter determines the size of the small signal system when HB pnoise is performed. This parameter is critical for the accuracy of HB pnoise analysis; using a small value for `maxsideband` might cause accuracy loss. The default value for the shooting pnoise is 7. For the HB pnoise, the default is the `harms/maxharms` setting in the HB large signal analysis.

Convergence parameters

17 `tolerance` Relative tolerance for shooting-based linear solver. The default value is 1.0e-9.

18 `relativeTol` Relative tolerance for harmonic balance-based linear solver. The default value is 1.0e-2.

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

20 `solver=turbo` Solver type.
Possible values are `std` and `turbo`.

21 `oscsolver=turbo` Oscillator solver type. It is recommended that you use `ira` for huge circuits.
Possible values are `std`, `turbo`, `ira`, and `direct`.

22 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, and `gmres_cycle`.

23 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, and `recyclelong`.

© 2003-2014 All Rights Reserved.

24 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is `autoset`. With `autoset`, the simulator will automatically select the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. On occasion, when `autoset` is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulator's instructions may speed up subsequent analyses.
Possible values are `basicsolver`, `blocksolver`, and `autoset`.

25 `krylov_size=200` The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching krylov_size iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase krylov_size if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

26 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, and `detailed_hb`.

27 `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.

Therefore, 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 is Ki= 0. Hence, we can associate K2=1 with the IF port and K1=0 with the RF port.

© 2003-2014 All Rights Reserved.

S21 represents 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, a choice of K1=-1 would be more appropriate.

Either `portharmsvec` or `harmsvec` 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, 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 area 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 `sweeptype` parameters behave differently in PSP than in PAC and PXF.

The `sweeptype` parameter controls the way the frequencies in the PSP analysis are swept. A `relative` sweep is a 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 down-converting mixer, `sweeptype=relative`, and a sweep range of f(rel)=0->50MHz, 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 `sweeptype=absolute` and sweeping the frequency from 900->950MHz would calculate the same quantities, because 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`). If `freqaxis` is `absin`, the S parameters at negative frequencies are taken conjugate and output at corresponding positive frequencies.

Unlike in PAC/PXF/PNoise, increasing the number of requested ports and harmonics increases 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:

© 2003-2014 All Rights Reserved.

No = total output noise at frequency f

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 (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(No^2/G^2)

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

NF = 10*log10(F)

Fdsb = (No^2 - Nl^2)/(Ns^2+Nsi^2)

NFdsb = 10*log10(Fdsb)

Fieee = (No^2 - Nl^2 - Nso^2)/Ns^2

NFieee = 10*log10(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, and not the gain from the internal port voltage source to the output.

© 2003-2014 All Rights Reserved.

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 define the sweep limits by specifying the end points or by providing the center value and span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. In addition, you can give a step size parameter (`step`, `lin`, `log`, or `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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

Periodic Steady-State Analysis (pss)

Description

This analysis computes the periodic steady-state (PSS) response of a circuit by using harmonic balance (in the frequency domain) or shooting (in the time domain). The simulation time of PSS analysis is independent of the time-constants of the circuit. In addition, PSS analysis determines the periodic operating point for the circuit. The periodic operating point can then be used during a periodic time-varying small-signal analysis, such as PAC, PXF, PNOISE, PSP, or PSTB.

Generally, harmonic balance (HB) is very efficient in simulating weak non-linear circuits while shooting is more suitable for highly non-linear circuits with sharply rising and falling signals. HB is also advantageous over shooting in handling frequency dependent components, such as delay, transmission line, and S-parameter data.

PSS analysis can handle both autonomous (non-driven) and driven (non-autonomous) circuits. Autonomous circuits, even though they are not driven by a time-varying stimulus, generate non-constant waveforms. 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, and mixers. When PSS is applied to autonomous circuits, it requires you to specify a pair of nodes, `p` and `n`. 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.

With driven circuits, specify 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 in advance. Instead, you specify an estimate of the oscillation period and PSS analysis computes the precise period along with the periodic solution waveforms.

PSS analysis consists of two phases, an initial transient phase, which initializes the circuit, and the shooting or harmonic balance phase, which computes the periodic steady-state solution. The transient phase consists of three intervals. The first interval starts at `tstart`, which is normally 0, and continues through the onset of periodicity `tonset` for the independent sources. The onset of periodicity, which is automatically generated, is the minimum time for which all sources are periodic. The second interval is an optional user-specified stabilization interval whose length is `tstab`. The final interval length is `period` for driven circuits, or four times `period` for autonomous circuits. This interval has a special use for the autonomous PSS analysis, that is, the PSS analysis monitors the waveforms in the circuit and develops a better estimate of the oscillation period. After the initial transient phase is complete, the shooting or HB phase begins. In this phase, the circuit is iteratively solved

© 2003-2014 All Rights Reserved.

using Newton method to find the periodic steady-state solution (and the period when applied to autonomous circuits).

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	`autofund=no`	If the value is `yes`, the program ignores period/fund value and calculates the fundamental frequency automatically from source information. Possible values are `no` and `yes`.
4	`tstab=0.0 s`	Extra stabilization time after the onset of periodicity for independent sources.
5	`autotstab=no`	Activates automatic initial transient (tstab) in harmonic balance. When set to yes, the simulator decides whether to run tstab and for how long. Typically, the initial length of tstab is 50 periods, but may be longer depending on the type of circuit and its behavior. If steady-state is reached (or nearly reached), tstab will terminate early. Possible values are `no` and `yes`.
6	`autosteady=no`	Activates automatic steady state detection during initial transient (tstab) in harmonic balance. When steady state is reached (or nearly reached), tstab will terminate early. Applies only when tstab>0 or when autotstab=yes. Possible values are `no` and `yes`.
7	`autoharms=no`	Activates automatic harmonic number calculation in harmonic balance. Applies only if tstab>0 or when autotstab=yes. If a steady-state is reached, Spectre does a spectrum analysis to

© 2003-2014 All Rights Reserved.

calculate the optimal number of harmonics for HB. The minimum number of harmonics is specified by `maxharms`. If steady-state is not reached to sufficient tolerance, autoharms may be disabled.
Possible values are `no` and `yes`.

8 `tstart=0.0 s` — Initial transient analysis start time.

9 `tstabenvlp=no` — Determines the envelope method to be used for tstab. If the value is set to `yes`, envelope method will be used for tstab. Default value is no.
Possible values are `no` and `yes`.

10 `envlpname` — Name of envelope analysis to be performed at tstab for pss.

Time-step parameters

11 `maxstep (s)` — Maximum time step. Default is derived from `errpreset`.

12 `maxacfreq` — Maximum frequency requested in a subsequent periodic small-signal analysis. Default is derived from `errpreset` and `harms`. This parameter is valid only for shooting.

13 `step=0.001 period s`
Minimum time step that would be used solely to maintain the aesthetics of the results. This parameter is valid only for shooting.

Initial-condition parameters

14 `ic=all` — The value be used to set the initial condition.
Possible values are `dc`, `node`, `dev`, and `all`.

15 `skipdc=no` — If set to `yes`, there is no DC analysis for initial transient.
Possible values are `no`, `yes`, and `sigrampup`.

16 `readic` — File that contains initial condition.

17 `oscic=default` — Oscillator IC method. It determines how the starting values for the oscillator are calculated. `oscic=lin` provides you an accurate initial value, but it takes time; `fastic` is very fast, but it is less accurate. `oscic=skip` directly uses the frequency you

© 2003-2014 All Rights Reserved.

provided as the initial guess frequency. It is only for two-tier method.
Possible values are `default`, `lin`, `fastic`, and `skip`.

18 `useprevic=no` — If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns.
Possible values are `no`, `yes`, and `ns`.

Convergence parameters

19 `readns` — File that contains an estimate of the initial transient solution.

20 `cmin=0 F` — Minimum capacitance from each node to ground.

Output parameters

21 `harms=9 for shooting, 10 for HB`
For shooting, it is the number of solution harmonics to output when `outputtype=freq` or `all`; for HB, it directly determines the solution dimension to be solved and impacts the accuracy and convergence of the simulation.

22 `harmsvec=[...]` — Array of desired harmonics. An alternative form of `harms` that allows selection of specific harmonics. This parameter is valid only for shooting.

23 `outputtype= all''`
Output type.
Possible values are `all`, `time`, and `freq`.

24 `save` — Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, and `nooutput`.

25 `nestlvl` — Levels of subcircuits to output.

26 `oppoint=no` — Should operating point information be computed for initial timestep; if yes, where should it be printed (screen or file).
Possible values are `no`, `screen`, `logfile`, and `rawfile`.

27 `skipstart=0 s` — The time to start skipping output data.

© 2003-2014 All Rights Reserved.

28 `skipstop=stop s`	The time to stop skipping output data.
29 `skipcount=1`	Save only one of every skipcount points.
30 `strobeperiod=0 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 `saveinit=no`	If set to `yes`, the waveforms for the initial transient before steady state are saved. Possible values are `no` and `yes`.

State-file parameters

33 `write`	File to which initial transient solution (before steady-state) is written.
34 `writefinal`	File to which final transient solution in steady-state is written. This parameter is now valid only for shooting.
35 `swapfile`	Temporary file to hold steady-state information. It 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 the analysis. This parameter is now valid only for shooting.
36 `writepss`	File to which the converged steady-state solution is written. The file of shooting and HB cannot be mutually reused.
37 `readpss`	File from which a previously converged steady-state solution is read. For shooting method, PSS loads the solution and checks the residue of the circuit equations only. The solution is re-used if the residue is satisfactory. Otherwise, the solution is re-converged using the finite difference method. The results from shooting QPSS cannot be used in HB QPSS analysis and vice-versa.
38 `checkpss=yes`	If set to `yes`, the previous PSS results (from `readpss` file) are checked and PSS+MIC is rerun if any condition has changed. If set to `no`, the simulator assumes that nothing has changed and uses the solution from the file without checking and running

© 2003-2014 All Rights Reserved.

PSS+MIC again. This parameter is now valid only for shooting.
Possible values are `no` and `yes`.

Integration method parameters

39 `method`	Integration method. The default is derived from `errpreset`. This parameter is valid only for shooting. Possible values are `euler`, `trap`, `traponly`, `gear2`, and `gear2only`.
40 `tstabmethod`	Integration method used in stabilization time. The default is `traponly` for autonomous circuits, or is derived from `errpreset` for driven circuits. Possible values are `euler`, `trap`, `traponly`, `gear2`, and `gear2only`.

Emir output parameters

41 `emirformat=none`	Format of the EM/IR database file. Possible values are `none` and `vavo`.
42 `emirstart (s)`	EM/IR start time.
43 `emirstop (s)`	EM/IR stop time.
44 `emirfile`	Name of the EM/IR database file. Default is `%A_emir_vavo.db`. The file is output to raw directory.

Accuracy parameters

45 `errpreset`	Selects a reasonable collection of parameter settings. Possible values are `liberal`, `moderate`, and `conservative`.
46 `relref`	Reference used for the relative convergence criteria. The default is derived from `errpreset`. Possible values are `pointlocal`, `alllocal`, `sigglobal`, and `allglobal`.
47 `lteratio`	Ratio used to compute LTE tolerances from Newton tolerance. The default is derived from `errpreset`.

© 2003-2014 All Rights Reserved.

48 `lteminstep=0.0 s` Local truncation error is ignored if the step size is less than `lteminstep`.

49 `steadyratio` Ratio used to compute steady state tolerances from LTE tolerance. The default is derived from `errpreset`.

50 `maxperiods` Maximum number of iterations allowed before convergence is reached in shoorting or harmonic balance Newton iteration. For PSS and QPSS, the default is 20 for driven circuits, and 50 for oscillators; For HB, the default is 100..

51 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, and `gmres_cycle`.

52 `itres=1e-4 for shooting, 0.9 for HB`
Controls the residual for iterative solution of linearized matrix equation at each Newton iteration. Tightening the parameter can help with the Newton convergence, but does not affect the result accuracy. The value should be between [0, 1].

53 `inexactNewton=no` Inexact Newton method.
Possible values are `no` and `yes`.

54 `finitediff` Enable finite difference method refinement for driven circuits after shooting method.
Possible values are `no`, `yes`, and `refine`.

55 `highorder` Perform a high-order refinement after low-order convergence. The Multi-Interval Chebyshev polynomial spectral algorithm is used. This parameter is only valid for shooting.
Possible values are `no` and `yes`.

56 `psaratio=1` Ratio used to compute high-order polynomial spectral accuracy from Newton tolerance. This parameter is only valid for shooting.

57 `maxorder` The maximum order of the Chebyshev polynomials used in waveform approximation. Possible values are from 2 to 16. Default value is 16 for driven circuits and 12 for autonomous circuits. This parameter is valid only for shooting.

58 `fullpssvec` Use the full vector containing solutions at all PSS time steps in the linear solver. The default is derived from the size of the

© 2003-2014 All Rights Reserved.

equation and the property of the PSS time steps. This parameter is valid only for shooting.
Possible values are `no` and `yes`.

59 `fdharms=10` — Number of harmonics considered for distributed (frequency-domain) components, such as nport, delay, mtline, and delayed controlled sources. This parameter is valid only for shooting and for those components for which the Fmax parameter of neither model nor instance is set.

Harmonic Balance parameters

60 `harmonicbalance=no`
Use Harmonic Balance engine instead of time-domain shooting.
Possible values are `no` and `yes`.

61 `flexbalance=no` — Same parameter as `harmonicbalance`.
Possible values are `no` and `yes`.

62 `pinnode` — Node to pin during autonomous HB simulation.

63 `pinnodeminus` — Second node to pin during autonomous HB simulation. Needed only when differential nodes exist in oscillator.

64 `pinnoderank` — Harmonic rank to pin during autonomous HB simulation.

65 `pinnodemag` — This parameter gives an estimate of the magnitude of the pin node voltage. The default value is `0.01`.

66 `oversamplefactor=1`
Oversample device evaluations.

67 `oversample=[...]` — Array of oversample factors for each tone. This parameter overrides `oversamplefactor`.

68 `oscmethod` — Osc Newton method for autonomous HB.

69 `hbhomotopy=tone` — Name of Harmonic Balance homotopy selection methods.
Possible values are `tstab`, `source`, `gsweep`, `tone`, and `inctone`.

70 `sweepic=none` — IC extrapolation method in sweep hb analysis.
Possible values are `none`, `linear`, and `log`.

© 2003-2014 All Rights Reserved.

71 `gstart=1.e-7` Start conductance for hbhomotopy of gsweep.

72 `gstop=1.e-12` Stop conductance for hbhomotopy of gsweep.

73 `glog=5` Number of steps, log sweep for hbhomotopy of gsweep.

74 `backtracking=yes` This parameter is used to activate the back tracing utility of Newtons Method. Default is `yes`.
Possible values are `no`, `yes`, and `forced`.

75 `krylov_size=10` The minimum iteration count of the linear matrix solver used in HB large-signal analysis. After reaching `krylov_size` iterations, the iteration is forced to terminate because of poor rate of convergence. Increase `krylov_size` if the simulation reports insufficient norm reduction errors in GMRES

Annotation parameters

76 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `estimated`, `steps`, `iters`, `detailed`, `rejects`, `alliters`, `detailed_hb`, and `internal_hb`.

77 `annotateic=no` Degree of annotation for initial condition.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, `iters`, `detailed`, and `rejects`.

78 `title` Analysis title.

Newton parameters

79 `maxiters=5` Maximum number of iterations per time step.

80 `restart=no` Restart the DC/PSS solution from scratch if set to `yes`; if set to `no`, reuse the previous solution as an initial guess; if set to `firstonly`, restart from scratch when it is first point of sweep (only supported in HB). The default value is `no` for HB and `yes` for shooting.
Possible values are `no`, `yes`, and `firstonly`.

© 2003-2014 All Rights Reserved.

Circuit age

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

Tstab save/restart parameters

82 `ckptperiod` Checkpoint the analysis periodically by using the specified period.

83 `saveperiod` Save the tran analysis periodically on the simulation time.

84 `saveperiodhistory=no` Maintains the history of saved files. If `yes`, stores all the saved files. Possible values are `no` and `yes`.

85 `saveclock (s)` Save the tran analysis periodically on the wall clock time. The default is 1800s for Spectre. This parameter is disabled in the APS mode by default.

86 `savetime=[...]` Save the analysis states into files on the specified time points.

87 `savefile` Save the analysis states into the specified file.

88 `recover` Specify the file to be restored.

89 `ppv=no` If set to `yes`, save the oscillators' perturbation projection vector (PPV) representing the oscillators' phase sensitivity to perturbations in the voltage or current at the nodes of the oscillator.
Possible values are `no` and `yes`.

90 `xdbcompression="no"` Sets the automatic gain compression analysis. In automatic gain compression analysis, Spectre automatically sweeps the input excitation until the gain, as defined by the analysis parameter xdbgain, compresses by the amount specified by the analysis parameter xdblevel. In gain compression analysis, Spectre ouputs the hb solution at the calculated compression point only. Dependent analyses, such as hbnoise and hbac, are supported and calculated about the calculated compression level. Auxiliary output includes the gain and voltage/power compression curves. These outputs are available for analysis and post-processing in ADE. The possible values are `yes` and `no`. Default is `no`.

© 2003-2014 All Rights Reserved.

91 `xdblevel=1.0` — Sets the gain compression level for compression analysis. The reference point for gain compression is the small-signal gain of the circuits, or as specified by the analysis parameter `xdbref`. Default is `1`.

92 `xdbgain="power"` — Chooses between the voltage gain or transducer power gain as the target for compression point calculation. When `xdbgain=power`, the gain is defined as *G (dB) = Pload (dBm) - Pavailable (dBm)*. When `xdbgain=voltage`, the gain is defined as *G (dB) = dB20(|Vload|/|Vsource|)*. In both cases, Spectre sweeps the excitation source until *xdbref - G = xdblevel*, where the analysis parameter xdbref defines the reference level for compression calculation. Possible values are `power` and `voltage`. Default is `power`.

93 `xdbref="linear"` — Sets the reference point for gain compression calculations. When `xdbref=linear`, spectre uses the small-signal gain as the reference. When `xdbref=max`, spectre uses the maximum observed gain as the reference. Possible values are `linear` and `max`. Default is `linear`.

94 `xdbsource` — The instance name of the excitation source, which is swept automatically to reach the compression level. When `xdbgain=power`, the excitation source must be a port instance. When `xdbgain=voltage`, the excitation source can be a vsource instance or a port instance. Note that in the voltage gain calculation, *dB20(|Vload|/|Vsource|)*, when `xdbsource` is a port and `xdbgain=voltage`, Spectre interprets `Vsource` as the voltage across a matched load, according to the Spectre usual port element conventions.

95 `xdbload` — The instance name of the load termination. When `xdbgain` is `power`, `xdbload` can be a port, a resistor, or a current probe.

96 `xdbnodep` — The output terminals for voltage gain calculation when `xdbgain=voltage`. If either is left unspecified, the terminal is assumed to be the global ground.

97 `xdbnoden` — The output terminals for voltage gain calculation when `xdbgain=voltage`. If either is left unspecified, the terminal is assumed to be the global ground.

© 2003-2014 All Rights Reserved.

98 `xdbrefnode` — The reference node when `xdbload` is a current probe. The default is the ground node.

99 `xdbharm=[...]` — The Integer array which specifies the harmonic indexes of the output voltage or power component.

100 `xdbsteps=100` — The maximum number of steps for the compression point search. The simulator terminates if `xdbsteps` exceeds before the compression point is found. The default is `100`.

101 `xdbmax` — The maximum input power (or voltage) for the compression point search. Default is `10 dBm` when `xdbgain=power` and `0.1V` when `xdbgain=voltage`.

102 `xdbstart` — The starting input power (or voltage) for the compression point search. Default is (xdbmax-50) dBm when `xdbgain=power`, and xdbmax/1000 when `xdbgain=voltage`.

103 `xdbtol=0.01` — Sets the tolerance for compression analysis. This tolerance is used in compression curve fitting and calculating the compression point.

104 `xdbrapid="no"` — Sets the automatic gain compression analysis in rapid mode. In this mode, Spectre does not trace the compression curve and calculates only the compression point.

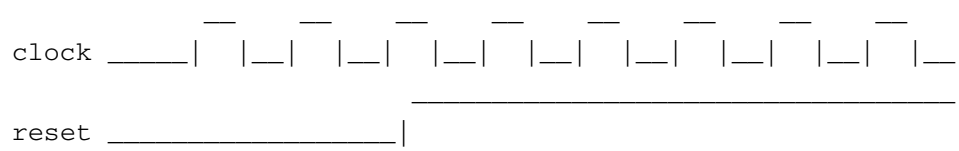
105 `xdbcpi` — Sets the estimated input-referred compression point for rapid compression analysis.

106 `memoryestimate="no"` — Sets the memory usage estimate for Harmonic Balance. If yes, a memory estimate is printed in the log file. You can use this memory estimate to plan the computing resources before submitting harmonic balance runs. In memory estimate mode, a short simulation is performed first, and the engine exits after printing the estimate in the log file without saving harmonic balance results. You must turn it off to perform an actual simulation. Memory estimation is not recommended for simulations that require less than 500MB approximately. For PSS analysis, memory estimate mode does not apply unless flexbalance=yes. The estimate applies only to large-signal analysis, and does not include subsequent noise or other small-signal simulations.

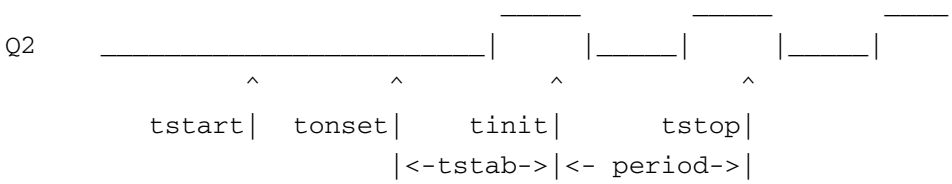
© 2003-2014 All Rights Reserved.

107 `tuneparam`	When set, `tuneparam` enables the tuning mode oscillator analysis. In the tuning mode analysis, a circuit parameter is automatically varied to reach the oscillation frequency specified by the fundfreqs parameter. The tuning parameter can be a device instance parameter (as determined by the parameters `tunedev`) or a netlist parameter. This mode applies only to autonomous circuits (oscillators).
108 `tunedev`	Sets the instance name of a device whose parameter (identified by `tuneparam)` will be varied such that the circuit oscillates at the specified frequency. Applies only in tuning mode autonomous analysis. `Tunedev` must be used with `tuneparam`.
109 `tunerange=[...]`	The tuning range of the parameter identified by `tuneparam`. Although `tunerange` is not required, it can aid in convergence, if set.
110 `lsspports=[...]`	Specifies the list of ports on which the large-signal 2-port S-parameters are calculated.
111 `lsspharms=[...]`	Specifies the output harmonic for large-signal S-parameter calculations. The input harmonic is defined by the frequency parameters on the input port instance. Default is `1`.
112 `lsspfile`	Identifies the file name for large-signal S-parameter output.
113 `lsspdatafmt`	Sets the file format of the large-signal S-parameter output. Possible values are `spectre` and `touchstone`. Default is `touchtone`.
114 `lsspdatatype`	Sets the data format or the large-signal S-parameter output. Possible values are `realimag`, `magphase` and `phase`. Default is `magphase`.

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 use of the initial transient simulation is to help in 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.



© 2003-2014 All Rights Reserved.



In the above figure, the initial transient analysis runs 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 `tstab` 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 or HB begins. Conceptually, when shooting or HB 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, starting at a good point can improve the convergence, and starting at a bad point can degrade the convergence and slow the analysis. 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, when shooting is used to simulate 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, it is likely that the opamps will undergo slew-rate limiting at the start of the shooting interval, which will slow convergence. Switching mixers follow similar rules.

When PSS analysis simulates oscillators, either transient or linear initialization is performed to obtain an initial guess of the steady-state solution and the oscillating frequency. Two initialization methods are implemented based on transient and linear analysis, respectively. When `oscic=default` is specified, transients initialization is used and the length of the transient is specified by `tstab`. It is necessary to start the oscillator by using initial conditions, or by using a brief impulsive stimulus, just as you would if you were simulating the turn-on transient of the oscillator using transient analysis. 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. The Designers Guide to Spice and Spectre [K. S. Kundert, Kluwer Academic Publishers, 1995] describes techniques for starting oscillators in some depth. When `oscic=default` is specified, `oscic=lin`, linear initialization is used. In this method, both oscillation frequency and amplitude are estimated based on linear analysis at DC solution. No impulsive stimulus or initial conditions are needed. Linear initialization is suitable for linear type of oscillators, such as LC and crystal oscillators. Note that tstab transient is still performed after linear initialization though it can be significantly shortened (or skipped in HB). Either way, specifying a non-zero `tstab` parameter can improve convergence.

© 2003-2014 All Rights Reserved.

By default, only the time-domain results are computed in shooting. If you specify either `harms` or `harmsvec`, or set `outputtype` to `freq` or `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 9. The time-domain output waveform generation can be inhibited by setting `outputtype` to `freq`.

The accuracy of the results does not depend on the number of harmonics that are requested, but only on the accuracy parameters, which are set in the same fashion as in the transient analysis. Besides a few new parameters, like `steadyratio` and `maxacfreq`, all the others parameters work in PSS analysis in exactly the same manner as they work on transient analysis. For HB, besides `reltol`, `abstol`, `steadyratio` and `lteratio`, the number of harmonics has the most impact on the accuracy of simulation results. When too few harmonics are used, an error occurs due to the aliasing effect. To obtain accurate results, `harms` should be big enough to cover the signal bandwidth.

Several parameters determine the accuracy of the PSS 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 errors in the computation of the circuit dynamics (such as time constants), relative to `reltol` and `abstol`, by setting the `lteratio` parameter.

For shooting, the `steadyratio` parameter adjusts the maximum allowed mismatch in node voltages or current branches from the beginning to the end of the steady-state period. For HB, the `steadyratio` parameter adjusts the maximum allowed error in the node voltages or in the current branches of the steady-state. This value is multiplied by `lteratio` and `reltol` to determine the convergence criterion. The relative convergence norm is printed along with the actual mismatch value at the end of each iteration, indicating the progress of the steady-state iteration.

For shooting, 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. For HB, the parameter `maxperiods` controls the maximum number of HB iterations for both driven and autonomous HB analysis. Its default value is set to `100`.

The `finitediff` parameter allows the use of finite difference (FD) after shooting. Usually this eliminates 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. You 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 tries 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 second-order method, uniform second order gear is used. `finitediff` is automatically changed from `no` to `yes` when `readpss` and `writepss` are specified to re-use PSS results.

© 2003-2014 All Rights Reserved.

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

The parameter `relref` determines how the relative error is treated. The `relref` values are as follows:

- `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 circuit signal 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 each node's 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 in shooting are as follows:

- If the circuit contains only one periodic tone and you are only interested in obtaining the periodic operating point, set errpreset to `liberal`. This setting provides reasonably accurate result and the fastest simulation speed.
- If the circuit contains more than one periodic tone and you are interested in intermodulation results, set errpreset to `moderate`. This setting provides accurate results.
- If you want a very low noise floor in your simulation result and accuracy is your main interest, set errpreset to `conservative`.

The effect of `errpreset` on other parameters for driven circuits is shown in the following table.

Table 3-3 Parameter defaults and estimated numerical noise floor in simulation result as a function of `errpreset`

errpreset	reltol	relref	method	lteratio	steadyratio	maxstep
liberal	1e-3	sigglobal	traponly	3.5	0.001	period/50

© 2003-2014 All Rights Reserved.

moderate	1e-3	alllocal	gear2only	3.5	0.001	period/200
conservative	1e-4	alllocal	gear2only	*	0.01	period/200

* : Iteratio=10.0 for conservative `errpreset`. Only if user-specified reltol <= 1e-4 * 10.0/3.5, Iteratio is set to 3.5.

The new errpreset settings include a new default `reltol` that is actually an upper limit. An increase of `reltol` above the default is ignored by the simulator. You can decrease this value in the options statement. The only way to increase `reltol` is to relax `errpreset`.

Estimated numerical noise floor for a weak non-linear circuit is -70dB for `liberal`, -90dB for `moderate`, and -120dB for `conservative` settings. For a linear circuit, the noise floor is even lower.Multi-interval Chebyshev (MIC) is activated when you explicitly set `highorder=yes`, which drops numerical noise floor by at least 30dB. MIC falls back to the original method if it encounters difficulty converging. You can tighten `psaratio` to further drop numerical noise floor.

Spectre sets the value of `maxstep` so that it cannot be larger than the value given in the table. Except for `reltol` and `maxstep`, `errpreset` does not change the value of parameters that you 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` setting is used. For HB, only `reltol` is affected by `errpreset` and the effect is the same as that in shooting. However, `lteratio` remains 3.5 and `steadyratio` remains 1 with all values of `errpreset`.

Guidelines for using errpreset in autonomous circuits are as follows:

- If you want a fast simulation with reasonable accuracy, you can set `errpreset` to `liberal`.
- If you have some concern for accuracy, you can set errpreset to `moderate`.
- If accuracy is your main interest, you can set errpreset to `conservative`.

The effect of `errpreset` on other parameters for autonomous circuits is shown in the following table.

Table 3-4 Parameter defaults as a function of errpreset

errpreset	reltol	relref	method	lteratio	steadyratio	maxstep
liberal	1e-3	sigglobal	traponly	3.5	0.001	period/50
moderate	1e-4	alllocal	gear2only	3.5	0.01	period/200
conservative	1e-5	alllocal	gear2only	*	0.1	period/400

© 2003-2014 All Rights Reserved.

* : Iteratio=10.0 for conservative `errpreset` by default. Only if user-specified reltol <= 1e-4*10.0/3.5, Iteratio is set to 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 cannot be 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. Multi-interval Chebyshev (MIC) is activated when you explicitly set `highorder=yes`, which drops numerical noise floor by at least 30dB. MIC falls back to the original method if it encounters difficulty in converging. You can tighten `psaratio` to further drop numerical noise floor.

A long stabilization (by specifying a large tstab) can help with PSS convergence. However, it can slow down simulation. By default, in the stabilization stage, the following settings are used: reltol=1e-3, maxstep=period/25, relref=sigglobal, and method=traponly. These settings are overwritten when maxstep, relref, or tstabmethod are specified explicitly in pss statement, or reltol is specified explicitly in options statement.

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 can 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 as follows:

- `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 is ignored.
- `ic=dev`: The `ic` parameters on the capacitors and inductors are used, and the `ic` statements are ignored.
- `ic=all`: Both `ic` statements and `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.

© 2003-2014 All Rights Reserved.

After 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 because Spectre can have convergence problems while trying to make the jump.

You can skip DC analysis entirely by using the parameter `skipdc`. If DC analysis is skipped, the initial solution is trivial or is given in the file that you specified by using the `readic` parameter, or if the `readic` parameter is not specified, by 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 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 as follows:

`skipdc=no`: Initial solution is calculated using 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 by 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 required solution. Second, they are a convergence

© 2003-2014 All Rights Reserved.

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, it is recommended that you use both `write` and `readns` parameters and give the same file name to both parameters. 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 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. Because differential equations have an infinite number of solutions, a complete set of initial conditions must be specified to identify the desired solution. Any initial conditions that 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 as follows:

`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, the 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: `strobing`, based on the simulation time, and `skipping` time points, which saves only every Nth point.

© 2003-2014 All Rights Reserved.

The parameters `strobeperiod` and `strobedelay` control the strobing method. `strobeperiod` sets the interval between the 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, 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.

With parameter hbhomotopy, you can specify harmonic balance homotopy selection methods. The possible values of parameter hbhomotopy and their meanings are as follows:

`hbhomotopy=tstab`: Simulator runs a transient analysis and generates an initial guess for harmonic balance analysis; it is recommended for nonlinear circuits or circuits with frequency dividers.

`hbhomotopy=source`: For driven circuit, simulator ignores tstab and accordingly increases the source power level; for oscillators, the simulator accordingly adjust the probe magnitude until probe has no effect on the oscillators. It is recommended for strongly nonlinear or high Q circuits

`hbhomotopy=tone`: This method is only valid for multi-tone circuit. The simulator first solves a single-tone circuit by turning off all the tones, except the first one, and then solves the multi-tone circuit by restoring all the tones and using the single-tone solution as its initial guess; it is recommended for multitone simulation with a strong first tone.

`hbhomotopy=inctone`: Simulator firstly solves a single tone, then turns on moderate tones incrementally till all tones are enabled. It is recommended for circuits with one strong large tone.

`hbhomotopy=gsweep`: A resistor, whose conductance is g, is connected with each node, the sweep of g is controlled by gstart, gstop, and glog; it is recommended for circuits containing high-impedance or quasi-floating nodes.

© 2003-2014 All Rights Reserved.

Parameter Index

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

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

Periodic STB Analysis (pstb)

Description

The periodic STB (PSTB) analysis is used to evaluate the local stability of a periodically varying feedback circuit. It is a small-signal analysis like STB analysis, except that 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 stability evaluation to include the effect of the time-varying operating point.

Evaluating the stability of a periodically varying circuit is a two-step process. In the first step, the small stimulus is ignored and PSS analysis is used to compute the periodic steady-state response of the circuit to a possibly large periodic stimulus. As part of the PSS analysis, the periodically time-varying representation of the circuit is computed and saved for later use. In the second step, a probe is used to compute the loop gain of the zero sideband. The local stability can be evaluated using gain margin, phase margin, or a Nyquist plot of the loop gain. To perform PSTB analysis, a probe instance must be specified as `probe` parameter.

The loop-based algorithm requires that a `probe` be 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. For the time-varying property of the circuit, the loop gain at different places can be different, but all values can be used to evaluate the stability. The loop-based algorithm provides stability information for single-loop circuits and for multiloop circuits in which a `probe` component can be placed on a critical wire to break all loops. For a typical multiloop circuit, such a critical wire may not be available. The loop-based algorithm can be used only on individual feedback loops to ensure that they are stable.

The device based algorithm requires the `probe` be a gain instant, such as a bjt transistor or a mos transistor. The device-based algorithm evaluates the loop gain around the `probe`, which can be involved in mutiloops.

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

Definition

```
Name pstb parameter=value ...
```


© 2003-2014 All Rights Reserved.

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.

Probe parameters

10	`probe`	Probe instance around which the loop gain is calculated.
11	`localgnd`	Node name of local ground. If not specified, the probe is referenced to global ground.

Output parameters

12	`save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
13	`nestlvl`	Levels of subcircuits to output.

© 2003-2014 All Rights Reserved.

Convergence parameters

14 `tolerance` Relative tolerance for shooting-based linear solver. The default value is 1.0e-9.

15 `relativeTol` Relative tolerance for harmonic balance-based linear solver. The default value is 1.0e-2.

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

17 `solver=turbo` Solver type.
Possible values are `std` or `turbo`.

18 `oscsolver=turbo` Oscillator solver type. It is recommended that you use ira for huge circuits.
Possible values are `std`, `turbo`, `ira`, or `direct`.

19 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, or `gmres_cycle`.

20 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

21 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is autoset. With autoset, the simulator will automatically select the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when autoset is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulators instructions may speedup subsequent analyses.
Possible values are `basicsolver`, `blocksolver`, and `autoset`.

© 2003-2014 All Rights Reserved.

22 `krylov_size=200` The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching krylov_size iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase krylov_size if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

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

24 `title` Analysis title.

You can specify sweep limits by providing the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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 values that the sweep parameter should take using the `values` parameter. If you specify 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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

```
krylov_size  20     probe  10           step  5
```


© 2003-2014 All Rights Reserved.

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. Unlike a conventional AC analysis that computes the response from a single stimulus to every node in the circuit, 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 required 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 specify the terminals, 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.

© 2003-2014 All Rights Reserved.

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=unspecified`
Specifies if the sweep frequency range is the absolute frequency of input or if it is relative to the port harmonics. When the unspecified value is used, Spectre RF sweeps the absolute input source for non-PSP-driven cases; for other cases, Spectre RF sweeps relative to the port harmonics.
Possible values are `absolute`, `relative`, or `unspecified`.

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

Probe parameters

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

Sampled analysis parameters

13 `ptvtype=timeaveraged`
Specifies if the ptv analysis will be traditional or sampled under certain conditions.
Possible values are `timeaveraged` or `sampled`.

14 `sampleprobe` The crossing event at this port triggers the sampled small signal computation.

15 `thresholdvalue=0` Sampled measurement is done when the signal crosses this value.

16 `crossingdirection=all`
Specifies the transitions for which sampling must be done.
Possible values are `all`, `rise`, `fall`, or `ignore`.

17 `maxsamples=16` Maximum number of sampled events to be processed during the sampled analysis.

© 2003-2014 All Rights Reserved.

18 `extrasampletimepoints=[...]`
Additional time points for sampled PTV analysis.

19 `sampleratio=1` The multiple times of fund frequency that sample frequency divides into.

Jitter parameters

20 `externalsources` Pairs of terminals or nodes corresponding to external jitter sources.

21 `extcorrsources1` Pairs of terminals and nodes for the first group of correlated external jitter sources.

22 `extcorrsources2` Pairs of terminals and nodes for the second group of correlated external jitter sources.

23 `deterministicsources`
Pairs of terminals or nodes corresponding to deterministic jitter sources.

24 `determsourcesfreqs`
Frequency list corresponding to the external deterministic jitter sources.

Output parameters

25 `stimuli=sources` Stimuli used for pxf analysis.
Possible values are `sources` or `nodes_and_terminals`.

26 `sidebands=[...]` Array of relevant sidebands for the analysis.

27 `maxsideband=7` An alternative to the `sidebands` array specification, which automatically generates the array: [-maxsideband ... 0 ... +maxsideband]. For shooting analysis, the default value is 7. For HB small signal analysis, the default value is the `harms/maxharms` setting in the HB large signal analysis. It is ignored in HB small signal when it is larger than the harms/maxharms of large signal.

28 `freqaxis` Specifies whether the results should be output versus the input frequency, the output frequency, or the absolute value of the

© 2003-2014 All Rights Reserved.

input frequency. Default is `absin`.
Possible values are `absin`, `in`, or `out`.

29 `save` — Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.

30 `nestlvl` — Levels of subcircuits to output.

31 `oscout=total` — The type of output for oscillator simulation. The default value is total for the output of total modulation response from oscillator simulation. Other values are- pm for the output of phase-modulation response and am for the output of amplitude-modulation response.
Possible values are `total`, `pm`, or `am`.

Convergence parameters

32 `tolerance` — Relative tolerance for shooting-based linear solver. The default value is 1.0e-9.

33 `relativeTol` — Relative tolerance for harmonic balance-based linear solver. The default value is 1.0e-2.

34 `gear_order=2` — Gear order used for small-signal integration.

35 `solver=turbo` — Solver type.
Possible values are `std` or `turbo`.

36 `oscsolver=turbo` — Oscillator solver type. It is recommended that you use ira for huge circuits.
Possible values are `std`, `turbo`, `ira`, or `direct`.

37 `lnsolver=gmres` — Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, or `gmres_cycle`.

38 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

39 `hbprecond_solver=autoset`

© 2003-2014 All Rights Reserved.

Select a linear solver for the GMRES preconditioner. Default is autoset. With autoset, the simulator will automatically select the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when autoset is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulators instructions may speedup subsequent analyses.
Possible values are `basicsolver`, `blocksolver`, and `autoset`.

40 krylov_size=200 — The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching krylov_size iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase krylov_size if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

41 `annotate=sweep` — Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, and `detailed_hb`.

42 `title` — Analysis title.

Modulation conversion parameters

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

44 `outmodharmnum=1` — Harmonic for the PXF output modulation.

45 `inmodharmvec=[...]` — Harmonic list for the PXF modulated sources.

46 `moduppersideband=1` — Index of the upper sideband included in the modulation of an output for PAC or an input for PXF.

© 2003-2014 All Rights Reserved.

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` parameter. For a given set of n integer numbers representing the sidebands `K1, K2, ... Kn`, the input signal frequency at each sideband is computed as `f(in)= f(out) + Ki * fund(pss)`, where, `f(out)` represents the (possibly swept) output signal frequency and 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, `Ki= +1` for the RF input represents the first upper-sideband, while Ki= -1 for the RF input represents the first lower-sideband. By setting the `maxsideband` value to Kmax, all 2 * Kmax + 1 sidebands from -Kmax to +Kmax 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 | max{f(in)} | is less than `maxacfreq`; otherwise, the computed solution might be contaminated by aliasing effects. The PXF simulation is not executed for | f(out) | greater than `maxacfreq`. Diagnostic messages are printed for those extreme cases, indicating how `maxacfreq` should be set in the PSS analysis. In majority of simulations, however, this is not an issue, because `maxacfreq` is never allowed to be smaller than 40x the PSS fundamental.

With PXF, the frequency of the stimulus and of the response are usually different (this is an important area in which PXF differs from XF). 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 input frequency (`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 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.

© 2003-2014 All Rights Reserved.

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 (for example, 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 by using the Analog Artist (ADE) environment. The `modulated` option for PXF and other modulated parameters are set by Artist. PXF analyses with this option produce results that could have limited use outside such an 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, see Spectre RF User Guide.

You can specify sweep limits by providing the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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 values that the sweep parameter should take by using the `values` parameter. If you specify 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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

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, you need to specify input sources and output voltages or currents. If no input or output is given, only poles are computed. If 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 uses default direct solver (method=qz) for better accuracy. Small to medium circuit size achieves better performance. For larger circuits, a Krylov subspace iterative solver (method=arnoldi) can be used for better performance, but with lesser accuracy.

Note: A frequency-dependent component means that the capacitance or conductance-equivalent representation of the component varies with the frequency. 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 specifying the parameter name without a `dev` or `mod` parameter. After the analysis is complete, 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 for cancellation. A subtle note on resistance: by default, a lower bound of resistance is enforced; you may remove this limitation by defining the resistor parameter `rac`. This may affect pz results.

Definition

```
Name ... pz parameter=value ...
```

Parameters

Probe parameters

1 `iprobe` Input probe for zeros of the transfer function.

© 2003-2014 All Rights Reserved.

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.

© 2003-2014 All Rights Reserved.

State-file parameters

18 `readns`	File that contains estimate of DC solution (nodeset).
19 `useprevic=no`	If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns. Possible values are `no`, `yes` or `ns`.

Output parameters

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

Filtering parameters

22 `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.
23 `docancel=yes`	If set, pole-zero cancellation is requested. Possible values are `no` or `yes`.
24 `absdiff=1e-6 Hz`	Pole-Zero cancel absolute distance in Hz.
25 `reldiff=1e-4`	Pole-Zero cancel relative distance.

Convergence parameters

26 `prevoppoint=no`	Use the operating point computed on the previous analysis. Possible values are `no` or `yes`.
27 `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`.

© 2003-2014 All Rights Reserved.

Annotation parameters

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

Miscellaneous parameters

30	`method=qz`	Method to perform pz analysis. Possible values are `qz` or `arnoldi`.
31	`numpoles`	Maximum number of poles requested, only for arnoldi method.
32	`numzeros`	Maximum number of zeros requested, only for arnoldi method.
33	`sigmar=0.1`	root finding control parameter, only for arnoldi method.
34	`sigmai=0.0`	root finding control parameter, only for arnoldi method.

Examples

```
mypz pz
```

Pole analysis is performed.

```
mypz2  (n1 n2) pz iprobe=VIN
```

Input is VIN and output is the voltage difference between nodes n1 and n2. Both pole and zero analyses are performed.

```
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.

```
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.

```
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.

```
mypz6 pz iprobe=VIN oprobe=R3 portv=1
```


© 2003-2014 All Rights Reserved.

Input is VIN, output is the voltage across the resistor R3. Both pole and zero analyses will be performed.

```
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.

```
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.

```
mypz9 (n1 n2) pz iprobe=VIN docancel=no
```

Do not perform pole-zero cancellation.

Note: `porti` allows you 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 exclusively with voltage sources and current probes and with other components that have voltage-defined branches.

When PZ analysis finishes, a table is printed by default. Included in the table are the values of poles/zeros and a brief notification on the "right-hand side poles" if there are any. This information can also be viewed graphically. In addition to these direct results, "DC gain" is also listed at the end. It calculates the gain of transfer function H(s) given s=0, including the contribution of the excluded poles/zeros (blocked by user specified fmax). "Constant factor" calculates the ratio of the coefficients of the leading terms in the numerator and denominator of H(s).

Parameter Index

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

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

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 that the circuit is first linearized about a quasi-periodically varying operating point, as opposed to a simple DC operating point. Linearizing about a quasi-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. In addition, the frequency of the sinusoidal stimulus is not constrained by the period of the large periodic solution.

Computing the small-signal response of a quasi-periodically varying circuit is a two-step process. First, the small stimulus is ignored and the quasi-periodic steady-state response of the circuit to possibly large periodic stimuli is computed using QPSS analysis. As part of the QPSS analysis, the quasi-periodically 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 independently; it must follow a QPSS analysis. However, any number of quasi-periodic 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`	Start sweep limit.
2	`stop`	Stop sweep limit.
3	`center`	Center of sweep.

© 2003-2014 All Rights Reserved.

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, that is, the actual frequency, or if it is relative to the "relharmvec" sideband frequency. Possible values are `absolute` or `relative`.
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=7`	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. The default is `absout`. Possible values are `absout`, `out`, or `in`.
15	`save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
16	`nestlvl`	Levels of subcircuits to output.

© 2003-2014 All Rights Reserved.

Convergence parameters

17 `tolerance` — Relative tolerance for linear solver; the default value is 1.0e-9 for shooting-based solver and 1.0e-6 for harmonicbalance-based solver.

18 `relativeTol` — Relative tolerance for harmonicbalance-based linear solver; the default value is 1.0e-2.

19 `gear_order=2` — Gear order used for small-signal integration, 1 or 2.

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

21 `lnsolver=gmres` — Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, or `gmres_cycle`.

22 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

23 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is autoset. With autoset, the simulator will automatically select the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when autoset is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulators instructions may speedup subsequent analyses.
Possible values are `basicsolver`, `blocksolver` and `autoset`.

24 `krylov_size=200` — The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching krylov_size iterations, the iteration is forced to terminate because of the poor

© 2003-2014 All Rights Reserved.

rate of convergence. Increase krylov_size if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

25 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, `steps`, and `detailed_hb`.
26 `title`	Analysis title.

You can select the set of periodic small-signal output frequencies of interest by setting either the `clockmaxharm` or the `sidevec` parameter. Sidebands are vectors in QPAC. Assuming that there is one large tone and one moderate tone in QPSS, a sideband K1 is represented as [K1_1 K1_2]. Corresponding frequency is as follows:

`K1_1 * fund(large tone of QPSS) + K1_2 * fund` (moderate tone of QPSS)

If 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 follows:

```
f(out)= f(in) + SUM_j=1_to_L{Ki_j * fund_j(qpss)},
```

where, `f(in)` represents the (possibly swept) input frequency, and 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}. You 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- the first fundamental is affected by this entry; the rest- moderate tones- are limited by `maxharms`, specified for a QPSS analysis. Given maxharms=[k1max k2max ... knmax] in QPSS and `clockmaxharm=Kmax all (2*Kmax + 1)*(2*k2max+1)*(2*k3max+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 area 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`).

© 2003-2014 All Rights Reserved.

You can specify sweep limits by specifying the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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 values that the sweep parameter should take using the `values` parameter. If you specify 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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

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, it is 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 quasi-periodically time-varying nature of the linearized circuit that accounts for the frequency conversion and intermodulation. 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 spectral density versus frequency. The output of the circuit is specified with 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 required, specify the input source by 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 computes 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, 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(input)| = |f(out) + refsideband frequency shift|.
```

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

Sidebands are vectors in QPNOISE. Assuming one large tone and one moderate tone in QPSS, a sideband Ki is a vector [Ki_1 Ki_2]. It gives the frequency at :

© 2003-2014 All Rights Reserved.

`Ki_1 * fund(large tone of QPSS) + Ki_2 * 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 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:

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(No^2 / G^2)

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

NF = 10*log10(F)

© 2003-2014 All Rights Reserved.

Fdsb = (No^2 - Nl^2)/(Ns^2+Nsi^2)

NFdsb = 10*log10(Fdsb)

Fieee = (No^2 - Nl^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.

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.

Definition

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

The optional terminals (p and n) specify the output of the circuit. If you do not specify the terminals, 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.

© 2003-2014 All Rights Reserved.

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, that is, the actual frequency, or if it is relative to the "relharmvec" sideband frequency.
Possible values are `absolute` or `relative`.

11 `relharmvec=[...]` Sideband- the 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` In shooting pnoise, the parameter determines the maximum sideband included when computing noise that is either up-converted or down-converted to the output by the periodic drive signal. The default value for the shooting pnoise is 7. In HB pnoise, this parameter determines the size of the small signal system when the HB pnoise is performed. This parameter is critical for the accuracy of the HB pnoise analysis; using small maxsideband may cause accuracy loss. The default value for HB pnoise is the `harms/maxharms` setting in the HB large signal analysis.

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

© 2003-2014 All Rights Reserved.

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

19 `nestlvl` Levels of subcircuits to output.

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

21 `separatenoise=no` Separate Noise into sources and transfer functions.
Possible values are `no` or `yes`.

Convergence parameters

22 `tolerance` Relative tolerance for linear solver; the default value is 1.0e-9 for shooting-based solver, and 1.0e-6 for harmonicbalance-based solver.

23 `relativeTol` Relative tolerance for harmonicbalance-based linear solver; the default value is 1.0e-2.

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

25 `solver=turbo` Solver type.
Possible values are `std` or `turbo`.

26 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, or `gmres_cycle`.

27 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

28 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is autoset. With autoset, the simulator will automatically select the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when autoset is selected, the simulator

© 2003-2014 All Rights Reserved.

may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulators instructions may speedup subsequent analyses.
Possible values are `basicsolver` and `autoset`.

29 `krylov_size=200` — The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching krylov_size iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase krylov_size if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

30 `annotate=sweep` — Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, and `detailed_hb`.

31 `title` — 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 includes only the noise that mixes with a finite set of harmonics that are specified using the `clockmaxharm` and `sidevec` parameters. Sidebands are vectors in quasi-periodic analyses. For one large tone and one moderate tone in QPSS, a sideband K1 is represented as [K1_1 K1_2]. Corresponding frequency shift is as follows:

`K1_1 * fund(large tone of QPS S) + K1_2 * fund` (moderate tone of QPSS)

Assuming 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.
```

If Ki represents sideband i, then:

```
f(noise_source) = f(out) +  SUM_j=1_to_L{ Ki_j * fund_j(qpss) },
```

The `clockmaxharm` parameter affects only clock frequency. It can be less or more than `maxharms[1]` in QPSS. Moderate tones are limited by `maxharms` specified in QPSS. Only the selected sidebands specified using the `sidevec` parameter are included in the calculation. Care should be taken when specifying `sidevec` or `clockmaxharm` QPNOISE

© 2003-2014 All Rights Reserved.

and `maxharms` in QPSS. Noise results are erroneous if you do not include the sidebands that contribute significant noise to the output.

The number of requested sidebands changes substantially the simulation time.

You can specify sweep limits by specifying the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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 values that the sweep parameter should take using the `values` parameter. If you specify 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 corresponding to each parameter name indicates where to find the description of that parameter.

© 2003-2014 All Rights Reserved.

Quasi-Periodic S-Parameter Analysis (qpsp)

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 done 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 quasi-periodically 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 quasi-periodically varying circuit is a two-step process. First, the small stimulus is ignored and the quasi-periodic steady-state response of the circuit to possibly large periodic stimulus is computed using QPSS analysis. As 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 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 independently; 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.

© 2003-2014 All Rights Reserved.

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, that is, the 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` or `relative`.

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. The default is `in`. Possible values are `absin`, `in`, or `out`.

© 2003-2014 All Rights Reserved.

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` — In shooting pnoise, the parameter determines the maximum sideband included when computing noise that is either up-converted or down-converted to the output by the periodic drive signal. The default value for the shooting pnoise is 7. In HB pnoise, this parameter determines the size of the small signal system when the HB pnoise is performed. This parameter is critical for the accuracy of the HB pnoise analysis; using small maxsideband may cause accuracy loss. The default value for the HB pnoise is the `harms/maxharms` setting in the HB large signal analysis.

Convergence parameters

17 `tolerance` — Relative tolerance for linear solver; the default value is 1.0e-9 for shooting-based solver and 1.0e-6 for harmonicbalance-based solver.

18 `relativeTol` — Relative tolerance for harmonicbalance-based linear solver; the default value is 1.0e-2.

19 `gear_order=2` — Gear order used for small-signal integration, 1 or 2.

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

21 `lnsolver=gmres` — Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, or `gmres_cycle`.

22 `resgmrescycle=short` — Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

© 2003-2014 All Rights Reserved.

23 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is autoset. With autoset, the simulator will automatically select the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. At times, when autoset is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulators instructions may speedup subsequent analyses.
Possible values are `basicsolver`,`blocksolver`, and `autoset`.

24 `krylov_size=200` The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching krylov_size iterations, the iteration is forced to terminate because of the poor rate of convergence. Increase krylov_size if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

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

26 `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. Assuming that there is one large tone and one moderate tone in QPSS, a sideband K1 is represented as `[K1_1 K1_2]`. Corresponding frequency is as follows:

`K1_1 * fund(large tone of QPSS) + K1_2 * fund` (moderate tone of QPSS)
`= SUM_j=1_to_L{Ki_j * fund_j(qpss)}`

It is also assumed 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,...K1_j..., K1_L } , K2, ... Kn.`

© 2003-2014 All Rights Reserved.

If we specify the relative frequency, the scattering parameters at each port are computed at the following frequencies:

```
f(scattered)= f(rel) + SUM_j=1_to_L{Ki_j * fund_j(qpss)},
```

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 f`und_j(qpss)` represents the fundamental frequency used in the corresponding QPSS analysis.

In the analysis of a down-converting mixer with a blocker and of the signal in the upper sideband, 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, it is Ki= {0 , 0}. Hence, you can associate K1={1 , 0} with the RF port and K2={0 , 0} with the IF port. S21 represents the transmission of a signal from RF to IF, and S11 represents the reflection of the signal back to the RF port. If the signal is in the lower sideband, a choice of K1={-1 , 0} would be more appropriate.

Either `portharmsvec` or `harmsvec` can be used to specify the sidebands of 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, 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. A `relative` sweep is a 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. If `sweeptype` is set to `relative` with a sweep range of f(rel)=-10MHz<->10MHz, 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, because `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`).

© 2003-2014 All Rights Reserved.

An increase in the number of requested ports increases the simulation time substantially. The same happens if you increase the number of sidebands to be included in 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:

No = total output noise at frequency f

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 (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(No^2 / G^2)

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

NF = 10*log10(F)

Fdsb = (No^2 - Nl^2)/(Ns^2+Nsi^2)

© 2003-2014 All Rights Reserved.

NFdsb = 10*log10(Fdsb)

Fieee = (No^2 - Nl^2 - Nso^2)/Ns^2

NFieee = 10*log10(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, and 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 relevant only to the noise computation features of QPSP.

You can specify sweep limits by specifying the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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 values that the sweep parameter should take by using the `values` parameter. If you specify 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 corresponding to each parameter name indicates where to find the description of that parameter.

© 2003-2014 All Rights Reserved.

```
freqaxis  14        log  8              step  5

gear_order  18      portharmsvec  12    stop  2
```


© 2003-2014 All Rights Reserved.

Quasi-Periodic Steady State Analysis (qpss)

Description

Quasi-periodic steady-state (QPSS) analysis computes circuit response with multiple fundamental frequencies using harmonic balance (in frequency domain) or shooting. QPSS can compute circuit's responses with closely spaced or incommensurate fundamentals, which cannot be resolved by PSS efficiently. The simulation time of QPSS analysis is independent of the time-constants of the circuit. In addition, QPSS analysis sets the circuit quasi-periodic operating point, which can then be used during a quasi-periodic time-varying small-signal analysis, such as QPAC, QPXF, QPSP, and QPNOISE.

Generally, harmonic balance (HB) is very efficient in simulating weakly nonlinear circuits while shooting is more suitable to compute a circuit response to several moderate input signals, in addition to a large signal. The large signal, which represents a LO or clock signal, is usually the one that causes the most nonlinearity or the largest response. A typical example is the intermodulation distortion measurements of a mixer with two closely spaced moderate input signals. HB is more efficient than shooting in handling frequency-dependent components, such as delay, transmission line, and S-parameter data.

QPSS consists of three phases. First, an initial transient analysis with all moderate input signals suppressed is carried out. Second, a number of (at least 2) stabilizing iterations are run with all signals activated. This is followed by the Newton method.

When the shooting method is used, QPSS employs the Mixed Frequency Time (MFT) algorithm extended to multiple fundamental frequencies. For details of MFT algorithm, see `Steady-State Methods for Simulating Analog and Microwave Circuits,` by K. S. Kundert, J.K. White, and A. Sangiovanni-Vincentelli, Kluwer, Boston, 1990.

Similar to shooting in PSS, shooting in QPSS uses 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 `maxharms`. Given `maxharms=[k1 k2 ... kn]`, QPSS always treats k1 as the maximum harmonic of the large signal, and the total number of integrations is `(2*k2+1)*(2*k3+1)*...*(2*kn+1)`. The first consequence is that the efficiency of the algorithm depends significantly on the number of harmonics required to model the responses of moderate fundamentals. Another consequence is that 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.

© 2003-2014 All Rights Reserved.

QPSS inherits most of the PSS parameters and adds a few new ones. The most important ones are `funds` and `maxharms`. They replace the PSS parameter, `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 the `fundname` parameter. In both shooting and HB QPSS analysis, the first fundamental is considered as the large signal. A few heuristics can be used for picking the large fundamental.

(1) Pick the fundamental that is not a sinusoidal.

(2) Pick the fundamental that causes the most nonlinearity.

(3) Pick the fundamental that 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.

The semi-autonomous simulation is a special QPSS analysis combining the autonomous simulation and the QPSS. For the semi-autonomous simulation, you need to specify an initial frequency guess for the oscillator inside the circuit, and two oscillator terminals, just like the autonomous simulation in the PSS. For example:

```
myqpss (op on) qpss funds=[1.1GHz frf] maxharms=[5 5] tstab=1u
flexbalance=yes
```

The semi-autonomous simulation is only available in the frequency domain.

Definition

```
Name ... qpss parameter=value ...
```

Parameters

QPSS fundamental parameters

1	`funds=[...]`	Array of fundamental frequency names for fundamentals to use in analysis.
2	`maxharms=[...]`	Array of number of harmonics of each fundamental to consider for each fundamental.
3	`selectharm`	Name of harmonics selection methods. Default is diamond when `maximorder` or `boundary` is set; otherwise, default is box. Possible values are `box`, `diamond`, `funnel`, or `axis`.

© 2003-2014 All Rights Reserved.

4	`evenodd=[...]`	Array of even, odd, or all strings for moderate tones to select harmonics.
5	`boundary`	Harmonic selection boundary.
6	`maximorder`	Maximum intermodulation order (same parameter as `boundary`).
7	`harmlist=[...]`	Array of harmonics indices.
8	`freqdivide`	Large signal frequency division.

Simulation interval parameters

9	`tstab=0.0 s`	Extra stabilization time after the onset of periodicity for independent sources.
10	`autotstab=no`	Activates automatic initial transient (tstab) in harmonic balance. When set to yes, the simulator decides whether to run tstab and for how long. Typically, the initial length of tstab is 50 periods, but may be longer depending on the type of circuit and its behavior. If steady-state is reached (or nearly reached), tstab will terminate early. Possible values are `no` and `yes`.
11	`autosteady=no`	Activates automatic steady state detection during initial transient (tstab) in harmonic balance. When steady state is reached (or nearly reached), tstab will terminate early. Applies only when tstab>0 or when autotstab=yes. Possible values are `no` and `yes`.
12	`autoharms=no`	Activates automatic harmonic number calculation in harmonic balance. Applies only if tstab>0 or when autotstab=yes. If a steady-state is reached, Spectre does a spectrum analysis to calculate the optimal number of harmonics for HB. The minimum number of harmonics is specified by `maxharms`. If steady-state is not reached to sufficient tolerance, autoharms may be disabled. Possible values are `no` and `yes`.
13	`stabcycles=2`	Stabilization cycles with both large and moderate sources enabled.

© 2003-2014 All Rights Reserved.

14 `tstart=0.0 s` Initial transient analysis start time.

Time-step parameters

15 `maxstep (s)` Maximum time step. The default is derived from `errpreset`.

16 `maxacfreq` Maximum frequency requested in a subsequent periodic small-signal analysis. The default is derived from `errpreset` and `harms`. This parameter is valid only for shooting.

17 `step=0.001 period s`
Minimum time step that would be used solely to maintain the aesthetics of the results. This parameter is valid only for shooting.

Initial-condition parameters

18 `ic=all` The value to be used to set the initial condition.
Possible values are `dc`, `node`, `dev`, or `all`.

19 `skipdc=no` If yes, there is no dc analysis for initial transient.
Possible values are `no`, `yes`, or `sigrampup`.

20 `readic` File that contains initial condition.

21 `useprevic=no` If set to yes or ns, Use the converged initial condition from previous analysis as ic or ns.
Possible values are `no`, `yes` or `ns`.

Convergence parameters

22 `readns` File that contains estimate of initial transient solution.

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

Output parameters

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

© 2003-2014 All Rights Reserved.

25 `nestlvl`	Levels of subcircuits to output.
26 `oppoint=no`	Should operating point information be computed for initial timestep; if yes, where should it be sent. Possible values are `no`, `screen`, `logfile`, or `rawfile`.
27 `skipstart=0 s`	The time to start skipping output data.
28 `skipstop=stop s`	The time to stop skipping output data.
29 `skipcount=1`	Save only one of every skipcount points.
30 `strobeperiod=0 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 `saveinit=no`	If set, the waveforms for the initial transient before steady state are saved. Possible values are `no` or `yes`.

State-file parameters

33 `write`	File to which initial transient solution (before steady-state) is to be written.
34 `writefinal`	File to which final transient solution in steady-state is to be written. This parameter is now valid only for shooting.
35 `swapfile`	Temporary file to hold steady-state information. It tells the simulator to use a regular file rather than virtual memory, to hold the periodic operating point. Use this option if the simulator complains about not having enough memory to complete the analysis. This parameter is now valid only for shooting.

Integration method parameters

36 `method`	Integration method. Default derived from `errpreset`. This parameter is valid only for shooting. Possible values are `euler`, `trap`, `traponly`, `gear2`, or `gear2only`.

© 2003-2014 All Rights Reserved.

Emir output parameters

37 `emirformat=none` — Format of the EM/IR database file.
Possible values are `none` or `vavo`.

38 `emirstart (s)` — EM/IR start time.

39 `emirstop (s)` — EM/IR stop time.

40 `emirfile` — Name of the EM/IR database file. The default is %A_emir_vavo.db. The file will be output to raw directory.

Accuracy parameters

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

42 `relref` — Reference used for the relative convergence criteria. The default is derived from `errpreset`.
Possible values are `pointlocal`, `alllocal`, `sigglobal`, or `allglobal`.

43 `lteratio` — Ratio used to compute LTE tolerances from Newton tolerance. Default derived from `errpreset`.

44 `lteminstep=0.0 s` — Local truncation error is ignored if the step size is less than lteminstep.

45 `steadyratio` — Ratio used to compute steady-state tolerances from LTE tolerance. The default is derived from `errpreset`.

46 `maxperiods` — Maximum number of iterations allowed before convergence is reached in shoorting or harmonic balance Newton iteration. For PSS and QPSS, the default is 20 for driven circuits, and 50 for oscillators; For HB, the default is 100.

47 `lnsolver=gmres` — Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, or `gmres_cycle`.

48 `itres=1e-4 for shooting, 0.9 for HB` — The itres parameter controls the residual for iterative solution of linearized matrix equation at each Newton iteration. Tightening

© 2003-2014 All Rights Reserved.

the parameter can help with the Newton convergence, but does not affect the result accuracy. The value should be between [0, 1].

49 `inexactNewton=no` Inexact Newton method.
Possible values are `no` or `yes`.

50 `finitediff` Options for finite difference method refinement after quasi-periodic shooting method. `finitediff` is changed from no to same grid automatically when `readqpss` and `writeqpss` are used to reuse QPSS results.
Possible values are `no`, `yes`, or `refine`.

Harmonic Balance parameters

51 `harmonicbalance=no`
Use Harmonic Balance engine instead of time-domain shooting.
Possible values are `no` or `yes`.

52 `flexbalance=no` Same parameter as `harmonicbalance`.
Possible values are `no` or `yes`.

53 `hbpartition_defs=[...]`
Define HB partitions.

54 `hbpartition_fundratios=[...]`
Specify HB partition fundamental frequency ratios.

55 `hbpartition_harms=[...]`
Specify HB partition harmonics.

56 `oversamplefactor=1`
Oversample device evaluations.

57 `oversample=[...]` Array of oversample factors for each tone. It overrides oversamplefactor.

58 `hbhomotopy=tone` Name of Harmonic Balance homotopy selection methods.
Possible values are `tstab`, `source`, `gsweep`, `tone`, or `inctone`.

59 `sweepic=none` IC extrapolation method in sweep hb analysis.
Possible values are `none`, `linear`, or `log`.

© 2003-2014 All Rights Reserved.

60 `gstart=1.e-7` Start conductance for hbhomotopy of gsweep.

61 `gstop=1.e-12` Stop conductance for hbhomotopy of gsweep.

62 `glog=5` Number of steps, log sweep for hbhomotopy of gsweep.

63 `backtracking=yes` This parameter is used to activate the backtracing utility of Newton Method. Default is yes.
Possible values are `no`, `yes`, or `forced`.

64 `krylov_size=10` The minimum iteration count of the linear matrix solver used in HB large-signal analysis. After reaching the `krylov_size` iterations, the iteration is forced to terminate because of poor rate of convergence. Increase `krylov_size` if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

65 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `estimated`, `steps`, `iters`, `detailed`, `rejects`, `alliters`, `detailed_hb`, and `internal_hb`.

66 `annotateic=no` Degree of annotation for initial condition.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, `iters`, `detailed`, or `rejects`.

67 `title` Analysis title.

Newton parameters

68 `maxiters=5` Maximum number of iterations per time step.

69 `restart=no` Restart the DC/PSS/QPSS solution from scratch if set to yes; if set to no, reuse the previous solution as initial guess; if set to firstonly, restart from scratch when it is first point of sweep; it is supported only in HB. The default value is no for HB and yes for shooting.
Possible values are `no`, `yes`, or `firstonly`.

© 2003-2014 All Rights Reserved.

Circuit age

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

71 `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. The file of shooting and HB cant mutually reuse.

72 `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. The file of shooting and HB cant mutually reuse.

Tstab save/restart parameters

73 `ckptperiod` Checkpoint the analysis periodically using the specified period.

74 `saveperiod` Save the tran analysis periodically on the simulation time.

75 `saveperiodhistory=no` Maintains the history of saved files. If `yes`, stores all the saved files. Possible values are `yes` and `no`.

76 `saveclock (s)` Save the tran analysis periodically on the wall clock time. The default is 1800s for Spectre. The feature is disabled in APS mode by default.

77 `savetime=[...]` Save the analysis states into files on the specified time points.

78 `savefile` Save the analysis states into the specified file.

79 `recover` Specify the file to be restored.

80 `oscic=default` Oscillator IC method. It determines how the starting values for the oscillator are calculated. `oscic=lin` gives you an accurate initial value, but it takes some time; `fastic` is very fast, but it is less accurate. oscic=skip directly uses the user provided frequency as the initial guess frequency. It is only for two-tier method.
Possible values are `default`, `lin`, `fastic`, or `skip`.

© 2003-2014 All Rights Reserved.

81 `xdbcompression="no"` Sets the automatic gain compression analysis. In automatic gain compression analysis, Spectre automatically sweeps the input excitation until the gain, as defined by the analysis parameter xdbgain, compresses by the amount specified by the analysis parameter xdblevel. In gain compression analysis, Spectre ouputs the hb solution at the calculated compression point only. Dependent analyses, such as hbnoise and hbac, are supported and calculated about the calculated compression level. Auxiliary output includes the gain and voltage/power compression curves. These outputs are available for analysis and post-processing in ADE. The possible values are `yes` and `no`. Default is `no`.

82 `xdblevel=1.0` Sets the gain compression level for compression analysis. The reference point for gain compression is the small-signal gain of the circuits, or as specified by the analysis parameter `xdbref`. Default is `1`.

83 `xdbgain="power"` Chooses between the voltage gain or transducer power gain as the target for compression point calculation. When `xdbgain=power`, the gain is defined as *G (dB) = Pload (dBm) - Pavailable (dBm)*. When `xdbgain=voltage`, the gain is defined as *G (dB) = dB20(|Vload|/|Vsource|)*. In both cases, Spectre sweeps the excitation source until *xdbref - G = xdblevel*, where the analysis parameter xdbref defines the reference level for compression calculation. Possible values are `power` and `voltage`. Default is `power`.

84 `xdbref="linear"` Sets the reference point for gain compression calculations. When `xdbref=linear`, spectre uses the small-signal gain as the reference. When `xdbref=max`, spectre uses the maximum observed gain as the reference. Possible values are `linear` and `max`. Default is `linear`.

85 `xdbsource` The instance name of the excitation source, which is swept automatically to reach the compression level. When `xdbgain=power`, the excitation source must be a port instance. When `xdbgain=voltage`, the excitation source can be a vsource instance or a port instance. Note that in the voltage gain calculation, *dB20(|Vload|/|Vsource|)*, when `xdbsource` is a port and `xdbgain=voltage`, Spectre

© 2003-2014 All Rights Reserved.

interprets `Vsource` as the voltage across a matched load, according to the Spectre usual port element conventions.

86 `xdbload` — The instance name of the load termination. When `xdbgain` is `power`, `xdbload` can be a port, a resistor, or a current probe.

87 `xdbnodep` — The output terminals for voltage gain calculation when `xdbgain=voltage`. If either is left unspecified, the terminal is assumed to be the global ground.

88 `xdbnoden` — The output terminals for voltage gain calculation when `xdbgain=voltage`. If either is left unspecified, the terminal is assumed to be the global ground.

89 `xdbrefnode` — The reference node when `xdbload` is a current probe. The default is the ground node.

90 `xdbharm=[...]` — The Integer array which specifies the harmonic indexes of the output voltage or power component.

91 `xdbsteps=100` — The maximum number of steps for the compression point search. The simulator terminates if `xdbsteps` exceeds before the compression point is found. The default is `100`.

92 `xdbmax` — The maximum input power (or voltage) for the compression point search. Default is `10 dBm` when `xdbgain=power` and `0.1 V` when `xdbgain=voltage`.

93 `xdbstart` — The starting input power (or voltage) for the compression point search. Default is (xdbmax-50) dBm when xdbgain=power, and xdbmax/1000 when xdbgain=voltage.

94 `xdbtol=0.01` — Sets the tolerance for compression analysis. This tolerance is used in compression curve fitting and calculating the compression point.

95 `xdbrapid="no"` — Sets the automatic gain compression analysis in rapid mode. In this mode, Spectre does not trace the compression curve and calculates only the compression point.

96 `xdbcpi` — Sets the estimated input-referred compression point for rapid compression analysis.

© 2003-2014 All Rights Reserved.

97 `memoryestimate="no"`

Sets the memory usage estimate for Harmonic Balance. If yes, a memory estimate is printed in the log file. You can use this memory estimate to plan the computing resources before submitting harmonic balance runs. In memory estimate mode, a short simulation is performed first, and the engine exits after printing the estimate in the log file without saving harmonic balance results. You must turn it off to perform an actual simulation. Memory estimation is not recommended for simulations that require less than 500MB approximately. For PSS analysis, memory estimate mode does not apply unless flexbalance=yes. The estimate applies only to large-signal analysis, and does not include subsequent noise or other small-signal simulations.

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 source. 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. Omitting fundamental name in the `funds` parameter is an error that stops the simulation. If `maxharms` is not given, a warning message is issued, and the number of harmonics defaults to 1 for each fundamental.

For QPSS analyses, the role of some PSS parameters is extended compared to their role in PSS analysis. In QPSS, 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 shooting or HB 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 set errpreset to `moderate`. If accuracy is your main interest, set errpreset to `conservative`.

If you 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 in shooting.

© 2003-2014 All Rights Reserved.

Parameter defaults as a function of errpreset					
errpreset	reltol	relref	method	lteratio	maxstep
liberal	1e-3	sigglobal	gear2only	3.5	clock period/80
moderate	1e-4	siggloaal	gear2only	3.5	clock period/100
conservative	1e-5	sigglobal	gear2only	*	clock period/200

* : lteratio=10.0 for conservative `errpreset` by default. However, when the specified reltol <= 1e-4*10.0/3.5, lteratio is set to 3.5.

The new errpreset settings include a new default `reltol` that is actually an enforced upper limit for appropriate setting. An increase of `reltol` above the default is ignored by the simulator. You 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 are used

For HB, only `reltol` is affected by `errpreset`, and the effect is the same as that in shooting. However, `lteratio` remains 3.5 and `steadyratio` remains 1 with all values of `errpreset`.

With parameter hbhomotopy, you can specify harmonic balance homotopy selection methods. The possible values of parameter hbhomotopy and their meanings are as follows:

`hbhomotopy=tstab`: Simulator runs a transient analysis and generates an initial guess for harmonic balance analysis; it is recommended for nonlinear circuits or circuits with frequency dividers.

`hbhomotopy=source`: For driven circuit, simulator ignores tstab and adaptively increases the source power level; for oscillators, the simulator adaptively adjusts the probe magnitude until probe has no effect on the oscillators. It is recommended for strongly nonlinear or high Q circuits

`hbhomotopy=tone`: This method is valid only for multi-tone circuit. The simulator first solves a single-tone circuit by turning off all the tones except the first one, and then solves the multi-tone circuit by restoring all the tones and uses the single-tone solution as its initial guess; it is recommended for multitone simulation with a strong first tone.

© 2003-2014 All Rights Reserved.

`hbhomotopy=inctone`: Simulator firstly solves a single tone, then turns on moderate tones incrementally till all tones are enabled. It is recommended for circuits with one strong large tone.

`hbhomotopy=gsweep`: A resistor, whose conductance is g, is connected with each node, the sweep of g is controlled by gstart, gstop, and glog; it is recommended for circuits containing high-impedance or quasi-floating nodes.

Parameter Index

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

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

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. Unlike a conventional AC analysis that computes the response from a single stimulus to every node in the circuit, 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 required 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 specify the terminals, 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.

© 2003-2014 All Rights Reserved.

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, that is, the actual frequency, or if it is relative to the "relharmvec" sideband frequency. Possible values are `absolute` or `relative`.
11	`relharmvec=[...]`	Sideband- vector of QPSS harmonics- to which relative frequency sweep should be referenced.

Probe parameters

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

Output parameters

13	`stimuli=sources`	Stimuli used for xf analysis. Possible values are `sources` or `nodes_and_terminals`.
14	`sidevec=[...]`	Array of relevant sidebands for the analysis.
15	`clockmaxharm=7`	An alternative to the `sidevec` array specification, which automatically generates the array: [-clockmaxharm ... 0 ... +clockmaxharms][-maxharms(QPSS)[2]...0...maxharms(QPSS)[2]][...].
16	`freqaxis`	Specifies whether the results should be output versus the input frequency, the output frequency, or the absolute value of the input frequency. The default is `absin`. Possible values are `absin`, `in`, or `out`.
17	`save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
18	`nestlvl`	Levels of subcircuits to output.

© 2003-2014 All Rights Reserved.

Convergence parameters

19 `tolerance` — Relative tolerance for linear solver; the default value is 1.0e-9 for shooting-based solver and 1.0e-6 for harmonicbalance-based solver.

20 `relativeTol` — Relative tolerance for harmonicbalance-based linear solver; the default value is 1.0e-2.

21 `gear_order=2` — Gear order used for small-signal integration, 1 or 2.

22 `solver=turbo` — Solver type.
Possible values are `std` or `turbo`.

23 `lnsolver=gmres` — Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, `resgmres`, or `gmres_cycle`.

24 `resgmrescycle=short`
Restarts GMRES cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

25 `hbprecond_solver=autoset`

Select a linear solver for the GMRES preconditioner. Default is autoset. With autoset, the simulator will automatically select the appropriate precondtioner. The preconditioner affects the rate of convergence of the linear matrix solver used in periodic small-signal analysis. On occasion, when autoset is selected, the simulator may decide to switch to a different preconditioner after the analysis begins. When that happens, the simulator may issue a warning instructing you to choose a different preconditioner during subsequent runs. Although not required, choosing a different preconditioner according to the simulators instructions may speed up subsequent analyses.
Possible values are `basicsolver`,`blocksolver` and `autoset`.

26 `krylov_size=200` — The minimum iteration count of the linear matrix solver used in periodic small-signal analysis. After reaching krylov_size iterations, the iteration is forced to terminate because of the poor

© 2003-2014 All Rights Reserved.

rate of convergence. Increase krylov_size if the simulation reports insufficient norm reduction errors in GMRES.

Annotation parameters

27 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, `steps`, and `detailed_hb`.
28 `title`	Analysis title.

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, select the periodic small-signal input frequencies of interest by setting either the `clockmaxharm` or `sidevec` parameter. Sidebands are vectors in QPXF. Assuming that there is one large tone and one moderate tone in QPSS, a sideband K1 is represented as `[K1_1 K1_2]`. Corresponding frequency is as follows:

`K1_1 * fund` (large tone of QPSS) `+ K1_2 * fund` (moderate tone of QPSS)

In addition, 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,...K1_j..., K1_L} , K2, ... Kn.
```

The input signal frequency at each sideband is computed as follows:

```
f(in)= f(out) + SUM_j=1_to_L{Ki_j * fund_j(qpss)},
```

where, `f(out)` represents the (possibly swept) output signal frequency, and 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, Ki= {1, 0} for the RF input represents the first upper-sideband, while Ki= {-1, 0} for the RF input represents the first lower-sideband.

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- the first fundamental is affected; the rest- moderate tones- are limited by `maxharms`, specified for a QPSS analysis. Given `maxharms=[k1max k2max ... knmax]` in QPSS and `clockmaxharm=Kmax, all (2*Kmax + 1)*(2*k2max+1)*(2*k3max+1)*...*(2*knmax+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 area in which QPXF differs from XF). The `freqaxis` parameter is used to specify

© 2003-2014 All Rights Reserved.

whether the results should be output versus the input frequency (`in`), the output frequency (`out`), or the absolute value of the input frequency (`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 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 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. You 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 required, specify the terminals in the save statement. You must use the `:probe` modifier (for example, Rout:1:probe) or specify `useprobes=yes` on the options statement. If transfer functions from all terminals are required, specify `currents=all` and `useprobes=yes` on the options statement.

You can specify sweep limits by specifying the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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 values that the sweep parameter should take by using the `values` parameter. If you specify 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.

© 2003-2014 All Rights Reserved.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Reliability Analysis (reliability)

Description

This analysis computes the reliability for MOSFETs and the circuit. The three reliability modules are:

- MOSFET Hot-Carrier Injection (HCI) module

  Predicts transistor and circuit performance degradation due to HCI effects

- PMOSFET Negative Bias Temperature Instability (NBTI) module:

  Predicts PMOSFET and circuit performance degradation due to NBTI and NBTI recovery effects

- NMOSFET Positive Bias Temperature Instability (PBTI) module:

  Predicts NMOSFET and circuit performance degradation due to PBTI effects

Synopsis:

```
name reliability  <global options> {
      <reliability control statements> ...
      <stress simulation statements> ...
      <aging testbench statements> ...
      <aging/post-stress simulation statements> ...
}
```

Note: In the MMSIM10.1.0 release version, the <global options> are ignored.

The other four sections containing the statements should be listed in the specified order.

Definition

```
Name reliability parameter=value ...
```

Parameters

1 `Analysis parameters`
: Analysis title.

2 `time_age=[...]`
: The duration in the future at which the transistor degradation and degraded SPICE model parameters are to be calculated.

© 2003-2014 All Rights Reserved.

3 `value_deltad=0.0` Specifies the degradation value.

4 `mod_deltad=[...]` Specifies the name of a model whose lifetime is calculated. The model name must be the same as that specified in the .model card.

5 `type_maskdev=include`
Includes or excludes the specified devices or the devices that belong to the listed subcircuit or model during reliability analysis. Possible values are `include` or `exclude`.

6 `type_deg_ratio=include`

Includes or excludes the specified devices or the device during TMI Aging Simulation flow. Possible values are `include` and `exclude`.

7 `dev_deg_ratio=[...]` Specifies the instances to be included or excluded for degradation ratio for TMI Aging flow.

8 `mod_maskdev=[...]` Specifies the models for which the related devices should be included or excluded while performing reliability analysis.

9 `dev_maskdev=[...]` Specifies the instances to be included or excluded during reliability analysis.

10 `sub_maskdev=[...]` Specifies the subcircuits for which the related devices should be included or excluded while performing reliability analysis.

11 `value_reportmodelparameter=no`
Determines whether to print the stress and aged parameters in the.bm# file.
Possible values are `no` and `yes`.

12 `value_loadexternalcmi=no`

Load external user-defined CMI flag. Possible values are `no` and `yes`.

13 `debug_loadexternalcmi=no`

The flag of outputting thedebug information for loading external customerdefined CMI shared library.

© 2003-2014 All Rights Reserved.

14 `level_accuracy=1` Specifies methods to be used in the reliability simulation when performing integration and substrate current calculation.
Possible values are 1 and 2.

15 `value_minage=0.0` Specifies the smallest Age value for which degraded SPICE model parameters are calculated.

16 `type_igatemethod=calc`
Specifies the method used for obtaining the gate currents of MOSFETs.
Possible values are `calc` and `spice`.

17 `type_idmethod=ids`
Specifies how the simulator obtains the drain current (Id) of MOSFETs to perform reliability calculations.
Possible values are `ids` (Ids static current), `idrain` (dynamic gate current), and `idstatic` (terminal static current).

18 `file_dumpagemodel`
Output file name of dump age model.

19 `dev_dumpagemodel=[...]`
Specifies the instances whose aged modelcard will be output.

20 `file_urilib` Specifies URI library file name.

21 `uri_mode_urilib=agemos`
Specifies which method should be used to perform aging simulation. Currently, only the agemos mode is supported.
Note: appendage mode is not supported in the MMSIM10.1.0 release.
Possible values are `agemos` or `appendage`.

22 `debug_urilib=0` Specifies the debug mode for URI library. The value can be 0 or 1. When specified, a flag is added to the URI library indicating whether the debug information should be printed. debugMode=1 prints debug messages. The default value is 0.

23 `start_relxtran=0.0`
Specifies the start time of reliability analysis during transient simulation.

© 2003-2014 All Rights Reserved.

24 `stop_relxtran=0.0`
Specifies the stop time of reliability analysis during transient simulation. If stop_time is not specified, the software stops in .tran statement.

25 `type_isubmethod=calc`
Specifies the method used for obtaining the substrate currents of MOSFET.
Possible values are `calc` and `spice`.

26 `type_opmethod=calc`
Specifies whether the Igate or Isub value should be obtained from the SPICE models (for example, BSIM3 or BSIM4) or whether the internal Igate or Isub equation should be used.
Possible values are `calc` and `spice`.

27 `threshold_degsort=0.0`
Prints MOS transistors based on the threshold and number settings. The results are sorted in the descending order of degradation.

28 `number_degsort=0` Prints only the first <number> transistors having the highest degradations. For example, if number=100, the software prints the first 100 transistors with highest degradations.

29 `value_agelevelonly=[...]`
Sets the level for reliability analysis, which is essentially the age level number of the reliability analysis to be performed.

30 `type_gradual_aging_agestep=lin`
Sets the type of agestep, linear or logarithm, the default type is linear.
Possible values are `lin` or `log`.

31 `start_gradual_aging_agestep=0`
Specifies the start time of agestep in gradual aging flow, the default value is 0.

32 `stop_gradual_aging_agestep=0`
Specifies the stop time of agestep in gradual aging flow.

© 2003-2014 All Rights Reserved.

33 `total_step_gradual_aging_agestep=1`
Specifies the total step numbers of agestep in gradual aging flow..

34 `points_gradual_aging_agepoint=[...]`
Specifies points of agepoints.

35 `steps_gradual_aging_save=[...]`
Save the specified step of gradual aging result files.

36 `savedatainseparatedir_gradual_aging_save`
Dump the waveform files in separate directory for gradual aging result files.
Possible values are `no` or `yes`.

37 `gradual_alter_times=[...]`
Set time for gradualAging alter.

38 `gradual_alter_params=[...]`
Set params for gradualAging alter.

39 `gradual_alter_values=[...]`
Set values for gradualAging alter.

40 `type_simmode` Mode of reliability analysis.
Possible values are `stress`, `aging`, and `all`.

41 `file_simmode` File of simMode.

42 `tmifile_simmode` Input file for TMI Aging flow.

43 `dev_macrodevice=[...]`

Specifies the instances for macrodevice.

44 `value_combinedeg=no`
Determines whether to combine the external URI results with internal URI results in bo0 file. Possible values are no and yes.

45 `value_keep_aged_data=yes`
Determines whether to keep the value of the aged model parameters after reliability analysis. Possible values are `no` and `yes`.

© 2003-2014 All Rights Reserved.

46 `type_check_neg_aging=error`
Specifies the message type to check the negative value in bt0.
Possible values are `error` and `warn`. Default is `error`.

47 `clamp_check_neg_aging=no`
Clamps the negative age value in the bt0 file. Possible values are `no` and `yes`.

48 `tranflag=0` Specify the reliability transient analysis flag..

49 `stress_tran_name=[...]`
Specifies stress transient analysis name.

50 `aging_tran_name=[...]` Specifies aging transient analysis name.

51 `value_enablenativeage=no`
Enable the negative age value.
Possible values are `no` and `yes`.

52 `value_outputopdegrad=no`
Output degraded operation point value.
Possible values are `no` and `yes`.

53 `value_outputbinaryfile=no`
Specifies reliability analysis output file format.
Possible values are `no` and `yes`.

54 `value_single_analysis_type=no`
Only one analysis type in reliabilty analysis.
Possible values are `no` and `yes`.

55 `hci_deg_ratio=1.0` The degradation ratio for HCI effect.

56 `nbti_deg_ratio=1.0` The degradation ratio for NBTI effect.

57 `pbti_deg_ratio=1.0` The degradation ratio for PBTI effect.

58 `bti_deg_ratio=1.0` The degradation ratio for NBTI/PBTI effect.

59 `type_resetanalysisparam=tran`
Reset the analysis parametersin the reliability block.
Possible values are `tran`, `dc`, and `ac`.

© 2003-2014 All Rights Reserved.

60 `value_setagingmcparam=no`
set the parameters for agingmontecarlo analysis.
Possible values are `no` and `yes`.

61 `type_compact_aged_netlist=none`
compatible with RelXpert.
Possible values are `mod`, `subckt`, `all`, and `none`.

62 `path_gradual_aging_spice_path`
compatible with RelXpert.

63 `append_type_urilib=inline`
Compatible with RelXpert.
Possible values are `inline`, `sub`, and `dev`.

64 `value_enable_ade_process=no`
running reliability in ADE.
Possible values are `no` and `yes`.

65 `type_outputopdegrad=agemos`
Specify the type of output_op_degrad.
Possible values are `agemos` and `appendage`

66 `Annotation parameters`
Analysis title.

67 `annotate=sweep` Specifies the degree of annotation.
Possible values are `no`, `title`, `sweep`, and `status`.

68 `title` Analysis title.

Detailed Description and Examples:

`age time=[...]`

The duration in the future at which the transistor degradation and degraded SPICE model parameters are to be calculated. The degraded SPICE model parameters are used in aged circuit simulation. The calculated transistor degradation can be transconductance, linear or saturation drain current, degradation, threshold voltage shift, and or any other degradation monitor. While specifying the value, attach the suffix y (year), h (hour), or m (minute). There should be no space between the number and the suffix. For example, 10m, 1e-5sec. Note: Currently, specifying multiple time values is not supported.

`deltad value=<deltad_value>`

© 2003-2014 All Rights Reserved.

Requests the calculation of lifetime for each transistor under the circuit operating conditions by using the specified degradation value. You can use multiple deltad statements for different types of transistors. The degradation value can be transconductance, linear or saturation drain current degradation, threshold voltage shift, or any other degradation monitor, depending on the definitions of the lifetime parameters H and m. deltad_value can be in decimal notation (xx.xx) or in engineering notation (x.xxe+xx).

```
maskdev type={include | exclude} {sub=<subckt_list> | mod=<model_list> |
dev=<device_list>}
```

Includes or excludes the specified devices or the devices that belong to the listed subcircuit or model during reliability analysis.

Note: In the MMSIM10.1version, the include and exclude values are mutually exclusive.

```
report_model_param value={yes | no}(defaults to no)
```

Determines whether to print the stress and aged parameters in the.bm# file. Possible values are yes and no. When set to yes, the stress and aged parameters are printed to the .bm# file.

```
accuracy level={1 | 2}(defaults to 1)
```

Specifies methods used in the reliability simulation when performing integration and substrate current calculation. In other words, specifies trapezoidal integration when performing integrations and calculates Isub for Vgs < Vth. When set to 1, the software uses backward Euler integration and sets Isub=0 when Vgs < Vth. When set to 2, the software uses trapezoidal integration and calculates Isub when Vgs < Vth. Setting accuracy to 2 is more accurate, but increases simulation time when compared to when accuracy is set to 1.

```
minage value=<minage_value>
```

Specifies the smallest Age value for which degraded SPICE model parameters are calculated. This statement speeds up aging calculation by using stress SPICE model parameter if the transistor age value is smaller than the specified `minage_value`. `minage_value` can be in decimal notation (xx.xx) or in engineering notation (x.xxe+xx).

```
igatemethod type={calc | spice}(defaults to calc)
```

Specifies the method used for obtaining the gate terminal current of a MOSFET. During MOSFET HCI simulation, the gate terminal current is required for calculating the degradation value. The simulator can either calculate this value or obtain it from the SPICE output rawfile, if the SPICE simulator in use provides such an option. If this command is not used, the simulator calculates the gate terminal current by using its own model parameters. Possible values are calc (default) and spice. When calc is specified, the gate terminal current is calculated using the model parameters, and when spice is specified, the gate terminal current value is obtained from the SPICE output rawfile.

```
idmethod type={ids | idrain}(defaults to ids)
```


© 2003-2014 All Rights Reserved.

Specifies how the simulator obtains the drain current (Id) to perform reliability calculations. The following types of drain currents, which are available from SPICE, are supported by reliability analysis:

* Dynamic drain current (also called AC drain current)- the current that flows into the drain node.

* Static drain current (also called channel drain current, DC drain current, or Ids)- Possible values are ids (Ids static current) or idrain (dynamic drain current); the default is ids.

```
uri_lib file={"filename"} uri_mode={agemos | appendage} debug={0 | 1}
```

Loads the Unified Reliability interface (URI) shared library and specifies which method (`uri_mode`) should be used to perform aging simulation. Currently, only the agemos mode is supported.

Note: `appendage` mode is not supported in the MMSIM10.1 release.

```
relx_tran start=<start_time> stop=<stop_time>
```

Specifies the start and stop time for reliability analysis during transient analysis. If `stop_time` is not specified, the software stops in .tran statement.

```
isubmethod type={calc | spice}(defaults to calc)
```

Specifies the method used for obtaining substrate terminal current of a MOSFET. During MOSFET HCI simulation, the substrate terminal current is required for calculating the degradation value. The Virtuoso RelXpert simulator can either calculate this value or obtain it from the SPICE output rawfile, if the SPICE simulator in use provides such an option. If this command is not used, the simulator calculates the substrate terminal current by using its own model parameters. Possible values are calc (default) and spice. When calc is specified, the substrate terminal current is calculated using the model parameters, and when spice is specified, the substrate terminal current value is obtained from the SPICE output rawfile.

```
opmethod type={calc | spice}(defaults to calc)
```

Specifies whether the Igate or Isub value should be obtained from the SPICE models (for example, BSIM3 or BSIM4) or the internal Igate or Isub equation should be used. Possible values are calc or spice. calc calculates the gate and substrate terminal current by using the Cadence Igate and Isub model equations (default). spice obtains the gate and substrate terminal current value from the SPICE model.

```
degsort {threshold=<threshold_value> | number=<number_value>}
```

Prints MOS transistors based on the threshold value or number settings. The results are sorted in the descending order of degradation.

Note: The threshold and number arguments are mutually exclusive. Therefore, only one of them can be specified with degsort to print the sorted device degradation results. When threshold_value is specified, the transistors with degradation values greater than the

© 2003-2014 All Rights Reserved.

threshold value are printed. The threshold value can be in decimal notation (xx.xx) or in engineering notation (x.xxe+xx). When number_value is specified, only the first <number_value> transistors having the highest degradations are printed. For example, if number=100, the software prints the first 100 transistors with highest degradations.

```
agelevel_only value=[<level_value> <model_list>, <level_value> <model_list>, ...]
```

Specifies the age level for performing reliability analysis on the specified models. You can specify different age levels for different set of models.

Note: This option also supports the URL-defined agelevel statement. If model names are not specified, the simulation is performed on all the devices at the specified age level. The following levels can be used to specify Cadence internal ageMOS models:

* 1: Specifies HCI reliability analysis.

* 2: Specifies NBTI reliability analysis.

* 3: Specifies PBTI reliability analysis.

Example:

```
    rel reliability {
        // reliability control statements
        age time = [10h 20y 30y]
        deltad value = 0.1
        accuracy level = 2
        agelevel_only value=[0 nch, 1 pch]
        relx_tran start=1us stop=10us
        idmethod type=idrain
        maskdev type=include mod=[pch] dev=[mdut6] sub=[inv]
        minage value = 0.00001
        report_model_param value=yes
        uri_lib file="./libURI.so" uri_mode=agemos debug=1
        degsort number=100
        igatemethod type=spice
        isubmethod type=spice
        opmethod type=spice
        // stress/stress statements.
        tran_stress tran start=0 step=1us stop=10us
        // aging testbench statements.
        change1 alter param=rel_temp value=125
        // aging simulation statements.
        tran_aged tran start = 0 step = 1us stop = 10us
}
```


© 2003-2014 All Rights Reserved.

Parameter Index

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

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

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=1.0e-6 V`	Voltage absolute tolerance convergence criterion.
4	`iabstol=1.0e-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`.

© 2003-2014 All Rights Reserved.

Convergence parameters

6 `homotopy=all` — Method used when no convergence on initial attempt of DC analysis. You can specify methods and their orders by giving vector setting such as homotopy=[source ptran gmin].
Possible values are `none`, `gmin`, `source`, `dptran`, `ptran`, `arclength`, `fast_dptran`, `fast_ptran`, and `fast_gmin`.

7 `newton=normal` — Method used on DC analysis. Users can specify methods newton=[none | normal], and the default value is normal..
Possible values are `none` and `normal`.

8 `limit=dev` — Limiting algorithms to aid DC convergence.
Possible values are `delta`, `log`, or `dev`, and `12`.

9 `gmethod=dev` — Stamp gdev, gnode or both in the homotopy methods (other than dptran). See below for more information.
Possible values are `dev`, `node`, or `both`.

10 `dptran_gmethod=node` — Stamp gdev, gnode, or both in the dptran (homotopy) methods.
Possible values are `dev`, `node`, or `both`.

11 `try_fast_op=yes` — This feature often speeds up the DC solution. For hard to converge designs, this feature fails and other methods are applied. In corner cases, this feature may have negative effects. If the DC analysis is unusually slow or if the memory usage of various processes keeps growing or if DC gets stuck even before homotopy methods start, try setting this option to no.
Possible values are `no` or `yes`.

Component parameters

12 `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`, `spiceplus`, or `eldo`.

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

© 2003-2014 All Rights Reserved.

Error-checking parameters

14 `diagnose=no`	Print additional information that might help diagnose accuracy and convergence problems. Possible values are `no` or `yes`.
15 `opptcheck=yes`	Check operating point parameters against soft limits. Possible values are `no` or `yes`.

Resistance parameters

16 `gmin=1e-12 S`	Minimum conductance across each nonlinear device.
17 `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`.
18 `gmindc=1e-12 S`	Minimum conductance across each nonlinear device in DC analysis.If gmindc is not given explicitly, the value of gmindc will be equal to gmin.Default value is 1.0e-12.
19 `rforce=1 Ω`	Resistance used when forcing nodesets and node-based initial conditions.

Quantity parameters

20 `quantities=no`	Print quantities. If quantities=min, the simulator prints out all defined quantities; if quantities=full, the simulator also prints a list of nodes and their quantities. Possible values are `no`, `min`, or `full`.

Annotation parameters

21 `narrate=yes`	Narrate the simulation. Possible values are `no` or `yes`.
22 `annotate=no`	Degree of annotation. Possible values are `no` and `title`.
23 `debug=no`	Give debugging messages. Possible values are `no` or `yes`.

© 2003-2014 All Rights Reserved.

24 `info=yes` Give informational messages.
Possible values are `no` or `yes`.

25 `note=yes` Give notice messages.
Possible values are `no` or `yes`.

26 `maxnotes=5` Maximum number of times a notice is issued per analysis. Note that this option has no effect on notices issued as part of parsing the netlist. Use the -maxnotes command-line option to control the number of all notices issued.

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

28 `maxwarns=5` Maximum number of times a warning message is issued per analysis. Note that this option has no effect on warnings issued as part of parsing the netlist. Use the -maxwarns command-line option to control the number of all warnings issued.

29 `maxwarnstologfile=5`
Maximum number of times a warning message is printed to the log file per analysis. Note that this option has no effect on warnings printed as part of parsing the netlist. Use the -maxwarnstolog command-line option to control the number of all warnings printed to the log file.

30 `maxnotestologfile=5`
Maximum number of times a notice message is printed to the log file per analysis. Note that this option has no effect on notices printed as part of parsing the netlist. Use the -maxnotestolog command-line option to control the number of all notices printed to the log file.

31 `error=yes` Generate error messages.
Possible values are `no` or `yes`.

32 `digits=5` Number of digits used when printing numbers.

33 `measdgt=0` Number of decimal digits in floating point numbers in measurement output in mt0 format.

© 2003-2014 All Rights Reserved.

34 `notation=eng` The notation to be used for displaying real numbers to the screen.
Possible values are `eng`, `sci`, or `float`.

Matrix parameters

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

36 `pivrel=0.001` Relative pivot threshold.

37 `pivabs=0` Absolute pivot threshold.

38 `preorder=partial` Try this option when simulation runs out of memory or if the simulation is unreasonably slow for the size of your design. It controls the amount of matrix preordering that is done and may lead to much fewer matrix fill-ins in some cases. Known cases include designs with very large number of small resistors and large number of behavioral instances containing voltage based equations.
Possible values are `partial` or `full`.

39 `limit_diag_pivot=yes`
If set to yes, there is a limit on the number of matrix fill-ins when selecting a pivot from a diagonal. For backward compatibility set this to no.
Possible values are `no` or `yes`.

40 `rebuild_matrix=no`
If yes, rebuild circuit matrix at the beginning of ac, dc, dcmatch, montecarlo, pz, stb, sweep, tdr, and tran analyses. This is to ensure consistent matrix ordering at the beginning of the analyses for consistent results. Notice that rebuild circuit matrix can incur performance overhead.
Possible values are `no` or `yes`.

41 `icversion=1` Convert initial condition to initial guess, when .ic statements exist in netlist and there are no other options to set IC or nodeset.

42 `minstepversion=1` Minstep algorithm flow control. If `minstepversion=0`, desperate big jump is taken when minstep is reached. If `minstepversion=1`, a forward/backward search algorithm is

© 2003-2014 All Rights Reserved.

taken to find a converged solution at small step size. By default, minstepversion=1.

43 `try_hard_in_disaster=yes`

When `minstepversion=1`, this option means whether simulator should try hard to search for a converged solution in disaster stage.
Possible values are `no` and `yes`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

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`	Action to be taken if the command returns nonzero error status. Possible values are `continue` or `quit`.
3	`annotate`	Degree of annotation. Possible values are `no`, `title`, or `yes`.

© 2003-2014 All Rights Reserved.

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 file name 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 the Spectre native format or the 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`, without a `dev` or `mod` parameter. You can sweep a netlist parameter by giving the parameter name without a `dev` or `mod` parameter. After the analysis is complete, the modified parameter returns to its original value.

Definition

```
Name sp parameter=value ...
```

Parameters

1	`prevoppoint=no`	Use the 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.

© 2003-2014 All Rights Reserved.

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	`ports=[...]`	List of active ports. Ports are numbered in the order given.

State-file parameters

16	`readns`	File that contains estimate of DC solution (nodeset).
17	`write`	DC operating point output file at the first step of the sweep.
18	`writefinal`	DC operating point output file at the last step of the sweep.

Initial condition parameters

19	`force=none`	Which set of initial conditions to use. Possible values are `none`, `node`, `dev`, or `all`.
20	`readforce`	File that contains initial conditions.

© 2003-2014 All Rights Reserved.

21 `skipdc=no` Skip the DC analysis.
Possible values are `no` or `yes`.

22 `useprevic=no` If set to yes or ns, Use the converged initial condition from previous analysis as ic or ns.
Possible values are `no`, `yes` or `ns`.

Output parameters

23 `file` S-parameters output file name.

24 `mode="ss"` S-parameters mode selector. Can be mm for mixed-mode.

25 `datafmt=spectre` Data format of the S-parameter output file.
Possible values are `spectre` or `touchstone`.

26 `paramtype=s` Output parameter type.
Possible values are `s`, `y`, `z`, or `yz`.

27 `datatype=realimag`
Data type of the S-parameter output file.
Possible values are `realimag`, `magphase`, or `dbphase`.

28 `noisedata=no` Should noise data be saved to the S-parameter output file; if yes, in what format.
Possible values are `no`, `twoport`, or `cy`.

29 `oppoint=no` Should operating point information be computed; if yes, where should it be sent. Operating point information is not printed if the operating point computed in the previous analysis remains unchanged.
Possible values are `no`, `screen`, `logfile`, or `rawfile`.

Noise parameters

30 `donoise=no` 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`.

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

32 `iprobe` Input probe. Refer the output noise to this component.

© 2003-2014 All Rights Reserved.

Convergence parameters

33 `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

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

If the list of active ports is specified with the `ports` parameter, 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, 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).

If the `mode` parameter is set to mm, differential and common-mode S-parameters (denoted as mixed-mode S-parameters) are calculated. When `mode=mm`, there must be 2N, with N > 1, active port statements in the circuit. The mixed-mode S-parameters are calculated referring to the pairing of the ports, with the port numbers ordered in pair as (1,2) (3,4), and so on in the ports list. With mm, Spectre calculates differential-to-differential, differential-to-common, common-to-differential, and common-to-common S-parameters. A combination of mixed-mode and standard S-parameters is calculated if the mode parameter is set to, say, m12m34s5. Then, additional differential-to-standard, common-to-standard, standard-to-differential, and standard-to-common S-parameters are calculated. In the example of mode=m12m34s5, the standard single-end port is port number 5, the two mixed-mode port pairs are (1,2) and (3,4); with Spectre placing restriction of the number on active ports to 5 given in the port list.

You can specify sweep limits by specifying the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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

© 2003-2014 All Rights Reserved.

significant component or circuit parameter has changed. However, if an operating point was computed during a previous analysis, you can set `prevoppoint=yes` to avoid recomputing it. For example, if `prevpoint=yes`, and the previous analysis was a transient analysis, the operating point is the state of the circuit at the final time point.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Stability Analysis (stb)

Description

The STB analysis linearizes the circuit about the DC operating point and computes the loop gain and 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 at each step. You can sweep the circuit temperature by giving the parameter name as `temp`, without a `dev` or `mod` parameter. You can sweep a netlist parameter by giving the parameter name without a `dev` or `mod` parameter. After the analysis is complete, the modified parameter returns to its original value.

Definition

```
Name stb parameter=value ...
```

Parameters

1	`prevoppoint=no`	Use the 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.

© 2003-2014 All Rights Reserved.

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.

16 `localgnd` Node name of local ground. If not specified, the probe is referenced to global ground.

State-file parameters

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

18 `write` DC operating point output file at the first step of the sweep.

19 `writefinal` DC operating point output file at the last step of the sweep.

Initial condition parameters

20 `force=none` Which set of initial conditions to use.
Possible values are `none`, `node`, `dev`, or `all`.

21 `readforce` File that contains initial conditions.

22 `skipdc=no` Skip the DC analysis.
Possible values are `no` or `yes`.

© 2003-2014 All Rights Reserved.

23 `useprevic=no`	If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns. Possible values are `no`, `yes` or `ns`.

Output parameters

24 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
25 `nestlvl`	Levels of subcircuits to output.
26 `oppoint=no`	Should operating point information be computed; if yes, where should it be sent. Operating point information would not be output if the operating point computed in the previous analysis remains unchanged. Possible values are `no`, `screen`, `logfile`, or `rawfile`.

Convergence parameters

27 `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

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

You can define sweep limits by specifying the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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

© 2003-2014 All Rights Reserved.

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.

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 by 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 required solution. 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 significantly speed convergence.

When you simulate the same circuit multiple times, it is recommended that you use both the `write` and `readns` parameters and assign 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.

During the initial operating point DC analysis, you may force some of the circuit variables to the values given in the `ic` file, `ic` statements, or `ic` parameter on the capacitors and inductors. The `ic` parameter controls the interaction of the various methods for setting the force values. The effects of individual settings are as follows:

`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 an `ic` file with the `readforce` parameter, force values from the file are used, and any `ic` statements are ignored.

After you specify the initial conditions, Spectre computes the DC operating point 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`).

© 2003-2014 All Rights Reserved.

Understanding Loop-Based and Device-Based Algorithms

Two algorithms- loop-based and device-based are available for small-signal stability analysis. Both algorithms are based on the calculation of Bodes 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 the parameter points to a current probe or voltage source instance, the loop-based algorithm is run; when the parameter points to a supported active device instance, the device-based algorithm is run.

Loop-Based Algorithm

The loop-based algorithm calculates the true loop gain, which 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 also for 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 that 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.

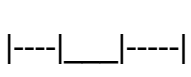
For the loop based algorithm, a probe needs to be placed on the feedback loop.The common way is to insert a current probe or voltage source instance in the netlist, and the `probe` parameter points to this instance. The second way to insert the probe is by specifying the terminal of some instance.

The snytax format :

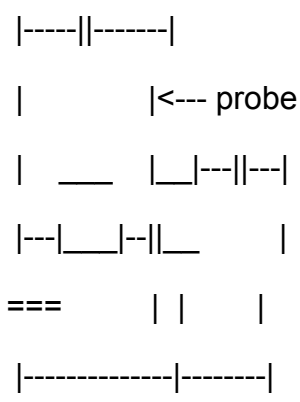
```
probe = X:n
```

`X` can be an instance or a subcircuit instance and `n` is the terminal index. The second way has a limitation: We can not specify a branch whose current consists of more than one instance's terminal current. A current probe or voltage source instance needs to be inserted manually.

This can be shown below:



© 2003-2014 All Rights Reserved.



No instance can specify the probe's location.

Device-Based Algorithm

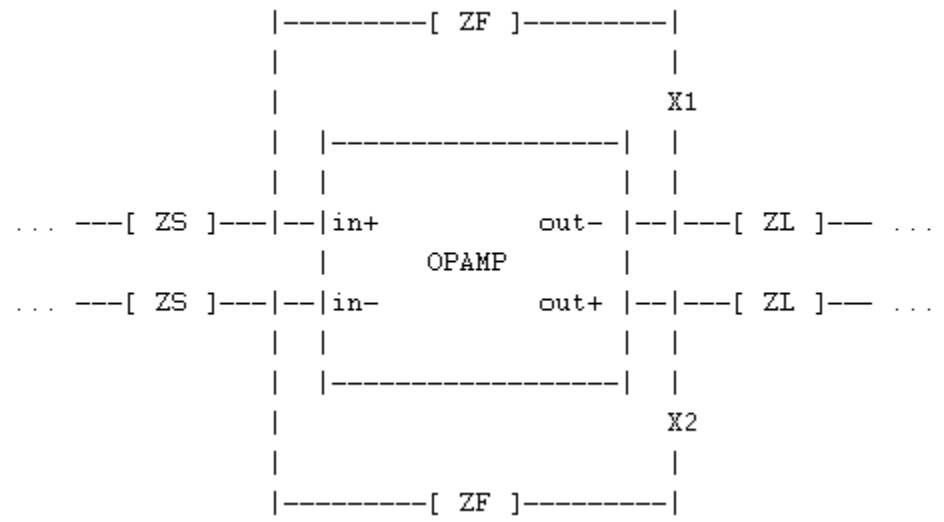
The device-based algorithm calculates the loop gain around a particular active device. This algorithm is often applied to assess the stability of circuit design in which local feedback loops cannot be neglected; the loop-based algorithm cannot be performed for these applications because the local feedback loops are inside the devices and cannot be accessed from the schematic or netlist level to insert the `probe` component.

With the `probe` parameter pointing to a particular active device, the dominant controlled source in the device is 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, so that nulling the dominant gain source of this device renders the whole network passive.

© 2003-2014 All Rights Reserved.

Stability Analysis of Differential Feedback Circuits

A balanced fully differential feedback circuit is illustrated below:



The feedback loops are broken at X1 and X2, with x1in and x2in being the input side nodes and x1out and x2out being the output side nodes. The following subcircuit connects these four nodes together:

```
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
```

If the localgnd parameter is specified, the above subcircuit should be modified as follows:

```
subckt diffprobe x1in x2in x1out x2out localgnd
    ibranch inout x1out iprobe
    vinj inout x1in iprobe
    evinj x2in x2out x1in x1out vcvs gain=0
    fiinj localgnd 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
```


© 2003-2014 All Rights Reserved.

```
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 corresponding to each parameter name indicates where to find the description of that parameter.

© 2003-2014 All Rights Reserved.

Sweep Analysis (sweep)

Description

The `sweep` analysis sweeps a parameter, running 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`, without a `dev`, `mod`, or `sub` parameter. You can sweep a top-level netlist parameter by giving the parameter name without a `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 is complete, the modified parameter returns to its original value.

Definition

```
Name sweep 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.

© 2003-2014 All Rights Reserved.

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 specifying the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) and determine whether the sweep is linear or logarithmic. If you do not specify 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
}
```


© 2003-2014 All Rights Reserved.

Parameter Index

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

© 2003-2014
All Rights Reserved.

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.
3	`start=-0.1 stop`	Time output waveforms begin.
4	`smoothing=2`	Window smoothing parameter (useful range is 0 to 15).
5	`vel=1`	Propagation velocity of medium normalized to c.
6	`points=64`	Number of time points.
7	`ports=[...]`	List of active ports. If not given, all ports are used.
8	`readns`	File that contains estimate of DC solution (nodeset).
9	`useprevic=no`	If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns. Possible values are `no`, `yes` or `ns`.
10	`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`.
11	`annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
12	`title`	Analysis title.

© 2003-2014 All Rights Reserved.

13 `oppoint=no` — Should operating point information be computed; if yes, where should it be sent. Operating point information would not be output if the operating point computed in the previous analysis remains unchanged.
Possible values are `no`, `screen`, `logfile`, or `rawfile`.

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

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 corresponding to each parameter name indicates where to find the description of that parameter:

© 2003-2014 All Rights Reserved.

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	`tpoints=[...] s`	Multiple of pairs<pstep, stop>.
3	`start=0 s`	Start time.
4	`pstep (s)`	print step.
5	`outputstart=start s`	Output is saved only after this time is reached.
6	`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. If the value is spice, autostop is consistent with spice simulator. Possible values are `no`, `yes`, or `spice`.

Time-step parameters

7	`maxstep (s)`	Maximum time step. The default is derived from `errpreset`.
8	`step=0.001 (stop-start) s`	Minimum time step used by the simulator solely to maintain the aesthetics of the computed waveforms.

© 2003-2014 All Rights Reserved.

9	`minstep (s)`	Minimum time step. If specified, the error tolerance requirements may be ignored when step size is less than minstep.

Initial-condition parameters

10	`ic=all`	The value to be used to set initial condition. Possible values are `dc`, `node`, `dev`, or `all`.
11	`skipdc=no`	If yes, there is no dc analysis for transient. Possible values are `no`, `yes`, `useprevic`, `waveless`, `rampup`, `autodc`, or `sigrampup`.
12	`readic`	File that contains initial condition.
13	`useprevic=no`	If set to yes or ns, Use the converged initial condition from previous analysis as ic or ns. Possible values are `no`, `yes` or `ns`.
14	`linearic=no`	Enable linear IC method to calculate initial conditions automatically from a type of stability analysis in the range [0.5*oscfreq, 1.5*oscfreq]. Overrides user-defined initial conditions, if instability is detected. Possible values are `no` and `yes`.
15	`oscfreq=0.0`	Estimation of the oscillation frequency when linear IC method is enabled.

Convergence parameters

16	`readns`	File that contains estimate of initial transient solution.
17	`cmin=0 F`	Minimum capacitance from each node to ground.

State-file parameters

18	`write`	File to which initial transient solution is to be written.
19	`writefinal`	File to which final transient solution is to be written.
20	`ckptperiod`	Checkpoint the analysis periodically using the specified period.

© 2003-2014 All Rights Reserved.

21 `saveperiod` Save the tran analysis periodically on the simulation time.

22 `saveperiodhistory=no` Maintains the history of saved files. If `yes`, store all the saved files. Possible values are `no` and `yes`.

23 `saveclock (s)` Save the tran analysis periodically on the wall clock time. The default is 1800s for Spectre. This parameter is disabled in APS mode by default.

24 `savetime=[...]` Save the analysis states into files on the specified time points.

25 `savefile` Save the analysis states into the specified file.

26 `recover` Specify the file to be restored.

Integration method parameters

27 `method` Integration method. Default derived from `errpreset`. Possible values are `euler`, `trap`, `traponly`, `gear2`, `gear2only`, `trapgear2`, or `trapeuler`.

Emir output parameters

28 `emirformat=none` Format of the EM/IR database file. Possible values are `none` or `vavo`.

29 `emirstart (s)` EM/IR start time.

30 `emirstop (s)` EM/IR stop time.

31 `emirfile` Name of the EM/IR database file. Default is %A_emir_vavo.db. The file will be output to raw directory.

Accuracy parameters

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

33 `relref` Reference used for the relative convergence criteria. The default is derived from `errpreset`.

© 2003-2014 All Rights Reserved.

Possible values are `pointlocal`, `alllocal`, `sigglobal`, or `allglobal`.

34 `lteratio` — Ratio used to compute LTE tolerances from Newton tolerance. The default is derived from `errpreset`.

35 `fastbreak=no` — If yes, VHDLAMS Break statement is handled using faster Verilog method.
Possible values are `no` or `yes`.

36 `d2aminstep=0` — Minimum stepsize that can be taken when there is a D2A event. If this is zero, the simulators min step size is chosen.

37 `fastcross=discrete`
Using limited threshold reject method for fast cross detection.
Possible values are `no`, `yes`, `discrete`, or `cm`.

38 `transres=1e-9 stop s`
Transition resolution. The transient analysis attempts to stop at corners of input waveforms (for example, corners of rising/falling edge of a pulse). If such events occur within a time less than transres, the analysis combines the events into one and forces only one time point. The rest of the steps are determined by error control. This may lead to loss of detail.

39 `lteminstep=0.0 s` — Local truncation error is ignored if the step size is less than lteminstep.

40 `ltethstep=1e-12 s` — LTE tolerance can be relaxed for signal with discontinuity when step size is less than ltethstep.

Annotation parameters

41 `annotate=sweep` — Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `estimated`, `steps`, `iters`, `detailed`, `rejects`, `alliters`, `detailed_hb`, and `internal_hb`.

42 `annotateic=no` — Degree of annotation for initial condition.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, `iters`, `detailed`, or `rejects`.

43 `title` — Analysis title.

© 2003-2014 All Rights Reserved.

Output parameters

44 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
45 `nestlvl`	Levels of subcircuits to output.
46 `oppoint=no`	Should operating point information be computed for initial timestep; if yes, where should it be sent. Possible values are `no`, `screen`, `logfile`, or `rawfile`.
47 `skipstart=0 s`	The time to start skipping output data.
48 `skipstop=stop s`	The time to stop skipping output data.
49 `skipcount=1`	Save only one of every skipcount points.
50 `strobeperiod=0 s`	The output strobe interval (in seconds of transient time).
51 `strobedelay=0 s`	The delay (phase shift) between the skipstart time and the first strobe point.
52 `strobeoutput=strobeonly`	Specifies which time points to output during strobe. Possible values are `strobeonly`, `all`, `none`, or `faulttimes`.
53 `strobestep=0 s`	Equivalent to strobeperiod.
54 `strobefreq`	The reciprocal of strobeperiod (strobestep).
55 `strobestart=0 s`	Equivalent to skipstart.
56 `strobestop=stop s`	Equivalent to skipstop.
57 `strobetimes=[...] s`	Times in ascending order when strobe output performed.
58 `progress_t`	Print out annotate message every interval specified by progress_t in terms of minutes. Note that this degrades performance.
59 `progress_p`	Print out the annotate message every progress_p percent of transient time. Note that this degrades performance.

© 2003-2014 All Rights Reserved.

60 `compression=no` Perform global waveform compression.
Possible values are `no`, `all`, and `wildcardonly`.

61 `compvabstol=1.0e-3 V`
Absolute voltage tolerance for waveform compression.

62 `compiabstol=1.0e-12 A`
Absolute current tolerance for waveform compression.

63 `compreltol=0.001` Relative tolerance for waveform compression.

64 `complvl` Enables waveform compression for specified hierarchy level and below (top level=1). All levels above specified level are not compressed. Complvl has higher priority than global compression statement.

65 `flushpoints` Flush all unwritten data in the buffer to outputs after number of calculated points.

66 `flushtime (s)` Flush unwritten data in the buffer to outputs after real time has elapsed.

67 `flushofftime (s)` Real time to stop flushing outputs.

68 `flushperiod (s)`

69 `infoname` Name of the info analysis to be performed at each time point in the `infotimes` array. There is no individual run for this info analysis.

70 `infotimes=[...] s` Times when the analysis specified by `infoname` is performed.

71 `acnames=[...]` Names of ac, noise, sp, stb, or xf analyses to be performed at each time point in the `actimes` array. The named small-signal analyses are not run separately, but only as part of the transient analysis.

72 `actimes=[...] s` Times when analyses specified in `acname` array are performed.

Newton parameters

73 `maxiters=5` Maximum number of iterations per time step.

© 2003-2014 All Rights Reserved.

74 `dcmaxiters=150` Maximum number of dc iterations in tranFindInitialState.

75 `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

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

Transient noise parameters

77 `noisefmax=0 Hz` The bandwidth of pseudorandom noise sources. A valid (nonzero) noisefmax turns on the noise sources during transient analysis. The maximum time step of the transient analysis is limited to 0.5/noisefmax.

78 `noisescale=1` Noise scale factor applied to all generated noise. Can be used to artificially inflate the small noise to make it visible above transient analysis numerical noise floor, but it should be small enough to maintain the nonlinear operation of the circuit.

79 `noiseseed` Seed for the random number generator. Should be positive integer. Specifying the same seed allows you to reproduce a previous experiment.

80 `noisefmin (Hz)` If specified, the power spectral density of the noise sources depends on the frequency in the interval from noisefmin to noisefmax. Below noisefmin, the noise power density is constant. The default value is noisefmax, so that only white noise is included by default, and noise sources are evaluated only at noisefmax for all models. 1/noisefmin cannot exceed the requested time duration of transient analysis.

81 `noisetmin (s)` Time interval between noise source updates. Default is 0.5/ noisefmax. Smaller values produce smoother noise signals, but reduce time integration step.

82 `noiseupdate=step` Forces evaluation of bias-dependent device noise sources with noisetmin step (`fmax`), or at each time step, even if it is smaller

© 2003-2014 All Rights Reserved.

	than noisetmin value (`step`). Possible values are `step`, `fmax` or `_stepold`.
83 `noiseon=[...]`	The list of instances to be considered as noisy during transient noise analysis.
84 `noiseoff=[...]`	The list of instances to be considered as not noisy during transient noise analysis.

Dynamic parameters

85 `param`	Name of the parameter to be updated to a different value with time during tran. You can use `param=isnoisy` with `param_vec=[...]` to turn On or Off the transient noise in time windows. For example, `param=isnoisy param_vec=[0ns 0 100ns 1 500ns 0 ]`. The transient noise is OFF (param value is 0) from time 0 to 100ns and the noise is ON (param value is 1) from 100ns to 500ns and OFF from 500ns to stop time.
86 `paramset`	Name of dynamic parameter set.
87 `param_vec=[...]`	The `time_value` points to `param=name`.
88 `param_file`	The file that contains the `time_value` points to `param=name`.
89 `sub`	Subcircuit instance for the subckt instance parameter given in `param=name`.
90 `mod`	Device model for the model parameter given in `param=name`.
91 `dev`	Device instance for the instance parameter given in `param=name`.
92 `param_step`	Defines how often to update the dynamic parameter values. If param_step=0, it updates the parameter value on given time point.
93 `faultreadic`	File that contains initial conditions for fault sensitivity analysis.

You can 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

© 2003-2014 All Rights Reserved.

controls the interaction of various methods of setting the initial conditions. The effects of individual settings are as follows:

`ic=dc`: All initial conditions are ignored, and the DC solution is used.

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

After 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 is trivial or is given in the file that you specified by using the `readic` parameter, or if the `readic` parameter is not given, by 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 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 as follows:

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

© 2003-2014 All Rights Reserved.

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

`skipdc=useprevic`: Initial solution obtained from the previous analysis is used.

`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 the start parameter. 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 by 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 required solution. 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, it is recommended that you use both `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 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. Because differential equations have an infinite number of solutions, a complete set of initial conditions must be specified to identify the required solution. Any initial conditions that you do not specify are computed by the simulator to be consistent. The transient waveforms then

© 2003-2014 All Rights Reserved.

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 as follows:

`method=euler`: Backward-Euler is used exclusively.

`method=traponly`: Trapezoidal rule is used almost exclusively.

`method=trapeuler`: 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.

`method=trap`: An advanced version of trap that uses all three integration methods.

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 errors in the computation of the circuit dynamics (such as time constants), relative to `reltol` and `abstol` by setting the `lteratio` parameter.

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

`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.

© 2003-2014 All Rights Reserved.

`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 node's 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	lteratio
liberal	* 10	sigglobal	trapgear2	Interval/50	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.

`errprest` also controls the LTE Check:

	Liberal	Moderate	Conservative
LTE Check	Caps/Inds	Loose nodes	strict nodes

It controls how the simulator follows signals other than capacitor voltages and inductor currents. When errpreset=liberal, the timestep is not controlled to follow these signals. When errpreset=moderate, the timestep is reduced to follow large changes in these signals. When errpreset=conservative, the timestep is reduced to follow small changes in these signals.

If the circuit you are simulating has 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.

© 2003-2014 All Rights Reserved.

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 the 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 that the data is computed, and not interpolated.

The skipping method is controlled by `skipcount`. If this is set to N, 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.

Dynamic Parameters during Transient Analysis

The parameters defined in the `Dynamic parameters` section allows you to change temperature, design parameters or some option parameters (reltol, residualtol, vabstol, iabstol, and isnoisy) during transient simulation.

Example1: change temperature during tran with param_step=0(default).

```
tran1 tran stop=0.5u param=temp param_vec=[0ns 20 50ns 25]
```

In this tran run, the temperature is 20C from 0ns-50ns, then it changes to 25C at 50ns. After tran is done, the temperature is reset back to its default value.

You can also define time value pairs in a file and give the file name through parameter param_file.

The format of the file is defined as follows:

```
; comments
 tscale tscale_value
 time  value
 20  50.0
 30  60.0
```

where, the comment line starts with a semicolon (;), tscale is a keyword, and tscale_value is a value, such as 1.0e-6 or 1.0e-9, that is applied to each time point under the time column. time and value are two key words that identify the time and value columns. The values under

© 2003-2014 All Rights Reserved.

the time column define the time points and each time point is scaled by tscale_value. The values under the value column define the values for the dynamic parameter.

Note that no unit is supported in the file format.

Example2: change temperature during tran with param_step=10ns

```
tran1 tran stop=0.5u param=temp param_vec=[0ns 20 50ns 25] param_step=10ns
```

In this tran run, the temperature is interpolated with slope (25-20)/(50ns-0ns) and updated every param_step (10ns).

Example3: change design parameter.

```
tran1 tran stop=0.5u param=gain sub=x1 param_vec=[0 5 1u 20]
```

Example4: turn On and Off transient noise in time windows.

```
tran1 tran stop=0.5u noisefmax=10G noiseseed=1
param=isnoisy param_vec=[0ns 0 100ns 1 500ns 0 ]
```

The transient noise is OFF from time 0 to 100ns. Noise is ON from 100ns to 500ns and noise is OFF from 500ns to 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, because every signal has the same x value. If `compression=all`, Spectre writes data to the output file only if the signal value changes by at least two times the convergence criteria. 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=all` results in a smaller output data file. If the signals in your circuit move around a lot, setting `compression=all` results in a larger output data file. The `compvabstol` and `compiabstol` options can be used to control output compression abstol for voltage and current respectively.

Parameter Index

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

© 2003-2014 All Rights Reserved.

© 2003-2014
All Rights Reserved.

© 2003-2014 All Rights Reserved.

Special Current Saving Options (uti)

Description

This command is used to report the dynamic current of all devices connected to the specified voltage source during dynamic simulation.

Definition

```
Name uti parameter=value ...
```

Parameters

1	`signal`	specify the name of signal for which voltage drop must be calculated.
2	`start`	specify the start name of the measure.
3	`clockcycle (s)`	specify the length of the clock cycle.
4	`intervals`	specify the number of measurement intervals within a clock cycle.
5	`cycles`	specify the number of clock cycles for which this measure will be calculated.
6	`filename`	specify the root name of the files containing the peak, average and RMS tap currents for this measure.
7	`termflag`	specify the terminals which will be output.
8	`method`	specify the method used in post-processing clock analysis data.
9	`namemangling`	specify namemangling.

© 2003-2014 All Rights Reserved.

Parameter Index

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

© 2003-2014 All Rights Reserved.

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 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 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. You 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 as follows:

Source	Output Type		Source
Type	voltage	current	Amplitude
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 `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

© 2003-2014 All Rights Reserved.

series with the terminal. By default, the transfer functions from a small set of terminals are computed. If transfer functions from specific terminals are required, specify the terminals in the save statement. You must use the `:probe` modifier (for example, Rout:1:probe) or specify `useprobes=yes` on the options statement. If transfer functions from all terminals are required, specify `currents=all` and `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 `temp` without a `dev` or `mod` parameter. You can sweep a netlist parameter by giving the parameter name without a `dev` or `mod` parameter. After the analysis is complete, the modified parameter returns to its original value.

Definition

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

The optional terminals (p and n) specify the output of the circuit. If you do not specify the terminals, 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.
6	`step`	Step size, linear sweep.
7	`lin=50`	Number of steps, linear sweep.
8	`dec`	Points per decade.

© 2003-2014 All Rights Reserved.

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).

17 `write` DC operating point output file at the first step of the sweep.

18 `writefinal` DC operating point output file at the last step of the sweep.

Initial condition parameters

19 `force=none` Which set of initial conditions to use.
Possible values are `none`, `node`, `dev`, or `all`.

20 `readforce` File that contains initial conditions.

21 `skipdc=no` Skip the DC analysis.
Possible values are `no` or `yes`.

22 `useprevic=no` If set to `yes` or `ns`, use the converged initial condition from previous analysis as ic or ns.
Possible values are `no`, `yes` or `ns`.

© 2003-2014 All Rights Reserved.

Output parameters

23 `stimuli=sources`	Stimuli used for xf analysis. Possible values are `sources` or `nodes_and_terminals`.
24 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
25 `nestlvl`	Levels of subcircuits to output.
26 `oppoint=no`	Should operating point information be computed; if yes, where should it be sent. Operating point information isnot printed if the operating point computed in the previous analysis remains unchanged. Possible values are `no`, `screen`, `logfile`, or `rawfile`.

Convergence parameters

27 `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

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

You can specify sweep limits by specifying the end points or the center value and 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 specify a step size parameter (`step`, `lin`, `log`, or `dec`) to determine whether the sweep is linear or logarithmic. If you do not specify 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

© 2003-2014 All Rights Reserved.

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.

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 by 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 required solution. 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 significantly speed up convergence.

When you simulate the same circuit multiple times, it is recommended that you use both `write` and `readns` parameters and assign the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed since the last simulation, and the nodeset file is automatically updated.

During the initial operating point DC analysis, you may force some of the circuit variables to the values given in the `ic` file, `ic` statements, or `ic` parameter on the capacitors and inductors. The `ic` parameter controls the interaction of the various methods for setting the force values. The effects of individual settings are as follows:

`force=none`: All initial conditions are ignored.

`force=node`: The `ic` statements are used, and the `ic` parameters 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 `ic` statements and `ic` parameters are used, with the `ic` parameters overriding the `ic` statements.

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

After you specify the initial conditions, Spectre computes the DC operating point 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`).

© 2003-2014 All Rights Reserved.

Parameter Index

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

© 2003-2014 All Rights Reserved.

4

Other Simulation Topics

This chapter discusses the following topics:

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

Using analogmodel for Model Passing (analogmodel)

Description

`analogmodel`, a reserved word in Spectre, enables you to bind an instance to different masters based on the value of a special instance parameter named `modelname`. An instance of `analogmodel` must have a parameter named `modelname`, whose string value represents 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`	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 be a valid string identifier or a netlist parameter, and must resolve to a valid master name, a primitive, a model, a subckt, or an AHDL module.
`param1 param2...`	Parameter values for the component. Depending on the master type, these can either be device parameters or netlist parameters. It is optional to specify these parameter values.

Example

```
//example spectre netlist to illustrate modelname parameter

simulator lang=spectre
parameters b="bottom"
include "VerilogAStuff.va"

topInst1 (out in) top
topInst2 (out in) analogmodel modelname="VAMaster" //VAMaster is defined in
                                                    "VerilogAStuff.va"
topInst3 (out in) analogmodel modelname="resistor" //topInst3 binds to a primitive
```


© 2003-2014 All Rights Reserved.

```
topInst4 (out 0) analogmodel modelname="myOwnRes" //topInst4 binds to modelcard
                                                    "myOwnRes" defined below
v1 in 0 vsource dc=1

    model myOwnRes resistor r=100
    subckt top out in
    parameters a="mid"
    x1 (out in) analogmodel modelname=a //topInst1.x1 binds to "mid"

ends top

subckt mid out in

    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 low out in

    x1 (out in) analogmodel modelname="bottom" //topInst1.x1.x1.x2.x1 binds to
                                                 "bottom"

ends low

subckt bottom out in

    x1 (out in) analogmodel modelname="resistor" //x1 binds to primitive "resistor"

ends bottom

dc1 dc

//"VerilogAStuff.va"
include "constants.h"
include "discipline.h"
module VAMaster(n1, n2);
inout n1, n2;
electrical n1, n2;
parameter r=1k;
analog begin
I(n1, n2) <+ V(n1, n2)/r;
end
endmodule
```


© 2003-2014 All Rights Reserved.

Behavioral Source Use Model (bsource)

Description

Behavioral source enables you to model a resistor, inductor, capacitor, voltage or current source as a behavioral 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.

`bsource` simulation performance has been improved by compiling the bsource devices. For more information, see the bsource Compilation section.

The syntax for bsource is as follows:

```
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 an Spectre expression, which contains:

- Netlist parameters.
- Current simulation time, `$time`.
- 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.
- Branch currents, `i("inst_id:index")`, where `inst_id` is an instance name given in the netlist and index is the port index that starts from 1. The default value of index is 1.

© 2003-2014 All Rights Reserved.

Note: If the value of the port index is set to 0, simple_expr treats it as the default value 1.

`generic_expr` is defined as a `simple_expr` or `ddt()` or `idt()` of `simple_expr`.

`param_list` is `param_name=value`.

`param_name` can have the multiplicity factor `m`. The value of `m` defaults to 1.

Temperature Parameters

`tc1`	Linear temperature co-efficient. Valid for all behavioral elements. Default value is 0 1/C.
`tc2`	Quadratic temperature co-efficient. Valid for all behavioral elements. Default value is 0 C^-2
`tnom`	Parameters measurement temperature. Valid for all behavioral elements. Default value is 27.0.
`trise`	Temperature rise for ambient. Valid for all behavioral elements. Default value is 0.0.
`tc1c`	Linear temperature coefficient of capacitor. Valid for resistor type behavioral element. Default value is 0 1/C.
`tc2c`	Quadratic temperature coefficient of capacitor. Valid for resistor type behavioral element. Default value is 0 C^-2.

Clipping Parameters

`max_val`	Maximum value of bsource expression. Valid for all behavioral elements, but generally used with i and v elements to clip the current or voltage between the specified values.
`min_val`	Minimum value of bsource expression. Valid for all behavioral elements, but generally used with i and v elements to clip the current or voltage between the specified values.

© 2003-2014 All Rights Reserved.

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. Default value is 0.
`white_noise`	White noise expression. Valid for v and i elements.
`flicker_noise`	Flicker noise expression. Valid for v and i elements.

DC Mismatch Parameters

`mr`	DC-Mismatch parameter. Valid only for r.

For the detailed algorithm, refer to "*Affirma Spectre DC Device Matching Analysis Tutorial*."

All the parameters in the `param_name` table are instance parameters. `white_noise` and `flicker_noise` may be assigned behavioral expressions; the other parameters must be assigned constant or parametric expressions.

Supported Instance Parameters

`bsource` supports the following instance parameters for Spectre primitives:

Resistor	`isnoisy`, `m`, `r`, `tc1`, `tc2`, `trise`, `kf`, `af`, `fexp`, `ldexp`, `wdexp`, `l`, `w`, `mr`
Capacitor	`c`, `m`, `tc1`, `tc2`, `trise`, `ic`
Inductor	`l`, `m`, `tc1`, `tc2`, `trise`

© 2003-2014 All Rights Reserved.

Mathematical Definitions

- `i = ddt(q) = ddt(simple_expr)`
- `v = ddt(phi) = ddt(simple_expr)`
- `v = i * r = i* simple_expr`
- `i = g * v = simple_expr * v`
- `i = c * ddt(v) = simple_expr * ddt(v)`
- `v = l * ddt(i) = simple_expr * ddt(i)`

Operating Point Parameters

Capacitor

`cap (F)`	Capacitance at operating point

Conductor

`g (S)`	Conductance at operating point
`v (V)`	Voltage at operating point
`i (A)`	Current through the conductor
`pwr (W)`	Power dissipation.

Current Source

`v (V)`	Voltage across the source
`i (A)`	Current through the source
`pwr (W)`	Power dissipation

Inductor

`ind (H)`	Inductance at operating point

© 2003-2014 All Rights Reserved.

`i (A)`	Current at operating point

Charge

`cap (F)`	Capacitance at operating point
`ddt_v (V/s)`	Voltage gradient at operating point

Resistor

`v (V)`	Voltage at operating point
`i (A)`	Current through the resistor
`res (Ohm)`	Resistance at operating point
`pwr (W)`	Power dissipation

Voltage Source

`v (V)`	Voltage across the source
`i (A)`	Current through the source
`pwr (W)`	Power dissipation

Temperature Effects on bsource

The equation for computing temperature factor is as follows:

```
tempFactor = [1 + tc1*(temp+trise-tnom)+tc2*(temp+trise-tnom)^2]
```

Examples of bsource Usage

Non-linear resistor/capacitor/inductor modeling

```
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))
```


© 2003-2014 All Rights Reserved.

```
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:1")
```

Current-controlled voltage source

```
vsrc (n1 n2) vsource v=10
ccvs1 (n3 n4) bsource v=100*i("vsrc:1")
```

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 coefficient for resistor

```
res (n1 n2) bsource r=100  tc1=0.01 tc2=0.003 trise=10 tnom=30
```

Giving DC mismatch parameter for resistor

```
res (n1 n2) bsource r=100  mr=0.3
```

bsource support model card

```
model model_card_res resistor tc1=0.1 tc2=0.1
res (n1 n2) model_card_res r=100*(1+(1/2)*v(n1,n2))
```

Doing 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))
ran1 tran stop=1u
```


© 2003-2014 All Rights Reserved.

```
altAnal  {
    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: With standard (simple) syntax for resistor/capacitor/inductor, the `bsource` keyword is not required in the statement. However, if the keyword is specified, Spectre treats the resistor/capacitor/inductor as a behavioral source.

Frequency Effects on bsource

The support for frequency-dependent bsource is available. You can use a frequency-dependent resistor, capacitor, inductor, or bsource component in ac/noise/xf/sp/hb/hbac/hbnoise analyses.

The syntax is as follows:

```
name ( node1 node2 ) model_name behav_param=freq_expression
```

where:

- `name` is the name of the instance
- `model_name` can be bsource, resistor, capacitor, and inductor
- `behav_param` is the r, c or l parameter.
- `freq_expression` can be specified by arbitrary expressions using the `$freq` keyword.

Following are some examples.

Example-1

```
res (n1 n2) bsource r= 100 + sqrt($freq)/1e6
res (n1 n2) resistor  r= 100 + sqrt($freq)/1e6
cap (n1 n2) bsource c=1e-12/(1+($freq)/1e9)
cap (n1 n2) capacitor c=1e-12/(1+($freq)/1e9)
ind (n1 n2) bsource l=1e-9+1e-9/(1+($freq)/1e9)
ind (n1 n2) inductor l=1e-9+1e-9/(1+($freq)/1e9)
```

Example-2 Frequency-dependent resistor component

```
R1  (1  0) resistor r= 100 + sqrt($freq)/1e6
//also can use: R1  (1  0) bsource r = 100 + sqrt($freq)/1e6
V1 (1 0 ) vsource type=sine freq=1G fundname="freq"
```


© 2003-2014 All Rights Reserved.

```
hb hb tstab=0n funds=["freq"] maxharms=[5] errpreset=conservative
hbnoise ( net1 ) hbnoise start=1 stop=100M dec=3
```

In this example, the resistance of R1 is frequency-dependent. Therefore, if we run hb+hbnoise analyses for different frequencies, the resistance varies accordingly.

Example-3 Frequency-dependent capacitor component

```
C1  (1  0) capacitor c=1e-12/(1+($freq)/1e9)
// also can use C1  (1  0) bsource c=1e-12/(1+($freq)/1e9)
V1 (1 0 ) vsource type=sine freq=1G  fundname="freq"
hb hb tstab=0n funds=["freq"] maxharms=[5] errpreset=conservative
hbac hbac start=1 stop=1G dec=3
```

In this example, the capacitance of C1 is frequency-dependent. Therefore, if we run hb+hbac analyses for different frequencies, the capacitance varies accordingly.

Example-4 Frequency-dependent inductor component

```
L1  (1  0) inductor l=1e-9+1e-9/(1+($freq)/1e9)
// also can use L1  (1  0) bsource l=1e-9+1e-9/(1+($freq)/1e9)
V1 (1 0 ) vsource type=sine freq=1G  fundname="freq"
hb hb tstab=0n funds=["freq"] maxharms=[5] errpreset=conservative
```

In this example, the inductance of L1 is frequency-dependent. Therefore, if we run hb analysis for different frequencies, the inductance varies accordingly.

bsource Compilation

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, re-compilation is performed only 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 might result in performance slowdown. To turn off compilation, set the `CDS_AHDLCMI_ENABLE` shell environment variable to `NO`, as follows:

```
setenv CDS_AHDLCMI_ENABLE NO
```

To re-enable bsource compilation, set the `CDS_AHDLCMI_ENABLE` to `YES`, as follows:

```
setenv CDS_AHDLCMI_ENABLE YES
```

To re-enable bsource compilation, you can also undefine the `CDS_AHDLCMI_ENABLE` environment variable, as follows:

© 2003-2014 All Rights Reserved.

```
unsetenv CDS_AHDLCMI_ENABLE
```


© 2003-2014 All Rights Reserved.

Checkpoint - Restart (checkpoint)

Description

Spectre has the ability to save the checkpoint files generated during analyses, and to restart an analysis from its checkpoint file. Checkpoint files can be generated in the following ways:

- Periodically, based on real time (wall clock time).
- Asynchronous UNIX signals.
- 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. This option is turned on by default with a value of 1800 seconds (30 minutes). Spectre deletes the checkpoint file if the simulation completes normally. If the simulation terminates abnormally, the checkpoint file is not deleted.

If Spectre receives the UNIX signal `USR2`, Spectre immediately writes a checkpoint file. If Spectre receives interrupt signals, such as `QUIT`, `TERM`, `INT`, or `HUP`, Spectre attempts to write a checkpoint file and then exits. For other fatal signals, Spectre may not write a checkpoint file.

The name of the checkpoint file is a combination of the circuit name and the analysis name with the extension `.ckpt`. For example, if the circuit is named `test1` and the transient analysis is named `timeSweep`, the checkpoint file is named `test1.timeSweep.tran.ckpt`.

Spectre keeps only the latest checkpoint file. It creates a new checkpoint file with a temporary name. After the file is successfully written, Spectre deletes the previous checkpoint file created earlier and renames the new file.

Currently, only transient analysis supports checkpoint files and restart.

Checkpoint

Transient analysis can generate checkpoint files periodically based on the transient simulation time. This is done by using a transient analysis parameter named `ckptperiod`, which is turned off by default. To enable the checkpoint feature, the argument `+checkpoint` must be added to the `spectre` command.

© 2003-2014 All Rights Reserved.

Restart

To restart an analysis from a checkpoint file, use the `+recover` option with the `spectre` command. Spectre searches the analyses log for the checkpoint file. If the checkpoint file for the analysis exists, Spectre skips any previous analyses, and restarts the analysis by using the information from the file.

© 2003-2014 All Rights Reserved.

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 as follows:

1. Spectre first reads the default CMI configuration file that specifies the default shared objects provided by Cadence.
2. The configuration file specified by the value of the `CMI_CONFIG` environment variable is then read.
3. The third configuration file that Spectre reads is `~/.cmiconfig`.
4. Finally, the configuration file specified in the `-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`: Adds a shared object to the list of shared objects to load

```
loads [path/]<shared_object_name>
```

`unload`: Removes a shared object from 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
```


© 2003-2014 All Rights Reserved.

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.

© 2003-2014 All Rights Reserved.

Built-in Mathematical and Physical Constants (constants)

Description

Spectre supports the following list of built-in mathematical and physical constants:

Note: `M_` is a mathematical constant

`M_E`	`2.7182818284590452354`	`exp(1) = e`
`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

Note: `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`	Boltzmans constant in joules/kelvin

© 2003-2014 All Rights Reserved.

P_H	6.6260755e-34	Plancks constant in joules*sec
P_EPS0	8.854187923944200139 68e-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 where a numeric value of the expression is expected.

© 2003-2014 All Rights Reserved.

Convergence Difficulties (convergence)

Description

If you are having convergence difficulties, try the following suggestions:

1. Evaluate and resolve any notice, warning, or error messages.
2. Ensure that the topology checker is being used (set topcheck=full on options statement) and heed any warnings it generates.
3. Perform sanity check on the parameter values by 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 by using `info` analysis. Ensure 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. Instead, use voltage sources or iprobes to measure current.
5. Use realistic device models. Check all component parameters, particularly nonlinear device model parameters, to ensure 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 to avoid regions that might contribute to convergence problems in the model.

DC Convergence Suggestions

After you have a solution, write it to a nodeset file by using the `write` parameter, and read it back in on subsequent simulations by 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 and the solution from the previous analysis is far from the solution for this analysis, set `restart=yes`.

© 2003-2014 All Rights Reserved.

4. If simulating a bipolar analog circuit, ensure that the region parameter on all transistors and diodes is set correctly.
5. If the analysis fails at an extreme temperature, but succeeds at room temperature, 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 a value in the range of 0.1 to 0.5 by using `pivrel` (on options statement).
7. Divide the circuit into smaller pieces and simulate them individually. However, ensure that the results are close to what they would be if you had simulated the whole circuit. 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 transient analysis well beyond the time when all the sources have reached their final value (remember that transient analysis is very cheap when none of the signals in the circuit are changing) and write the final point to a `nodeset` file. To make 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 fails, or is very slow because the circuit contains an oscillator. Often, for finding the dc solution, the oscillation can be eliminated by setting the minimum capacitance from each node to ground (`cmin`) to a large value.

Transient Convergence Suggestions

1. Ensure 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 jump in the solution.

© 2003-2014 All Rights Reserved.

encryption (encryption)

Description

Encryption enables you to protect your proprietary parameters, subcircuits, models, netlists, and release your libraries to your customers without revealing sensitive information.

1. Define Encryption blocks in the netlist

   Keywords (`.protect`, `.unprotect`) are used for defining an encryption block.`(protect, unprotect)` is the accepted syntax in native Spectre mode.The dot keywords are used in the context of the spice mode. (`.protect`, `.unprotect`)/ (`protect`, `unprotect`) can be abbreviated to (`.prot`, `.unprot`)/(`prot. unprot`), respectively.

   If the whole file needs to be encrypted, one method is to put `protect` at the beginning of the file and `unprotect` at the end of the file. Alternatively,you can use the `-all` option of the spectre encrypt command.

   Examples of netlist with protected blocks:

   Example: Protection of subckts

   ```
   .protect
   .subckt sub1 ( ... )
   ....
   .ends sub1
   .subckt sub2 ( ... )
   .....
   .ends sub2
   .unprotect
   ```

2. Encrypt the netlist

   `spectre_encrypt` is a standalone encryptor, and is used as follows:

   ```
   spectre_encrypt [-i input_file] [-o output_file] [-all]
   ```

   where

   `[-i Input_file]`: Netlist to be encrypted

   `[-o output _file]`: Output file of the encrypted netlist

   `[-all]`: The whole file will be encrypted in the input netlist

   Note: The include files or the library files in the netlist need to be encrypted separately. Spectre encryptor does not encrypt the included files automatically.

© 2003-2014 All Rights Reserved.

3. Simulate the encrypted netlist

There is no difference in how you run Spectre on an encrypted or unencrypted netlist. Spectre automatically decrypts and encrypted netlist.

For encrypted netlists, Spectre turns on the protection for devices, models, signals, and parameters in the encrypted blocks. Any error or warning messages and the outputs from the protected information is suppressed or filtered out.

The following list describes how protection is implemented:

- Circuit inventory does not include encrypted parts.
- The `info` command suppresses all information on encrypted parts.
- Errors on encrypted parts are reported generally as:

  ```
  Error has occurred within the encrypted block (no details are given).
  ```

- If a portion of the model is encrypted, the entire model is encrypted.
- Any command that references the protected elements results in an error message reporting that those elements do not exist.
- Protected device and model parameters cannot be altered directly through alter. However, if they depend on other alterable parameters, protected parameters are recalculated. Protected devices and models can be replaced.
- Protected nodes are output in an encrypted format when the `ic` file is requested (similarly, for nodes in checking point and restart).
- The encryption feature is not available in models using CMI 2.0.

4. Output operating points on protected devices

By default, all information about the protected devices is suppressed and is not visible. However, IP providers have the control to expose the operating points of the protected devices to the end users for back annotation.

Keywords (`visible`, `invisible`) or (`.visible`, `.invisible`) in the spice netlist content are defined to expose the operating points of encrypted devices.

The operating point of the protected devices between visible and invisible is not suppressed by adding `what=oppoints` to the `visible` statement.

For example:

```
prot
x1 n1 n2 n3 n4  nmos
visible what=oppoints
x2  n5 n6 n7 n8  nmos
```


© 2003-2014 All Rights Reserved.

```
x3  n9 n10 n10 n8 pmos
invisible
X4   n11 n12 n13 n13 pmos
unprot
```


© 2003-2014 All Rights Reserved.

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 the top-level netlist parameters or subcircuit parameters. Calls to algebraic and trigonometric functions are also supported. The supported operators, algebraic, and trigonometric functions are listed after the 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
```

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

© 2003-2014 All Rights Reserved.

model parameter, bf), or when specifying initial conditions and nodesets for individual circuit nodes.

Operators

The following operators are supported, listed in the order of decreasing precedence. Parentheses can be used to change the order of evaluation. For a binary expression, such as `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 raised to the power of second operand
Multiply, Divide	`*, /`	Sum, Difference of operands
Binary Plus/Minus	`+, -`	Sum, Difference of operands
Shift	`<<, >>`	First operand shifted left or right by the number of bits specified by the 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 are true.
Logical OR	`||`	True if either operand is true
Ternary Operator	`(cond) ? x : y`	Returns x if `cond` is `true`, and y if `cond` is false, where x and y are expressions

© 2003-2014 All Rights Reserved.

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
`int(x)`	integer value of x	All x
`ifloor(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
`sgn(x)`	The sign of x	All x
`sign(x,y)`	sgn(y)*fabs(x)	All x, all y
`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

© 2003-2014 All Rights Reserved.

Function	Description	Domain
`cosh(x)`	Hyperbolic cosine	All x
`tanh(x)`	Hyperbolic tangent	All x
`asinh(x)`	Arc-hyberbolic 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 a 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.

© 2003-2014 All Rights Reserved.

The fastdc command line option (fastdc)

Description

The `fastdc` option enables you to speed up the DC simulation in Spectre and APS for large scale circuits and for cases where DC convergence is very slow or there is a difficulty in DC convergence.

The fastdc option provides a method to quickly obtain the approximate DC solution to start transient analysis.

Note: The DC solution obtained using the `fastdc` command-line option may not be as accurate as the true DC solution. Therefore, it should not be used in applications which require accurate DC solutions.

Definition

```
+fastdc in the command line
```


© 2003-2014 All Rights Reserved.

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 are taken as real values, and the functions return real values. A user-defined function may contain only a single statement in braces, and this statement must return an expression (which is typically an expression involving the function arguments). The return expression may reference the built-in parameters `temp` and `tnom`. User-defined functions must be declared only at the top level, and must not be declared within subcircuits. User-defined functions may be called from anywhere an expression can be currently used in Spectre. User-defined functions may call other functions (both user-defined and built-in), however, any user-defined function needs 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.

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 is as follows:

```
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:

```
r1 (1 0) resistor r=myfunc(2.0, 4.5)
```


© 2003-2014 All Rights Reserved.

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 subcircuit 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

```
global <ground> <node> ...
```


© 2003-2014 All Rights Reserved.

IBIS Component Use Model (ibis)

Description

IBIS (I/O Buffer Information Specification) is a standard for electronic behavioral specification of integrated circuit (IC) input/output analog characteristics. It allows you to define a model for the IC component package, and a buffer model for each pin. IBIS standard also allows you to describe a board-level component containing several components on a common substrate or printed circuit board (PCB). For example, a `SIMM` module is a board-level component that is used to attach several DRAM components on the PCB to a motherboard through edge connector pins. Board pins, components on the board, and connections between them are defined in an Electrical Board Description file with extension `.ebd`. Component pins, buffers, and package descriptions are in a separate IBIS file with extension `.ibs`. An Additional Package file with extension `.pkg` can be used to describe advanced package models.

```
<SPECTRE NETLIST SUPPORT>
```

IBIS files can be referenced in Spectre netlist by using the `ibis_include` statement:

```
ibis_include "DRAM.ibs" [options]
```

IBIS file "DRAM.ibs" is translated into Spectre netlist format by using `ibis2subckt` utility. The output file, "DRAM.scs", containing subcircuit definitions for each IBIS component, board, and package found in the input IBIS file, is included in the netlist. The list of options may consist of: `corner={typ|min|max|slow|fast}`, `swsel=<int>` and `mdsel=<int>`. These options are transferred to `ibis2subckt`. They are used to select IBIS buffer model corner, change position of series-switch models, and choose the required models from model selector list.

IBIS component and board subcircuits can be instantiated in a Spectre netlist along with regular Spectre primitives. Subcircuit name is the same as the component name, but is appended with the suffix_`ibis`. Subcircuit terminals are component pins, arranged in the order they are listed in the IBIS file. Each pin terminal is followed by a number of signal terminals, depending on the type of the pin buffer model. For example, if m1 is defined in IBIS file as an input buffer model, and m2 - as I/O type, the following IBIS component:

```
[Component] IC
[Pin]  signal_name   model_name   R_pin     L_pin     C_pin
p1      s1            GND
p3      s3            m1
p4      s4            NC
p5      s5            m2
p2      s2            POWER
```

can be instantiated in the netlist as:

© 2003-2014 All Rights Reserved.

```
x_ic ( p1 p3 s3_in p4 p5 s5_in s5_out s5_en p2 ) IC_ibis
```

`ibis2subckt` can also be used as a stand-alone utility with the following command-line arguments:

```
ibis2subckt  -in <IBIS files> -out <subckt file> -corner {typ|min|max|slow|fast} -
swsel <int> -mdsel <int>
```

Default values are:

```
corner  typ
mdsel   -1
swsel   -1
```

<SPICE NETLIST SUPPORT>

For the IBIS component, Spectre also supports the `SPICE .IBIS` and `.EBD` statements.

The supported parameters of `.IBIS` syntax are `file`, `component`, `mod_sel`, `package`, and `typ`.

.IBIS Parameters

1 `file = <string>` Specifies the IBIS file name with suffix `.ibs`.

2 `component = <string>`
Specifies the used component name in the `.ibs` file.

3 `mod_sel = <string1=string2>`
Maps the model selector name (string1) to the actual model name (string2), given as [Model selector] in the `.ibis` file. Multiple selectors are supported.

4 `package = <3|0|1|2>`
Specifies the type of package. Default value is 3 (use the best available package model). `package=0` means no package is used. `package=1` means that an RLC package is used with the same values for all pins provided in the `[Package]` section. `package=2` means that an RLC package is used with individual RLC values for each pin provided in the `[Pin]` section. package=3 means that an advanced package model is used in the `[Package Model]` section.

5 `typ = <typ|min|max|fast|slow>`
Specifies the corner of the IBIS buffer. Default value is `typ` (typical).

© 2003-2014 All Rights Reserved.

The supported parameters of the `.EBD` syntax are `file`, `model`, and `component`.

.EBD Parameters

1 `file = <string>` Specifies the EBD file name with the suffix `.ebd`

2 `model = <string>` Specifies name of the board-level model provided in `.ebd` file

3 `component = <string>`
Specifies the component name of the ibis buffer. Multiple components are supported

Examples

```
.ibis I1
+ file = file.ibs
+ component = Component
+ mod_sel = DQ=DQ1,CQ=CQ1
+ package=0
+ typ=slow

.ebd pkg
+ file = file.ebd
+ model = XXXX
+ component = Component1
+ component = Component2
```


© 2003-2014 All Rights Reserved.

Initial Conditions (ic)

Description

The `ic` statement is used to provide initial conditions for nodes in transient analysis. It can occur multiple times in the input, and the information provided in all the occurrences is collected. Initial conditions are accepted only 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. Note that specifying `cmin` for a transient analysis, does not satisfy the condition that a node has a capacitive path to ground.

Definition

```
ic <X[:param]=value> ...
```

This statement takes a list of signals with the state information to the DC and transient analysis. `X` can be a node, a component, or a subcircuit and `param` can either be a component output parameter or a terminal index. To specify a class of signals, use the pattern matching characters `*` for any string and `?` for any character.

The concept of nodes for the statement 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 an inductor or the voltage of the internal node in a diode. Topological nodes can either be 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.

© 2003-2014 All Rights Reserved.

The Structural if-statement (if)

Description

The structural if-statement can be used to conditionally instantiate other instance statements.

Definition

```
if <condition> <statement1> [ else <statement2> ]
```

The condition is a Boolean expression based on the comparisons of various arithmetic expressions that 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 conditions are as follows:

- References to an instance with duplicate names is possible only within a structural if statement that has both an "if" part and an "else" part.
- Both the "if" part and the "else" part must be 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 chooses 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
```


© 2003-2014 All Rights Reserved.

```
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
```


© 2003-2014 All Rights Reserved.

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, the path is taken relative to the directory of the file currently being read.

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 the Spectre `include` statement, and the CPP `#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 Spectre `include` statement in that the CPP macro processing is not performed on files included by Spectre, but is performed on files included by CPP. If your netlist contains a `#include` statement, you must run CPP to perform this inclusion; otherwise, an error occurs.

If the file to be included cannot be found in the same directory as the including file, both the Spectre `include` and CPP `#include` search for the file to be included along the search path specified by the `-I` command-line arguments.

The Spectre `include` statement allows you to include a library section. The following is the syntax for specifying a library reference:

```
include "file" section=sectionName
```

where, `file` is the name of the library file to be included, and `sectionName` matches the name of the section defined in the library. The library reference statement looks like an `include` statement, except for the specification of the library section. When the file is inserted, only the named section is actually included.

The Spectre `include` statement allows you to embed special characters in the name of the file to be included. It automatically expands the ~ character to the user's home directory. The Spectre `include` statement also expands the environment variables and % codes, such as

```
include "~/models/${SIMULATOR}_pd/npn.scs"
```


© 2003-2014 All Rights Reserved.

which look in the directory given by the environment variable `SIMULATOR`, followed by `_pd`, which is under the `models` directory in the user's home directory.

Note: These special character features are not available with the CPP `#include` statement.

Definition

```
include "filename"
```


© 2003-2014 All Rights Reserved.

Spectre Netlist Keywords (keywords)

Description

The following are Spectre keywords, including netlist keywords and built-in mathematical and physical constants. Spectre has special use for these keywords, and they are reserved in certain contexts. You should follow the rules given below when using them to prevent errors.

- Netlists keywords cannot be used as instance names, subckt names, model names, or function names.
- Built-in mathematical and physical constants cannot be used as node names, instance names, subckt names, model names, function names, or parameter names.

Keyword	Keyword Type
M_1_PI	Mathematical Constant
M_2_PI	Mathematical Constant
M_2_SQRTPI	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

© 2003-2014 All Rights Reserved.

`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
	Netlist Keyword
`correlate`	Netlist Keyword
`else`	Netlist Keyword
`end`	Netlist Keyword
`ends`	Netlist Keyword
`export`	Netlist Keyword
`for`	Netlist Keyword
`function`	Netlist Keyword
`global`	Netlist Keyword
`ic`	Netlist Keyword
`if`	Netlist Keyword
`inline`	Netlist Keyword
`invisible`	Netlist Keyword
`library`	Netlist Keyword

© 2003-2014 All Rights Reserved.

`local`	Netlist Keyword
`march`	Netlist Keyword
`model`	Netlist Keyword
`nodeset`	Netlist Keyword
`parameters`	Netlist Keyword
`paramset`	Netlist Keyword
`plot`	Netlist Keyword
`print`	Netlist Keyword
`protect`	Netlist Keyword (The first four letters of the keyword (in lowercase or uppercase) can be used as an abbreviation. For example, prot and PROT are valid keywords).
`pwr`	Netlist Keyword
`real`	Netlist Keyword
`return`	Netlist Keyword
`save`	Netlist Keyword
`sens`	Netlist Keyword
`statistics`	Netlist Keyword
`subckt`	Netlist Keyword
`to`	Netlist Keyword
`truncate`	Netlist Keyword
`unprotect`	Netlist Keyword. (The first six letters of the keyword (in lowercase or uppercase) can be used as an abbreviation. For example, unprot and UNPROT are valid keywords).
`vary`	Netlist Keyword

© 2003-2014 All Rights Reserved.

`visible` Netlist Keyword

© 2003-2014 All Rights Reserved.

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 specified 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, the path is taken relative to the directory of the file currently being read.

There are two types of library statements. One that references a library section, and another that defines a library section. The definition of a library section is prohibited in the netlist.

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).

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 kpnp 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
```


© 2003-2014 All Rights Reserved.

```
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
   + 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 kpnp bjt type=pnp is=10e-13 bf=30 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 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 kpnp 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
endlibrary
```


© 2003-2014 All Rights Reserved.

Tips for Reducing Memory Usage (memory)

Description

If you are facing an insufficient memory problem, try the following suggestions:

1. Try using a 64-bit executable if you are using the 32-bit one.

2. Try `ulimit`/`unlimit` command to adjust memory limitations.

3. Try another machine that has more memory, if hardware limit is the cause

4. Refer to `spectre -h rfmemory`, if you faced the problem during RF analyses.

© 2003-2014 All Rights Reserved.

Node Sets (nodeset)

Description

The `nodeset` statement is used to provide an initial guess for nodes in DC analysis or to provide the initial condition calculation for transient analysis. The `nodeset` statement can occur multiple times in the input and the information provided in all the occurrences is collected. For more information, read the description of DC analysis.

Definition

`nodeset <X>[:param]=value`

This statement takes a list of signals with the state information to the DC and transient analyses. `X` can be a node, a component, or a subcircuit, and `param` can be either a component output parameter or a terminal index. To specify a class of signals, use the pattern matching character `*` for any string and `?` for any character.

The concept of nodes for the statement 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 an 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.

© 2003-2014 All Rights Reserved.

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, 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 rb 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 to provide limits on parameters that do not have built-in limits. To do so, create a parameter range limits file and run Spectre by providing the name of the file after the `+param` command-line option. For example:

```
spectre +param limits-file input-file
```

Limits are specified 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 specified more than once, the limits specified at last override the earlier limits. The primitive name must be a Spectre primitive name, and not a name used for SPICE compatibility. For example, `mos3` must be used instead of `mos`. Parameter limits can be written using Spectre native mode metric scale factors. Therefore, a limit of `f <= 1.0e6` can also be written as `f <= 1M`.

Examples

```
mos3       0.5u <= l <= 100u
    0.5u <= w
    0 < as <= 1e-8
    0 < ad <= 1e-8
model |vto| <= 3
```


© 2003-2014 All Rights Reserved.

Note that it is not necessary to specify the primitive name each time. If the primitive name is not specified, it is assumed to be the same as the previous parameter. Upper and lower limits may be specified, but if these are not specified, there is no limit on the parameter value. Therefore, in the example, if w is less than 0.5um, a warning is issued, but there is no limit on how large `w` can be. If a parameter is mentioned, but no limits are given, 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 <= x < 0.5
1 <= y <= 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, including 0.5. The limits are applied separately, therefore, x must be both greater than or equal to 1 (1 <= 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 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`, and `sat` (see `spectre -help bjt`). Each enumeration is assigned a number starting at 0 and counting up. Therefore, `off=0`, `fwd=1`, `rev=2`, and `sat=3`. The specification `bjt 3 <= region <= 1`indicates that a warning should be printed if region=rev because the conditions (`3 <= region`) and (`region <= 1`) exclude only (`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"
```

temporarily suspends 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, and not from the directory from which Spectre was run.

© 2003-2014 All Rights Reserved.

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 you shell initialization file (`.profile` for sh, `.cshrc` for csh)

```
setenv SPECTRE_DEFAULTS "+param /cds/etc/spectre/param.lmts"
```

Spectre always reads the specified limits file.

© 2003-2014 All Rights Reserved.

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>[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.

© 2003-2014 All Rights Reserved.

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. In addition, 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, 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. 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 view the details for any of these analyses, where `<analysis>` is the analysis of interest.

© 2003-2014 All Rights Reserved.

Temperature as a Parameter

You can use the reserved parameters `temp` and `tnom` anywhere 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 everytime the circuit temperature or measurement temperature changes.

You can also alter or sweep the `temp` and `tnom` parameters by using any of the techniques available for altering or sweeping the netlist or subcircuit parameters (with the exception of s).

This capability allows you to write temperature dependent models, for example, by using `temp` in an equation for a model or an 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 must not be declared as either top-level parameters or subcircuit parameters: `temp`, `tnom`, `scale`, `scalem`, `freq`, `time`.

© 2003-2014 All Rights Reserved.

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 that 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> paramset {
}
```

Example:

```
data paramset {
   p1  p2  p3
   1.1 2.2 3.3
   4.4 5.5 6.6
}
```


© 2003-2014 All Rights Reserved.

Pspice_include File (pspice_include)

Description

Spectre supports PSPICE netlist format targeting to include PCB components that are modeled in PSPICE format. This solution does not support PSPICE designs. A top-level netlist and control statement needs to be defined in Spectre or SPICE format. The recommended approach is to define a subckt in PSPICE netlist format and to instantiate the subckt in a Spectre netlist.

A PSPICE netlist can be included in Spectre by using the following include statements.

```
pspice_include <file> (Spectre format)
.pspice_include <file> (SPICE format)
```

PSPICE 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. If the name given is not an absolute path specification, it is taken relative to the directory of the file currently being read.

The PSPICE file defined in the pspice_include statement is required to contain only PSPICE netlist format. A PSPICE file may also contain include statements such as .inc or .lib to include other PSPICE files. The pspice_include statement cannot be used in a PSPICE netlist.

When Spectre-simulator opens a pspice_include file then it automatically switches to PSPICE mode. When in PSPICE mode, all elements and device models used in the PSPICE netlist are simulated using PSPICE default values and equations. Switching from PSPICE mode to Spectre mode inside a PSPICE file is not supported. 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.

Definition

```
Pspice_include "filename"
```


© 2003-2014 All Rights Reserved.

Tips for Reducing Memory Usage with SpectreRF (rfmemory)

Description

Problem: How can you reduce memory usage when running SpectreRF simulations? What are the things that you need to be aware of?

Solution: The amount of swap/memory that Spectre/SpectreRF requires depends on the following:

- The analysis type you are running (PSS, QPSS, transient, dc, and so on). PSS and QPSS can take up a lot of memory.
- The simulator options. `Reltol`, `Vabstol`, `maxstep`, and `relref` are the primary options that affect swap/memory.
- The complexity of models being used.
- The size of your circuit. For example, the number of active devices.
- Substrate effects. If you are using unreduced substrate networks, the memory requirements become **very** high.
- The number of time points taken and the number of harmonics requested in PSS analysis. If the number of harmonics becomes larger than 10, the number of timepoints taken by the simulator increases. However, this effect is not large when compared to an HB type simulator.
- The number of nodes/nets you are saving. This has a much smaller effect on memory compared to the amount of memory used in PSS/QPSS. It can change the amount of disk space used for the solution only.
- For QPSS, the harmonic numbers for the moderate tones are input parameters. The harmonic numbers control the number of integration intervals QPSS takes, and therefore, dominates the memory usage of QPSS.

You can consider using the following workarounds:

1. Check to see how much swap and memory you have on your computer. Greater than 2GB RAM and 5GB swap is recommended.

   A 64-bit Spectre executable is available with MMSIM releases. This means that there is no 4GB process size limit, which was the case for the 32-bit Spectre executable.

© 2003-2014 All Rights Reserved.

However, if the circuit is large, the RAM requirements could be significantly high, especially while running simulation after performing parasitic extraction.

2. Reduce the number of data points stored by reducing the number of harmonics saved. Specify only the individual harmonics that you want to view. You can do this by selecting *Array of Coefficients* or *Array of Indices* from the *Output Harmonics* section of the *Choosing Analyses* form.

   For harmonics, after you exceed 10, the number of timepoints goes up by forcing a smaller maxstep. Because all solution matrices at each timepoint need to be saved and also need to be accessed at the end of each PSS iteration, the memory requirements can get quite large. `maxacfreq` also affects `maxstep`. Higher `maxacfreq` (above 40 * PSS beat freq) causes more timepoints, and therefore, requires more memory. If you use `maxacfreq`, set it to the highest frequency in your circuit. If running Pnoise analysis, choose all the sidebands you can, focusing on the ones that will give you most noise.

3. Use the swapfile option to PSS/QPSS. The swapfile parameter of PSS/QPSS analyses is used to direct the simulator to use a conventional file, rather than virtual memory, to hold the periodically varying small-signal representation of the circuit. The data that the simulator saves to the disk file is the entire series of solution matrices from the PSS analysis. Spectre wires the data in concentric cylinders so that disk access time is as short as possible. If you have access to a computer with sufficient RAM, it is recommended to use a 64-bit Spectre executable.

   a. If running SpectreRF in Artist, the swapfile option is located in the PSS/QPSS Choosing Analyses Options form. Access the swapfile option and enter "<path_to>/some_file_name".

   b. If running standalone SpectreRF, on the pss analysis line add swapfile="<path_to>/some_file_name".

      In both cases, make sure that there is enough disk space in the <path_to> directory.

      Note: The swapfile is used only during PSS/QPSS iterations and for small signal PXX/QPXX analyses. It is not used during the Fourier Integral calculation of the harmonics you requested. UNIX swap is used when physical memory is insufficient; hence, the advice on requesting only the PSS harmonic information that you really need (in item 2 above).

      Your Spectre process may run at 99% CPU utilization until the physical memory runs out. It then uses UNIX swap and the process runs at 5% of CPU because of the slow swapping to disk. Using a swapfile can mean that the process runs at 50% because larger sections of the disk are being swapped, typically 3x (and up to 10x) faster.

© 2003-2014 All Rights Reserved.

4. Reduce the number of nodes or nets that you are saving (do not use save all voltages/ currents). However, if you are running PSS analysis and then trying to plot the frequency domain current (or power) spectrum, you must set this in the netlist: `currents=yes`

Never set useprobes=yes for SpectreRF simulations. It can cause large errors in the small signal Pxx analyses. Use useprobes for linear AC analysis only.

5. Choose the largest possible PSSfund frequency. A higher fundamental frequency (PSSfund) yields a shorter simulation time.

6. The ratio of the highest frequency to the lowest frequency in your circuit is also a concern. If you have too many cycles of the highest input frequency, PSS requires too much disk space and too much time to execute. Remember that PSS analysis saves all matrices from each time point, where data is saved for each iteration of the shooting interval, starting from the second iteration. All the data is kept when the initial state equals the final state (within the tolerance parameters). The more time points you require, the more disk space you use. Here is a list of guidelines:

   - For greater than 30 periods of the largest amplitude tone, use QPSS followed by QPxx small signal analyses.
   - For 10-30 periods of the largest amplitude tone, use PSS or QPSS. The number of harmonics, moderate tones, and other options that you specify depend on your circuit.
   - For less than 10 periods of the largest amplitude tone, use PSS followed by Pxx small signal analyses.

7. To improve QPSS convergence, follow the guidelines in the order presented (that is, if A does not work, try B.):

   - A: Raise harmonics on the moderate tones to 5, and set stabcycles=10
   - B: Raise harmonics on the moderate tones to 9, and set stabcycles=25
   - C: Sweep the input power. Use smaller spacings in the power sweep as the power level gets high.

     For more information about `stabcycles`, see solution 11159516.

8. The Spectre frontend parser, `csfe`, is used in MMSIM releases by default. Using this parser decreases memory usage and helps solve SpectreRF memory usage issues.

9. Break the circuit into smaller chunks and simulate the chunks individually. You may also model more complex parts of the circuit with Verilog-A modules.

© 2003-2014 All Rights Reserved.

10. Run Spectre when no other users are using the same machine.

© 2003-2014 All Rights Reserved.

Output Selections (save)

Description

The `save` statement indicates that the values of specific nodes or signals must be saved in the output file. It works in conjunction with the `save` parameter for most analyses. The output file is written in Cadence Waveform Storage Format (WSF), Cadence Parameter Storage Format (PSF), or in Nutmeg/SPICE3 format, which is controlled by a command-line argument or a global option (see the options statement). An appropriate postprocessor should be used to view the output, generate plots, or do any further processing.

Definition

```
save X[:param] ... [depth=num] [sigtype=node|dev|subckt|all]
[devtype=component_type] [subckt=subckt_master]
[exclude=[wildcard_patterns_list]] [compression=no|yes] [ports=yes|no]
[filter=none|rc]
```

The `save` statement takes a list of signals followed by some optional parameters as an argument. `X` can be a node, a component, or a subcircuit and `param` can be a component output parameter or a terminal index. To specify a class of signals, use the pattern matching character `*` for any string and `?` for any character. The parameters control pattern matching. `depth` controls the depth of pattern matching and, by default, matches signals at all hierarchical levels. `sigtype` with the default value `node` defines the type of `X`. If `sigtype=all` `X` can be a node, a component, or a subcircuit. `devtype` defines the component type and has no default value. `subckt` is used to save signals that are contained only in instances of a given subcircuit master. The signals matching a pattern from the list specified with `exclude` are not saved. When `subckt` is given, the wildcard patterns in the save statement and the depth of pattern matching must be relative to the subcircuit master.

The concept of nodes for the `save` statement is 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 an inductor or the voltage of the internal node in a diode. Topological nodes can be at the top level or in the subcircuit.

The nodes in the `save` statement are not compressed by default. When simulation results need compression and also need accuracy result on some nodes, the `save` statement can be used. However, if wildcards are used in the `save` statement and you want to compress those signals, use `compression=yes` to compress the signals.

The ports parameter saves the subckt ports names as well if `yes` is specified. The default value is `no`. The filter parameter works with wildcarding to filter out the nodes that connect only to parasitics, if `rc` is specified. The default value is `none`.

© 2003-2014 All Rights Reserved.

For example:

```
save 7 out OpAmp1.comp M1:currents D3:oppoint L1:1 R4:pwr
```

specifies 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 by the `save` parameter for the analysis.

To specify a component terminal current, specify 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, 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.

Examples of pattern matching

- `save x*.*1 depth=3`

  Saves the voltages of all nodes from level 2 to level 3 whose name starts with `x` and ends in `1`. For example, `x1.n1`, `x1.x2.x3` but not `x1.x2.x3.x4`.

- `save x*.*1 sigtype=subckt`

  Saves all terminal currents of subcircuits from level 2 and above whose name starts with `x` and ends in `1`. For example, `x1.x21:2, x1.x2.x31:3`.

- `save *:c devtype=bjt`

  Saves all collector currents

- `save * subckt=inv`

  Saves the voltages of all nodes in the instances of the subcircuit `inv`. For example, `X1.n1` for an instance `X1` of `inv` but not `net091` at the top-level

- `save * exclude=[X1* X2*]`

  Saves the voltages of all nodes excluding the ones whose names start with `X1` or `X2`, eg `net091`, `X0.res3.n2` but not `X21.res3.n2`.

© 2003-2014 All Rights Reserved.

Savestate - Recover (savestate)

Description

Savestate-Recover is a transient analysis feature. It is a replacement for the current Checkpoint-Restart capability.

The current Checkpoint-Restart capability saves only the circuit solution for the timepoint at which the simulation is interrupted. Because there is no history information saved for the circuit, glitches, convergence issues, and inaccuracies can result when the simulation is resumed. The new Savestate-Recover feature saves the complete state of the circuit, avoiding these issues.

Savestate-Recover provides the following functions:

- You have the option to save circuit information at set intervals or at multiple points during transient analysis. If the simulation halts unexpectedly, you can restart transient analysis from any saved timepoint.
- You can experiment with different accuracy settings over different transient time periods to obtain the optimal speed/accuracy trade-off.

Requirements for Savestate-Recover

When the simulation is restarted:

- Netlist topology must not be changed. Topology changes or the removal/addition of nodes in the restore file causes a fatal error.
- You may edit any netlist parameter as long as the circuit topology remains the same.
- The stop time of Transient analysis must be larger than the timepoint corresponding to the savestate file.
- Saved state file is binary and platform dependent.
- Savestate-Recover works only for transient analysis and the transient analysis must not be within a Sweep or Monte Carlo analysis.

Use Model

- Savestate

  Savestate is enabled/disabled by using the `spectre` command, as follows.

© 2003-2014 All Rights Reserved.

```
spectre [+savestate ] [-savestate ] ...
```

where:

- `+savestate` - Enables Savestate. You may use `+ss` as an abbreviation for `+savestate`.
- `-savestate` - Disables Savestate. You may use `-ss` as an abbreviation for `-savestate`.

By default, savestate is on, and checkpoint is off.

- Define saved time points and state file in the `tran` statement. For example:

```
DoTran tran stop=stoptime [ [saveperiod=time] | [saveclock=clock_time] |
[savetime=[time1 time2...]]   [savefile=file.srf]
```

where:

- `saveperiod`, `saveclock`, and `savetime` define the time points to save the states.
  - If `saveperiod` is given, Spectre generates a saved state file periodically based on the transient simulation time. Only the last saved state file is kept.
  - If `saveclock` is given, Spectre generates a saved state file periodically based on real time (wall clock time). Its default value is 1800 seconds (30 minutes).
  - If `savetime` is given, Spectre generates a saved state file on each specified time point.
- `savefile` defines where the saved states are written.
  - If `savefile` is not defined, the default file name is `%C.%A.srf`.

    Where `%C` is the input circuit file name, and `%A` is the analysis name.

If multiple save time points are given, That is,

```
analysisName tran stop=stoptime savetime=[time1 time2 ...] savefile=filename
```

the saved state file is `filename_at_time1`, `filename_at_time2`, and so on.

Besides saving the state based on saveperiod or saveclock or savetime, if savestate is enabled, Spectre automatically saves the states to a file when an interrupt signal like `QUIT`, `TERM`, `INT`, or `HUP` is received for the first time. If interrupt signals are received more than once, Spectre quits immediately.

The `saveclock`, `saveperiod`, and `savetime` parameters should not be specified at the same time.

If more than one parameter is specified, Spectre reads them in the following order:

© 2003-2014 All Rights Reserved.

```
saveperiod
savetime
saveclock
```

- Recover

  There are two ways to recover the simulation from the saved state file. The first is to define the recover file using the `spectre` command. The second is to define the recover file in a `tran` statement.

  Defining the recover file in the `tran` statement is strongly recommended, especially if there are multiple analyses statements in the netlist.

- Recover from command line

  ```
  spectre [+recover[=filename]]  [-recover] ...
  ```

  where:

  - `+recover` - Enables recover. You may use `+rec` as an abbreviation for `+recover`.
  - `-recover` - Disables recover. You may use `-rec` as an abbreviation for `-recover`.

- Recover from the `tran` statement

  ```
  analysisName tran  recover=filename ...
  ```

  By default, recover is disabled.

- Output Directory on Recovering

  When recovering from a saved state in a Spectre run by using `+recover=state_file` in a command-line option, a new raw directory is created to avoid overwriting the previous simulation results. However, if `recover=state_file` is given in a `tran` statement, the default raw directory is used.

  When defining `recover=state_file` in a `tran` statement, use a different tran name to avoid previous simulation results to be overwritten by the recovered results.

  When a new raw directory is created, the raw directory name is the same as the default raw directory, except that an index (starting from 0) is suffixed to raw, such as `*.raw#`, where `#` is 0, 1, 2, and so on.

  For example, in the first run, enter:

  ```
  spectre input.scs
  ```

  Spectre saves the simulation state in a file on a time point. By default, `input.raw` directory is created.

  When Spectre runs in recover mode:

  ```
  spectre +recover=saved_state_file input.scs
  ```


© 2003-2014 All Rights Reserved.

a new raw directory named `input.raw0` is created. The index of the raw directory is increased by one at each successive recover run. Another use model is that you define multiple transient-analysis runs in a netlist. In the first transient-analysis run, Spectre saves the simulation state on a time point.

In the next transient analysis run, simulation is continued from the saved time point. For example:

```
tran1 tran step=1ps stop=200ns savetime=[50ns] savefile=tran1_save
tran2 tran step=1p stop=400ns  recover=tran1_save_at_50.00ns
```

In this case, the default raw directory is used.

© 2003-2014 All Rights Reserved.

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

```
sens (output_variables_list) to (design_parameters_list) for (analyses_list)
```

where:

- `output_variables_list` = `ovar1` `ovar2`, and so on.
- `design_parameters_list` = `dpar1`, `dpar2`, and so on.
- `analyses_list` = `anal1` `anal2` , and so on.

The list of 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 attempts 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 if 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 may also 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` are computed with respect to `dc` voltage level of voltage source `vcc` and model parameter `rb` for the DC analysis `analDC`. The results are stored in the raw directory with the name in the format `analysisname.sens.analysistype`, that is, `dc.sens.analDC`.

- `sens (1 n2 7) to (q1:area nbjt1:rb) for (analAC)`

© 2003-2014 All Rights Reserved.

For this statement, AC sensitivities of nodes 1, n2, and 7 are 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 are stored in the output file `analAC.sens.ac`.

- `sens (1 n2 7) for (analAC)`

  For this statement, AC sensitivities of nodes 1, n2, and 7 are 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 are stored in the file in the format `ac.sens.analAC`.

- `sens (vbb:p q1:int_c q1:gm 7) to (q1:area nbjt1:rb) for (analDC1)`

  For this statement, DC sensitivities of 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 are computed with respect to instance parameter `area` for instance `q1` and model parameter `rb` for model `nbjt1`.

© 2003-2014 All Rights Reserved.

SpectreRF Summary (spectrerf)

Description

SpectreRF is an optional collection of analyses that is 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 and are capable of computing large-and-small-signal behavior, including noise behavior. Therefore, 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 about the SpectreRF analyses, run `spectre -help analysisName` where `analysisName` is `pss`, `pac`, `pxf`, `pnoise`, `psp`, `qpss`, `qpac`, `qpxf`, `qpnoise`, `qpsp`, `envlp`, `hb`, `hbac`, or `hbnoise`.

© 2003-2014 All Rights Reserved.

Stitch Flow Use Model (stitch)

Description

Stitching enables APS to plug in the parasitic elements on the fly during simulation. Compared to the flat RC netlist and the hierarchical RC netlist approaches, the Stitching flow has the following advantages:

- Reuse of the pre-layout simulation test-bench. There is no need to change the probe and the measure statements.
- What-if analysis powered by selective stitching.

APS stitching is enabled by options.

Parasitic File Loading Parameters

- `spf`

  This option specifies the to-be-stitched DSPF file and its stitching scope. The syntax is `spf="scope filename"`. The scope can be a subcircuit or an instance. When a subcircuit is specified as the scope, the DSPF file is stitched to all the instances of that subcircuit. When an instance is specified as the scope, the DSPF file is stitched to that instance only. Multiple DSPF files can be specified for stitching by using the option multiple times.

  Example:

  ```
  spf="mem mem.dspf"
  ```

- `dpf`

  This option specifies the to-be-stitched DPF file and its stitching scope. The syntax is `dpf="scope filename"`. The scope can be a subcircuit or an instance. When a subcircuit is specified as the scope, the DPF file is stitched to all the instances of that subcircuit. When an instance is specified as the scope, the DPF file is stitched to that instance only. Multiple DPF files can be specified for stitching by using the option multiple times.

  Example:

  ```
  dpf="X1.XPLL PLL.dpf" dpf="X1.XMEM mem.dpf"
  ```

  This means that the `PLL.dpf` file needs to be stitched to the `X1.XPLL` instance and the `mem.dpf` file needs to be stitched to the `X1.XMEM` instance.

- `spef`

© 2003-2014 All Rights Reserved.

This option specifies the to-be-stitched SPEF file and its stitching scope. The syntax is `spef="scope filename"`. The scope can be a subcircuit or an instance. When a subcircuit is specified as the scope, the SPEF file is stitched to all the instances of that subcircuit. When an instance is specified as the scope, the SPEF file is stitched to that instance only. Multiple SPEF files can be specified for stitching by using the option multiple times.

Example:

```
spef="adc a.spef"
```

Stitching Parsing Options

- `spfscale`

  This option specifies the scaling factor of all the elements in the parasitic files (DSPF/SPEF/DPF). For example, consider that the width of a device in the DSPF file is w=2. If spfscale=1.0e-6 is used, the actual width will be w=2.0e-6

- `spfswapterm`

  This option specifies the swappable terminals of a subcircuit macro-model. The syntax is `spfswapterm="terminal1 terminal2 subcktname"`.

  Example:

  ```
  spfswapterm="n1 n2 nch_mac"
  ```

  This indicates that terminals `n1` and `n2` of `subckt nch_mac` are swappable. In general, this is applicable to devices that are modeled by subcircuits. Multiple `spfswapterm` statements are supported.

- `spfxtorprefix`

  This option specifies the prefix in the names (devices and nets) in the DSPF/SPEF/DPF file. The device names in the prelayout netlist and the DSPF/SPEF file often do not match. The `spfxtorprefix` option can be used to help match the device names. The syntax is `spfxtorprefix="<substring> [<replace_substring>]"`.

  Example:

  ```
  spfxtorprefix="XM X"
  ```

  `XX1/XM1` exists in the prelayout netlist but the corresponding device name in the DSPF file is `XMX1/XM1`. This option will change `XM` to `X`.

- `spfaliasterm`

  Sometimes the terminal names of devices in DSPF/SPEF/DPF files are different from those in the simulation model library. This happens often in the technology nodes that

© 2003-2014 All Rights Reserved.

uses subcircuits to model devices. The syntax is `spfaliasterm="<model|subckt> <prelayout_term1>=<spf_alias1> <prelayout_term2>=<spf_alias2>... <prelayout_termN>=<spf_aliasN>"`. Multiple statements are supported.

Example:

```
spfaliasterm="nfet_mac n1=D n2=G n3=S n4=B"
```

This means that in subckt `nfet_mac`, terminal `n1` corresponds to terminal `D` in the DSPF file, `n1` corresponds to terminal `G`, `n3` corresponds to terminal `S` and `n4` corresponds to terminal `B`.

- `speftriplet`

  This option specifies the value that should be used for stitching in the SPEF file. This is effective only when the values in the SPEF file are represented by triplets (for instance, `0.325:0.41:0.495`). Default value is 2. Possible values are 1, 2 and 3.

Selective Stitching Options

- `spfcnet`

  This option specifies the net that has its total capacitance stitched. All other parasitic components, say parasitic resistors, associated with this net are ignored. The full hierarchical names are required. Multiple statements are supported. Wildcards are supported.

  Example:

  ```
  spfcnet=X1.netA
  ```

- `spfcnetfile`

  This option has the same functionality as `spfcent`.However, it accepts a text file in which all the C-only stitched nets are listed. Only one file can be specified. The syntax is `spfcnetfile="filename"`. The format in the file is one line per net.

  Example:

  ```
  spfcnetfile="nets.tex"
  ```

  The format in the nets.tex is:

  ```
  netA
  netB
  netC
  ```

- `spfrcnet`

  This option specifies the name of the net to be stitched with parasitic resistors and capacitors. The other nets are stitched with lumped total capacitances. Multiple

© 2003-2014 All Rights Reserved.

statements are supported. Wildcards are supported and you can specify multiple nets. Full hierarchical names are required.

Example:

```
spfrcnet=netA
```

- `spfrcnetfile`

  This option has the same functionality as `spfrcnet`. However, it accepts a text file in which all the RC stitched nets are specified. Only one file can be specified. The syntax is `spfrcnetfile="filename"`. The format in the file is one line per net.

  Example:

  ```
  spfrcnetfile="nets.tex"
  ```

  The format in the nets.tex is:

  ```
  netA
  netB
  netC
  ```

- `spfnetcmin`

  This option allows you to select the net for stitching by the value of its total node capacitance. If a net's total node capacitance exceeds `spfnetcmin`, all parasitics associated with the net are stitched correctly, otherwise, only the total capacitance is added to the net node.

  Example:

  ```
  spfnetcmin=1.0e-6
  ```

- `spfskipnet`

  This option specifies the net to be skipped for stitching, that is, all parasitic components of the net are not stitched. Wildcards and multiple statements are supported.

  Example:

  ```
  spfskipnet=X1.nodeA
  ```

- `spfskipnetfile`

  This option allows you to specify the nets to be skipped as a list in the text file called `file_name`. The syntax is `spfskipnetfile="filename"`. Only one file can be specified. The format in the file is one line per net.

  Example:

  ```
  spfskipnetfile="nets.tex"
  ```

  `nets.tex` file format is:

© 2003-2014 All Rights Reserved.

```
netA
netB
netC
```

Stitching Message Control Option

- `spfmsglimit`

  This option specifies the maximum number of messages to be printed in the `spfrpt` file. The messages in the `spfrpt` file are categorized by their ID number (STITCH-ID). This option specifies the maximum number of messages for a particular type of messages by using their STITCH-ID. The syntax is `spfmsglimit="number STITCH-ID_1 STITCH-ID_2"`. When STITCH-ID is not specified, the tool assigns the maximum message number limit to all messages categories (STITCH-IDs).

  Example:

  ```
  spfmsglimit="10 STITCH-0010"
  ```

  This tells the tool to print not more than 10 messages for the STITCH-0010 message category, in the meantime, for the other message categories, the default maximum limit of 50 messages will apply.

© 2003-2014 All Rights Reserved.

Subcircuit Definitions (subckt)

Description

Hierarchical Circuit

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. Therefore, a subcircuit definition may contain instances of other subcircuits. Subcircuits may also contain component, analysis, or model statements. Subcircuit definitions can 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, parameters that have been declared for the outer subcircuit definition are also available within the inner subcircuit definition. Parameters that are specified are referred to by name, optionally followed by an `=` 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 details about Spectre expression handling capability. Run `spectre -h parameters` for details about how Spectre handles netlist parameters, including subcircuit parameters, and how they inherit within nested subcircuit definitions.

Subcircuits always have an implicitly defined parameter `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. Any attempt to explicitly define `m` on a `parameters` line results in an error. In addition, because `m` is only implicitly defined, it is not available for use in expressions in the subcircuit.

© 2003-2014 All Rights Reserved.

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. It instead inherits the subckt call name. 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, based on 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. When the subcircuit is flattened, the `inline` component does not take on a hierarchical name such as `X1.M1`, it instead takes on the name of the subckt call, such as `X1`. Any non-inline components in the subckt take on the regular hierarchical name, as if the subcircuit were a regular subckt (that is, 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:

- All terminal currents. For example, `save q1:currents`
- Specific (index) terminal current. For example, `save q1:1 (#1=collector)`
- Specific (named) terminal current. For example, `save q1:b ("b"=base)`
- Save all operating point info. For example, `save q1:oppoint`
- Save specific operating point info. For example, `save q1:vbe`
- Save all currents and oppoint info. For example, `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 that references that model, you can create instances of the device, and each device automatically gets an appropriately scaled model assigned to it. For example, the instance parameters to an inline

© 2003-2014 All Rights Reserved.

subckt could represent something like emitter width and length of a BJT device, and within the subckt, a model card that is parameterized for emitter width and length and scales can be created accordingly. When you instantiate the macro, supply 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, the inline device can be probed just as if it were 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 Spectres `structural if` statement with inline subckts to perform automatic model selection based on `any` netlist/ subckt parameter.

Definition

```
[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:

```
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` and `M3`)

```
M1 (2 1 0 0) mos_mod
```


© 2003-2014 All Rights Reserved.

```
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. However, the parasitic capacitors (which you may not really interested in, or perhaps, even aware of, if the inline subckt definition was written by a different modeling engineer) have full hierarchical names.

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 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
```


© 2003-2014 All Rights Reserved.

Vec/Vcd/Evcd Digital Stimulus (vector)

Description

Spectre supports Digital Vector (VEC), Verilog-Value Change Dump (VCD), and Extended Verilog-Value Change Dump (EVCD).

VEC

To process digital vector files, the following command card needs to be specified in the netlist.

In Spice netlist:

```
.vec "vector_filename" [HLCheck = 0|1]
```

or

```
.vec vector_filename [HLCheck = 0|1]
```

Quotation marks can be double or single in Spice Netlist.

In Spectre netlist,

```
vec_include "vector_filename" [HLCheck = 0|1]
```

where,

`vector_filename` is the filename of the digital vector file.

`HLCheck = 0 | 1` is a special flag (default = off) to create the vector output check for the H and L states of input signals. Bidirectional and output signals always check H and L states and are unaffected by the HLCheck flag.

Normally, you do not need to use the HLCheck flag.

Each command card specifies only one VEC file. If a netlist needs to include multiple VEC files, `multiple.vec`/`vec_include` cards must be used. For example, if a netlist contains three VEC files, it needs three .vec cards, as given below:

```
.vec "file1.vec"
.vec "file2.vec"
.vec "file3.vec"
```


© 2003-2014 All Rights Reserved.

VCD/EVCD

VCD and EVCD formats are widely used in digital circuit design and contain different kinds of information for transistor-level simulation. You need to provide signal information, such as timing characteristics, voltage threshold, and driving ability of input signals, for each VCD or EVCD file.

Because VCD and EVCD formats are compatible, the same signal information file can be shared between them.

The VCD file (ASCII format) contains information about value changes for selected variables in the circuit design. Spectre simulator supports two types of VCD files:

`Four states` - represents variable changes in 0, 1, x (unknown or not needed), and z (tri-state) without providing strength information and port direction.

`Extended` - represents variable changes in all states and provides strength information and port direction.

To process the VCD/EVCD file in Spectre, the following command card needs to be specified in the netlist.

In Spice netlist:

```
.vcd "vcd_filename" "signal_info_filename"
.evcd "evcd_filename" "signal_info_filename"
```

In Spectre netlist:

```
vcd_include  "vcd_filename" "signal_info_filename"
evcd_include "evcd_filename" "signal_info_filename"
```

Each command card specifies only one VCD file. If a netlist needs to include multiple VCD files, `multiple.vcd`/`vcd_include` cards must be used. For example, if a netlist contains three VCD files, it needs three `.vcd` cards, as given below:

```
.vcd "file1.vcd" "file1.signal"
.vcd "file2.vcd" "file2.signal"
.vcd "file3.vcd" "file3.signal"
```

Output Check

For VEC, VCD, and EVCD output check, the results are written in two files under the raw directory, one a check error report file `%A.vecerr` and the other a check summary report file `%A.veclog` where, `%A` is the analysis name.

© 2003-2014 All Rights Reserved.

To find VEC VCD and EVCD file, `signal_info` file format description and examples, refer to the *Spectre User Guide* for details.

© 2003-2014 All Rights Reserved.

Verilog-A Usage and Language Summary (veriloga)

Description

Verilog-A is an analog hardware description language standard from Open Verilog International. It enables analog circuit behavior to be described at a high level of abstraction. Behavioral descriptions of modules and components may be instantiated in a Spectre netlist along with regular Spectre primitives.

Verilog-A descriptions are written in files different 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`, that has two nodes and two parameters.

AHDL Linter can be used to improve Verilog-A model quality. It can help to avoid potential convergence or performance problems, and to improve model accuracy, reusability and portability. Refer to the *Verilog-A Language Reference manual* for more information.

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

```
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
        @ ( initial_step ) begin
        // performed at the first timestep of an analysis
        end
```


© 2003-2014 All Rights Reserved.

```
        // module behavioral description
        V(n1, n2) <+ I(n1, n2) * arg1;
        @ ( final_step ) begin
        // performed at the last time step of an analysis
        end
    end
endmodule
```

Verilog-A Compilation

The simulation performance of Verilog-A has been improved by performing a onetime compilation step. The performance improvement obtained is proportional to the complexity and amount of Verilog-A in your design. Following the initial compilation, recompilation is performed only if the Verilog-A source is changed.

Verilog-A compilation is enabled by default. If you are making frequent changes to 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`, as follows:

```
setenv CDS_AHDLCMI_ENABLE NO
```

To re-enable Verilog-A compilation, set the `CDS_AHDLCMI_ENABLE` to `YES`. For example,

```
setenv CDS_AHDLCMI_ENABLE YES
```

To re-enable Verilog-A compilation, you can also undefine the `CDS_AHDLCMI_ENABLE` environment variable, as follows:

```
unsetenv CDS_AHDLCMI_ENABLE
```

Note that Verilog-A compilation cannot be turned off in APS.

The compiled C code flow stores the compiled shared objects in a database on the disk for the simulation to use. The shared objects are stored in a directory named `ahdlSimDB`. By default, this database is created in the current working directory and given a name created by appending `.ahdlSimDB` to the circuit name. You can specify an alternative location for ahdlSimDB by setting the `CDS_AHDLCMI_SIMDB_DIR` environment variable to the path of a directory, as follows:

```
setenv CDS_AHDLCMI_SIMDB_DIR /projects/ahdlcmiSimDirs
```

If the path is writable, `adhlSimDB` is created there. If the path is not writable or does not exist, an error is reported.

To store compiled objects, use a second type of database, named `ahdlShipDBs`. To create such databases, set the `CDS_AHDLCMI_SHIPDB_COPY` to `YES`, as follows:

```
setenv CDS_AHDLCMI_SHIPDB_COPY YES
```


© 2003-2014 All Rights Reserved.

In this case, an `ahdlShipDB` is created for each Verilog-A file in the directory that contains the Verilog-A files, if the directory is writable. If the directory is not writable, no `ahdlShipDBs` are created for the modules in the Verilog-A file that is being processed.

If the `CDS_AHDLCMI_SHIPDB_DIR` environment variable (or the equivalent, but obsolete, `CDS_AHDLCMI_DIR` variable) is also set to a writable path, the `ahdlShipDB` database is created there and shared by all the Verilog-A files used for simulations that are run while this environmental variable is set. If the `CDS_AHDLCMI_SHIPDB_DIR` is not set to a writable path or the path does not exist, a warning is reported and ahdlShipDBs are not created.

Language Summary

The following provides a summary of the Verilog-A analog hardware description language. For more information refer to *Verilog-A Reference Manual*.

Analog Operators/Waveform Filters

`ddt(x <,abstol> )` Differentiate x with respect to time.

`idt(x, ic <, assert <, abstol> > )`
Integrate x with respect to time. Output = ic during dc analysis and when assert is 1.

`idtmod(x, <ic <, modulus <, offset> > > )`
Circular Integration of x with respect to 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, that is, 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, < ttransition <,sample offset time > )`
z-domain filter function, numerator-denominator form.

© 2003-2014 All Rights Reserved.

`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 > )`

© 2003-2014 All Rights Reserved.

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.

© 2003-2014 All Rights Reserved.

`initial_step< ( arg1 <, arg2 <, etc... > > )`

Generate an event at the initial step of an analysis. arg1, arg2, and so on. Examples of analyses strings are "dc", "tran", "ac", "pss", "noise", "pdisto", "qpss", "pac", "pnoise", "pxf", "sp", "tdr", "xf", "envlp", "psp", "qpsp", "qpac", "qpnoise", "qpxf", "static", "ic", and so on.

`final_step< ( arg1 <, arg2 <, etc... > > )`

Generate an event at the final step of an analysis. arg1, arg2, and so on. Examples of analyses strings are "dc", "tran", "ac", "pss", "noise", "pdisto", "qpss", "pac", "pnoise", "pxf", "sp", "tdr", "xf", "envlp", "psp", "qpsp", "qpac", "qpnoise", "qpxf", "static", "ic", and so on.

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.

© 2003-2014 All Rights Reserved.

`$fscanf(fptr, "format string" <, arguments>)`

Read data from a file

`$fopen("filename", mode)` Open a file for reading/writing

`$fclose(fptr)` Close a file

`$finish<(n)>` Finish the simulation

`$stop<(n)>` Stop the simulation

Simulator Environment Functions

`$realtime` Returns current simulation time

`$temperature` Returns ambient simulation temperature (K)

`$vt` Returns thermal voltage

`$vt(temp)` Returns thermal voltage at given temp

`$analysis(analysis_string1<, analysis_string2 <, ...>>)`

Returns true(1) if the current analysis phase matches one of the given analyses strings. The following are the examples of analyses strings: "dc", "tran", "ac", "pss", "noise", "pdisto", "qpss", "pac", "pnoise", "pxf", "sp", "tdr", "xf", "envlp", "psp", "qpsp", "qpac", "qpnoise", "qpxf", "static", "ic", and so on.

Parameter Functions

`$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.

© 2003-2014 All Rights Reserved.

Data Qualifiers

`parameter` 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.

```
module_or_primative #(<.param1(expr1)<,...>>) inst_name (<node1 <,
                      ..>> );
```

Creates a new instance of module_or_primative named `inst_name`.

© 2003-2014 All Rights Reserved.

5

Circuit Checks

This chapter discusses the following topics:

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

Dynamic Subckt Activity Check (dyn_activity)

Description

Reports subckt activities in the form of a ratio, maintaining the hierarchy information.

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_activity parameter=value ...
```

Parameters

Filtering parameters

1 `time_window=[tstart, tstop] sec`
Time window during which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

2 `min_activity=0 %` Any block activity below the minimum activity percentage will not be reported. Default is 0%.

Wildcard scoping

3 `inst=[...]` Subcircuit instances to which the check is applied. Default is `none`.

© 2003-2014 All Rights Reserved.

Dynamic Active Node Check (dyn_actnode)

Description

The active node checking analysis detects nodes with voltage changes that exceed the user-defined threshold `dv`. The voltage change is defined as peak-to-peak voltage (Vpp) within a time window.

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_actnode parameter=value ...
```

Parameters

Design check parameters

1 `node="[...]"` Nodes to which the check is applied. Default is all (node=*).

2 `dv=0.1 v` Voltage change threshold for the active nodes. Default is 0.1 volt.

3 `type=act` Report inactive or active nodes or both.
Possible values are `act`, `inact`, and `both`.

Filtering parameters

4 `time_window=[tstart, tstop] sec`
Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is 0 to tend.

5 `error_limit=10000`
Maximum number of errors reported. Default is 10000.

Wildcard scoping

6 `inst=[...]` Subcircuit instances to which the check is applied. Default includes all instances (inst=*).

© 2003-2014 All Rights Reserved.

7	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is none.
8	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).
9	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is none.
10	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic Capacitor Voltage Check (dyn_capv)

Description

Reports capacitor elements fulfilling the conditional expression on element voltages for a duration longer than the user-specified threshold duration.

Supported capacitor variables are: v(1,2), v(1), and v(2)

Supported operators are: +, -, *, /, ==, !=, <, <=, >, >=, ||, &&, and !

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

This check is supported only by XPS, For Spectre and Spectre APS, use the `assert` statement.

Definition

```
Name dyn_capv parameter=value ...
```

Parameters

Design check parameters

1 `cond` The conditional expression to be fulfilled. Default is none.

2 `duration=5.00E-09 sec`
Duration threshold. Default is `5.00E-09 sec`.

Filtering parameters

3 `time_window=[tstart, tstop] sec`
The time window during which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

4 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

5	`inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
6	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
7	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
8	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
9	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

```
cond  1            error_limit  4     time_window  3

depth  9           inst  5           xinst  6

duration  2        subckt  7         xsubckt  8
```


© 2003-2014 All Rights Reserved.

Dynamic DC Leakage Path Check (dyn_dcpath)

Description

Reports conductance paths between user-specified nets. The qualifying paths carry an absolute current higher than the current threshold (`ith`) and for a time longer than the user-specified duration (`duration`).

If more than two nets are specified, Spectre checks the conducting path between each pair of nodes. For example, if net=[vdc1 vdc2 0] is specified, then the conducting path between vdc1 and vdc2, vdc1 and 0, and vdc2 and 0 is checked.

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_dcpath parameter=value ...
```

Parameters

Design check parameters

1 `net="[...]"`
The leakage path between the voltage source nets is checked. `net` defines a set of nodes and not a pair. For example, `net = [vdd vdd1 0]` checks the leakage path between vdd and vdd1, vdd and 0, and vdd1 and 0.

2 `ith=5.00E-05 A`
The leakage path with the absolute current higher than the threshold value is reported. Default is `5.00E-05 A`.

3 `duration=5.00E-09 sec`
Duration threshold. Default is `5.00E-09 sec`.

Filtering parameters

4 `time_window=[tstart, tstop] sec`
The time window during which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

© 2003-2014 All Rights Reserved.

5 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

Wildcard scoping

6 `leaki_times=[...] sec`
Time point(s) at which dynamic dcpath check is performed.

© 2003-2014 All Rights Reserved.

Dynamic Diode Voltage Check (dyn_diodev)

Description

Reports diode elements fulfilling the conditional expression on element voltages for a time longer than a user-specified duration threshold.

Supported diode variables: `v(a,c)`, `v(a)`, `v(c)`

Supported operators: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !

The results are reported into a file with the extension dynamic.xml, which can be read with a web browser. This check is supported only by XPS. For Spectre and Spectre APS use the `assert` statement.

Definition

```
Name dyn_diodev parameter=value ...
```

Parameters

Design check parameters

1 `cond` The conditional expression to be fulfilled. Default is none.

2 `duration=5.00E-09 sec`
Duration threshold. Default is 5.00E-09 sec.

3 `model=[...]` model names to include.

Filtering parameters

4 `time_window=[tstart, tstop] sec`
Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is 0 to tend.

5 `error_limit=10000`
Maximum number of errors reported. Default is 10000.

© 2003-2014 All Rights Reserved.

Wildcard scoping parameters

6	`inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
7	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is none.
8	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
9	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is none.
10	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic Excessive Element Current Check (dyn_exi)

Description

Reports elements and devices carrying currents (absolute value) higher than the current threshold (`ith`) for a time longer than the duration threshold (`duration`). The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_exi parameter=value ...
```

Parameters

Design check parameters

1 `dev="[...]"` The instance names of device or elements to be checked.

2 `ith (ampere)` Current threshold. Default is `none`.

3 `duration=5.00E-09 sec`
Duration threshold. Default is `5.00E-09 sec`.

Filtering parameters

4 `time_window=[tstart, tstop] sec`
The time window during which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

5 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

Wildcard scoping

6 `inst=[...]` Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).

© 2003-2014 All Rights Reserved.

7	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
8	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
9	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
10	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic Excessive Rise, Fall, Undefined State Time Check (dyn_exrf)

Description

Reports the nodes with excessive rise times, fall times, or with an undefined state. The rise and fall times are measured between low (vlth) and high (vhth) voltage thresholds. The duration longer than the user-specified rise and fall time threshold are reported. Undefined states are defined by node voltages between low and high voltage threshold and are reported when their duration is longer than the undefined time threshold (utime).

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_exrf parameter=value ...
```

Parameters

Design check parameters

1	`node="[...]"`	Nodes to which the check is applied. Default is all (`node=*`).
2	`rise (sec)`	Risetimes longer than the specified value are reported. Default is `none`.
3	`fall (sec)`	Falltimes longer than the specified value are reported. Default is `none`.
4	`utime (sec)`	The undefined time threshold. An undefined state is defined by node voltages between the low and high voltage thresholds.

Digitize parameters

5	`vlth (V)`	Low voltage threshold for rise, fall, and utime measurements. Default is `none`.
6	`vhth (V)`	High voltage threshold for rise, fall, and utime measurements. Default is `none`.

© 2003-2014 All Rights Reserved.

Filtering parameters

7 `time_window=[tstart, tstop] sec`
The time window during which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

8 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

Wildcard scoping

9 `inst=[...]` Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).

10 `xinst=[...]` Subcircuit instances to be excluded from the check. Default is `none`.

11 `subckt=[...]` The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).

12 `xsubckt=[...]` The instances of the specified subcircuits that are excluded from the check. Default is `none`.

13 `depth=8` Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic Floating Node Induced DC Leakage Path Check (dyn_floatdcpath)

Description

Reports the DC leakage paths that are caused by floating nodes. The check is performed at specified times (`leaki_times`) of a transient simulation. Floating nodes and MOSFET devices with floating gates are detected. Potential leakage paths caused by floating gate MOSFET devices are reported.The leakage paths are checked between user-specified power supply nodes (net).

A node is considered floating if it has no conducting path to a power supply or ground. The following device conducting rules are used for the floating node detection:

- MOSFET is conducting if ids>mos_ith OR gds>mos_gds

- Resistors, controlled resistors, and inductors are conducting if R<=res_th

- BJT is conducting if Vbe>bjt_vbe OR Ic>bjt_ith

- Diode is conducting if V>diode_vth

- Vsources and iprobes are considered conducting

- Isources, VCCS, and CCCS are conducting if i>isource_ith

- JFET is considered conducting

- Mutual inductors and controlled capacitors are not conducting

- VerilogA: conducting path depends on module details

A report of all MOSFETs with floating gates can be printed by using the parameter `floatgate`. The leakage path detection uses the following conducting rules:

- MOSFET with floating gate is always considered conducting.

- Resistors, controlled resistors, and inductors are conducting if R<=res_th

- BJT is conducting if Vbe>bjt_vbe OR Ic>bjt_ith

- Diode is conducting if V>diode_vth

- Vsources is conducting if V=0 otherwise it is not conducting

© 2003-2014 All Rights Reserved.

- iprobes are considered conducting

- Isources, VCCS, and CCCS are conducting if i>isource_ith

- VerilogA: conducting path depends on module details

If more than two nets are specified, Spectre checks the leakage path between each pair of nets. For example, if net=[vdc1 vdc2 0] is specified, then the conducting path between vdc1 and vdc2, vdc1 and 0, and vdc2 and 0 is checked.

Higher accuracy settings are recommended when using the XPS solver: +cktpreset=sram_pwr.

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_floatdcpath parameter=value ...
```

Parameters

Design check parameters

1	`net="[...]"`	Hierarchical node names between which the leakage path is checked. Wildcarding is not allowed.
2	`mos_ith=100E-9`	Mosfet ids conducting threshold for high impedance node detection. Default is `100 nA`.
3	`mos_gds=1.0E-5`	Mosfet gds conducting threshold for floating node detection. Default is `1e-5`.
4	`bjt_vbe=0.4`	BJT vbe conducting threshold for floating node detection. Default is `0.4 V`.
5	`bjt_ith=50E-9`	BJT ic conducting threshold for floating node detection. Default is `50 nA`.
6	`res_th=100E6`	Resistor conducting threshold for floating node detection. Default is `100 MOhm`.
7	`diode_vth=0.6`	Diode voltage conducting threshold for high impedance node detection. Default is `0.6V`.

© 2003-2014 All Rights Reserved.

8 `isource_ith=1.0E-12`
Current source conducting threshold for floating node detection. Default is `1 pA`.

9 `detailed_path=per_fm`
If set to yes, prints the detailed path. Default is per_fm, print one path per floating mosfet. If set to per_fn, print one path per floating node. Possible values are per_fm, yes, and per_fn. Possible values are `per_fm`, `yes`, and `per_fn`.

10 `leaki_times=[...] sec`
time point(s) at which dynamic floatdcpath check is performed.

11 `floatgate=no`
Whether to report all MOSFETs with floating gate at leaki_times. Possible values are no and yes. Default is no.
Possible values are `no` and `yes`.

Filtering parameters

10 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic Glitch Check (dyn_glitch)

Description

A 'Glitch' occurs when:

- A low signal goes above the mid-level, and crosses the mid-level again within the user-defined duration.

- A high signal goes below the mid-level, and crosses the mid-level again within the user-defined duration.

This check applies only to blocks with single and constant power supply. The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_glitch parameter=value ...
```

Parameters

Design check parameters

1 `node="[...]"` Nodes to which the check is applied. Default is all (`node=*`).

2 `duration=5.00E-09 sec`
Duration threshold. Default is `5.00E-09 sec`.

3 `low=0 volt` Low-level voltage. Default is `0V`.

4 `high (volt)` High-level voltage. Default is `none`.

5 `mid=0.5 (high + low) volt`
Mid-level voltage for glitch detection. Default is `0.5*(high+low)`.

Filtering parameters

6 `time_window=[tstart, tstop] sec`
The time window during which the circuit check is applied.

© 2003-2014 All Rights Reserved.

Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

7 `error_limit=10000`

Maximum number of errors reported. Default is `10000`.

Wildcard scoping

8 `inst=[...]` Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).

9 `xinst=[...]` Subcircuit instances to be excluded from the check. Default is `none`.

10 `subckt=[...]` The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).

11 `xsubckt=[...]` The instances of the specified subcircuits that are excluded from the check. Default is `none`.

12 `depth=8` Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

depth 12	high 4	mid 5	time_window 6
duration 2	inst 8	node 1	xinst 9
error_limit 7	low 3	subckt 10	xsubckt 11

© 2003-2014 All Rights Reserved.

Dynamic HighZ Node Check (dyn_highz)

Description

Reports the nodes that are in high impedance state for a duration longer than the user-defined threshold. A high impedance state is reached when there is no DC path from the node to any power supply or ground. The following device conditions determine the high impedance nodes:

- MOSFET is conducting if ids>mos_ith or gds>mos_gds
- Resistors, controlled resistors, and inductors are conducting if R<=res_th
- BJT is conducting if Vbe>bjt_vbe or Ic>bjt_ith
- Diode is conducting if V>diode_vth
- Vsources and iprobes are considered conducting
- Isources, VCCS, and CCCS are conducting if i>isource_ith
- JFET is considered conducting

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_highz parameter=value ...
```

Parameters

Design check parameters

1 `node="[...]"` Nodes to which the check is applied. Default is all (`node=*`).

2 `duration=5.00E-09 sec`
Duration threshold. Default is `5.00E-09 sec`.

3 `mos_ith=100E-9` Mosfet ids conducting threshold for high impedance node detection. Default is `100 nA`.

© 2003-2014 All Rights Reserved.

4 `mos_gds=1.0E-5` Mosfet gds conducting threshold for high impedance node detection. Default is `1e-5`.

5 `bjt_vbe=0.4` BJT vbe conducting threshold for high impedance node detection. Default is `0.4 V`.

6 `bjt_ith=50E-9` BJT ic conducting threshold for high impedance node detection. Default is `50 nA`.

7 `res_th=100E6` Resistor conducting threshold for high impedance node detection. Default is `100 MOhms`.

8 `diode_vth=0.6` Diode voltage conducting threshold for high impedance node detection. Default is `0.6 V`.

9 `isource_ith=1.0E-12` Current source conducting threshold for high impedance node detection. Default is `1 pA`.

10 `inverse=no` If set to no, reports all nodes that are in highz sate. If set to yes, reports all nodes not being in highz state. Default is no. Possible values are `no` and `yes`.

Filtering parameters

11 `time_window=[tstart, tstop] sec` Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is 0 to tend.

12 `error_limit=10000` Maximum number of errors reported. Default is 10000.

13 `fanout=all` Fanout setting to filter node with specified connection. Possible values are `all`, `gate`, and `bulk`.

Wildcard scoping

14 `inst=[...]` Subcircuit instances to which the check is applied. Default includes all instances (inst=*).

© 2003-2014 All Rights Reserved.

15	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is none.
16	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).
17	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is none.
18	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic MOSFET Voltage Check (dyn_mosv)

Description

Reports MOSFET devices fulfilling the conditional expression on device voltages and device size (w, l) for a time longer than the user-specified threshold duration.

Supported MOSFET variables are: v(g,s), v(g,d), v(g,b), v(d,s), v(d,b), v(s,b), v(g), v(d), v(s), v(b), l, and w

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

This check is supported only by XPS. For Spectre and Spectre APS use the `assert` statement.

Definition

```
Name dyn_mosv parameter=value ...
```

Parameters

Design check parameters

1 `model="[...]"` MOSFET device model names to be checked.

2 `cond` The conditional expression to be fulfilled. Default is `none`.

3 `duration=5.00E-09 sec`
Duration threshold. Default is `5.00E-09 sec`.

Filtering parameters

4 `time_window=[tstart, tstop] sec`
The time window during which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

5 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

6	`inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
7	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
8	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
9	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
10	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic Node Capacitance Check (dyn_nodecap)

Description

Reports the node capacitance at specified times (`time`) of a transient simulation. Device capacitances, grounded capacitances, and coupling capacitances are combined into one value.

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_nodecap parameter=value ...
```

Parameters

Design check parameters

1	`node="[...]"`	Nodes for which the capacitance needs to be checked.

Filtering parameters

2	`time=[...] sec`	time point(s) at which dynamic nodecap check is performed.
3	`error_limit=10000`	Maximum number of errors reported. Default is 10000.

Wildcard scoping

4	`inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (inst=*).
5	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is none.
6	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).

© 2003-2014 All Rights Reserved.

7	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is none.
8	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic Noisy Node Check (dyn_noisynode)

Description

Identifies the nodes with unstable or noisy node conditions. A node is considered unstable or noisy if its voltage fulfills the condition abs(dV/dt)>e1 and abs(d(dV/dt)dt)>e2 for a time longer than duration. Stable periods shorter than skip are ignored.

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_noisynode parameter=value ...
```

Parameters

Design check parameters

1 `node="[...]"` Nodes to which the check is applied. Default is all (`node=*`).

2 `duration=5.00E-07 sec`
Duration threshold. Default is `5.00E-07 secs`.

3 `e1=5.00E04` The first derivative threshold. Default is `5.00E04`.

4 `e2=2.00E16` The second derivative threshold. Default is `2.00E16`.

5 `skip=50.0E-09 sec` The stable period less than skip is ignored. Default is `50.0E-09`.

Filtering parameters

6 `time_window=[tstart, tstop] sec`
The time window during which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

7 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

8	`inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
9	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
10	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
11	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
12	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic Pulse Width Check (dyn_pulsewidth)

Description

Reports nodes with pulse width error.

The pulse widths that fall outside of `pwmin_low` and `pwmax_low`, and `pwmin_high` and `pwmax_high` are reported.

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_pulsewidth parameter=value ...
```

Parameters

Design check parameters

1 `node="[...]"` Nodes to which the check is applied.

2 `pwmin_low=0.0 sec` The minimum value of the pulse width in logic 0 state.

3 `pwmax_low=infinity sec`
The maximum value of the pulse width in logic 0 state.

4 `pwmin_high=0.0 sec`
The minimum value of the pulse width in logic 1 state.

5 `pwmax_high=infinity sec`
The maximum value of the pulse width in logic 1 state.

Digitize parameters

6 `vlth=0.2 V` Low voltage threshold for signal net.

7 `vhth=0.8 V` High voltage threshold for signal net.

© 2003-2014 All Rights Reserved.

Filtering parameters

8 `time_window=[tstart, tstop] sec`
Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is 0 to tend.

9 `error_limit=10000`
Maximum number of errors reported. Default is 10000.

Wildcard scoping

10 `inst=[...]` Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).

11 `xinst=[...]` Subcircuit instances to be excluded from the check. Default is `none`.

12 `subckt=[...]` The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).

13 `xsubckt=[...]` The instances of the specified subcircuits that are excluded from the check. Default is `none`.

14 `depth=8` Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Dynamic Resistor Voltage Check (dyn_resv)

Description

Reports the resistor elements fulfilling the conditional expression on element voltages for a duration longer than the user-specified duration threshold.

Supported resistor variables are: v(1,2), v(1), and v(2)

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

This check is supported only by XPS. For Spectre and Spectre APS use the `assert` statement.

Definition

```
Name dyn_resv parameter=value ...
```

Parameters

Design check parameters

1 `cond` The conditional expression to be fulfilled. Default is `none`.

2 `duration=5.00E-09 sec`
Duration threshold. Default is `5.00E-09 secs`.

Filtering parameters

3 `time_window=[tstart, tstop] sec`
The time window during which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

4 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

5	`inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
6	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
7	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
8	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
9	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

```
cond  1            error_limit  4     time_window  3

depth  9           inst  5           xinst  6

duration  2        subckt  7         xsubckt  8
```


© 2003-2014 All Rights Reserved.

Dynamic Setup and Hold Check (dyn_setuphold)

Description

Reports nodes with setup or hold timing error. For setup timing, the transitions that happen between refTime + delay - setupTime and refTime + delay are reported. For hold timing, the transitions that happen between refTime + delay and refTime + delay + holdTime are reported.

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_setuphold parameter=value ...
```

Parameters

Design check parameters

1	`node="[...]"`	Nodes to which the check is applied. Default is all (`node=*`).
2	`ref_node`	Name of the referenced clock (net).
3	`setup_time=0.0 sec`	Setup time violation window. If specified, the setup check is enabled. Default is `0.0 sec`.
4	`hold_time=0.0 sec`	Hold time violation window. If specified, the hold check is enabled.
5	`delay=0.0 sec`	Delay of the reference signal.
6	`edge=rise`	Edge type of the signal net. Possible values are `rise`, `fall`, and `both`. Default is `rise`.
7	`ref_edge=rise`	Edge type of the reference signal. Possible values are `rise`, `fall`, and `both`. Default is `rise`.

© 2003-2014 All Rights Reserved.

Digitize parameters

8 `vlth=0.2 V` Low voltage threshold for signal net.

9 `ref_vlth=0.2 V` Low voltage threshold for referenced net.

10 `vhth=0.8 V` High voltage threshold for signal net.

11 `ref_vhth=0.8 V` High voltage threshold for referenced net.

Filtering parameters

12 `time_window=[tstart, tstop] sec`
The time window during which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

13 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

Wildcard scoping

14 `inst=[...]` Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).

15 `xinst=[...]` Subcircuit instances to be excluded from the check. Default is `none`.

16 `subckt=[...]` The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).

17 `xsubckt=[...]` The instances of the specified subcircuits that are excluded from the check. Default is `none`.

18 `depth=8` Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

delay 5	inst 14	ref_vlth 9	vlth 8
depth 18	node 1	setup_time 3	xinst 15
edge 6	ref_edge 7	subckt 16	xsubckt 17
error_limit 13	ref_node 2	time_window 12	
hold_time 4	ref_vhth 11	vhth 10	

© 2003-2014 All Rights Reserved.

Dynamic Subckt Port Power Check (dyn_subcktpwr)

Description

Reports port currents, port powers, and subcircuit powers.

The port current is positive when current is going into a subcircuit. This check will report average, RMS, and the maximum values of the current entering a port.

The power analysis can be done by using the `power` parameter. When the `power` parameter is set, two additional sections are generated. The first section reports the average, RMS, and the maximum power entering all the ports that are defined by the `port` parameter. The second section reports the average, RMS, and maximum power consumed by each instance of a subcircuit that are defined by the `inst` parameter. Note that for the second section, it is mandatory to have the parameter port set to [*].

Note that Max is defined as the maximum of the absolute of a waveform within a time_window. For example, consider a current entering a port having a peak value of 1mA and another peak value of -2mA below 0. Therefore, the check will report "Max (A)" as -2mA and the time point will be reported in "Max Time(s)". Therefore, this check will use an absolute function to find the maximum current/power but it will report that finding with a regular current/power. Unlike Max, the average and RMS are defined as the average or RMS value of a regular waveform within a time_window, respectively.

The `filter` parameter can be used to filter out ports that are connected only to the gate of MOSFET.

The results are reported to the dynamic.xml file, which can be viewed in a Web browser.

Definition

```
Name dyn_subcktpwr parameter=value ...
```

Parameters

Design check parameters

1 `port=[...]` Ports to be checked. Default is none.

2 `embedded_delimiter`
Used to set embedded hierarchical delimiter for post layout.

© 2003-2014 All Rights Reserved.

3 `power=off` Report power or not.
Possible values are `off` and `on`.

Filtering parameters

4 `filter=none` Check all ports or only those not connecting to gates.
Possible values are `none` and `gates`.

5 `time_window=[tstart, tstop] sec`
Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is 0 to tend.

6 `error_limit=10000`
Maximum number of errors reported. Default is 10000.

Wildcard scoping

7 `inst=[...]` Subcircuit instances to which the check is applied. Default is none.

8 `depth` Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is none.

Parameter Index

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


```
depth  8           error_limit  6     inst  7          power  3
```


© 2003-2014 All Rights Reserved.

Static Capacitor Check (static_capacitor)

Description

Reports all capacitors within or outside the range of `cmin` and `cmax`. It can also generate a distribution list of all the capacitors in a circuit. If the `type` parameter is set to range, all capacitors outside the range of `cmin` and `cmax` will be reported.

If the `type` parameter is set to `print`, all capacitors between `cmin` and `cmax` will be reported. In the report, the capacitor names can be sorted by clicking on the `Device name` header in a Web browser.

If the `type` parameter is set to `distr`, a distribution list will be generated for all capacitors. There are a maximum of 9 bins: -Inf - 0, 0 - 10a, 10a - 100a, 100a - 1f, 1f - 10f, 10f - 100f, 100f - 1p, 1p - 10p, and 10p - Inf

However, if two or more consecutive bins are empty, they will merge into one bin, reducing the number of bins. If type is set to `distr`, the parameters `cmin`, `cmax`, and `error_limit` are ignored.

The results are reported to the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_capacitor parameter=value ...
```

Parameters

Design check parameters

1	`type=print`	Checking types. Possible values are `range`, `distr`, and `print`.
2	`cmin=-1 F`	Minimum capacitor value.
3	`cmax=1 F`	Maximum capacitor value.

© 2003-2014 All Rights Reserved.

Filtering parameters

4 `error_limit=10000`

Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Static Capacitor Voltage Check (static_capv)

Description

Reports capacitor devices fulfilling the conditional expression on device voltages.

Supported capacitor variables are: v(1,2), v(1), and v(2)

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !

The results are written to the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_capv parameter=value ...
```

Parameters

Design check parameters

1	`cond`	The conditional expression to be fulfilled. Default is `none`.

Digitize parameters

2	`vlth=0.2 v`	DC sources with voltage lower than `vlth` carry static 0. Low voltage threshold. Default is `0.2 V`.
3	`vhth=0.8 v`	DC sources with voltage higher than `vhth` carry static 1. High voltage threshold. Default is `0.8 V`.
4	`pwl_time=`	Time for pwl src to be considered as constant vsrc.

Filtering parameters

5	`error_limit=10000`	Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

6 `inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
7 `xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
8 `subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
9 `xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
10 `depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static DC Leakage Path Check (static_dcpath)

Description

Reports the always conducting paths between the power supply nodes.

The results are written to the static.xml file, which can be viewed in a Web browser.

This check is supported only by XPS.

Definition

```
Name static_dcpath parameter=value ...
```

Parameters

Design check parameters

1 `net="[...]"` The leakage path between the voltage source nodes is checked. `net` defines a set of nodes and not a pair. For example, node = [vdd vdd1 0] checks the leakage path between vdd and vdd1, vdd and 0, and vdd1 and 0.

Digitize parameters

2 `vlth=0.2 v` DC sources with voltage lower than `vlth` carry static 0. Default is `0.2 V`.

3 `vhth=0.8 v` DC sources with voltage higher than `vhth` carry static 1. Default is `0.8 V`.

4 `pwl_time=`
Time for pwl src to be considered as constant vsrc.

Filtering parameters

4 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

5	`inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
6	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
7	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
8	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
9	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static Diode Voltage Check (static_diodev)

Description

Reports the diode devices fulfilling the conditional expression on device voltages.

Supported diode variables are: `v(a,c)`, `v(a)`, and `v(c)`

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !

The results are written to the static.xml file, which can be viewed in a Web browser.This check is supported only by XPS.

Definition

```
Name static_diodev parameter=value ...
```

Parameters

Design check parameters

1	`model="[...]"`	MOSFET device model names to be checked.
2	`cond`	The conditional expression to be fulfilled. Default is none.

Digitize parameters

3	`vlth=0.2 V`	DC sources with voltage below vlth carry static 0. Default is 0.2 V.
4	`vhth=0.8 V`	DC sources with voltage above vhth carry static 1. Default is 0.8 V.
5	`pwl_time=`	Time for pwl src to be considered as constant vsrc.

Filtering parameters

6	`error_limit=10000`	Maximum number of errors reported. Default is 10000.

© 2003-2014 All Rights Reserved.

Wildcard scoping

7	`inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (inst=*).
8	`xinst=[...]`	Subcircuit instances to be excluded from the check. Default is none.
9	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).
10	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is none.
11	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static ERC Check (static_erc)

Description

Performs various electrical rule checks and reports the devices with violations. The results are reported to the static.xml file, which can be viewed in a Web browser.

This check is only supported by XPS.

Definition

```
Name static_erc parameter=value ...
```

Parameters

Design check parameters

1	`hotwell=off`	Reports MOSFET with bulk not connected to VDD or GND. Possible values are `off` and `on`.
2	`floatbulk=off`	Reports MOSFET with floating bulk. Possible values are `off`, `all`, and `no_top`.
3	`floatgate=off`	Reports MOSFET with floating gate. Possible values are `off`, `all`, `no_top`, `no_moscap`, and `no_top_moscap`.
4	`dangle=off`	Reports the dangling nodes. Possible values are `off`, `all`, and `no_top`.
5	`gate2power=off`	Report PMOS with gate connected to ground and NMOS with gate connected to VDD. Possible values are `off` and `on`.

Digitize parameters

6	`vlth=0.2 v`	DC sources with voltage lower than `vlth` carry static 0. Default is `0.2 V`.

© 2003-2014 All Rights Reserved.

7 `vhth=0.8 v` — DC sources with voltage higher than `vhth` carry static 1. Default is `0.8 V`.

8 `pwl_time` — Time for pwl src to be considered as constant vsrc.

9 `rmax=100000000 ohm` — Maximum resistance value where the node is consideredto be connected to the voltage source node.

Filtering parameters

10 `error_limit=10000` — Maximum number of errors reported. Default is `10000`.

Wildcard scoping

11 `inst=[...]` — Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).

12 `xinst=[...]` — Subcircuit instances to be excluded from the check. Default is none.

13 `subckt=[...]` — The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).

14 `xsubckt=[...]` — The instances of the specified subcircuits that are excluded from the check. Default is `none`.

15 `depth=8` — Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

depth 15	gate2power 5	rmax 9	xinst 12
error_limit 10	hotwell 1	subckt 13	xsubckt 14
floatbulk 2	inst 11	vhth 7	

© 2003-2014 All Rights Reserved.

Static HighZ Node Check (static_highz)

Description

Reports the nodes that do not have any possible conducting path to a DC power supply or ground.

The results are written to the static.xml file, which can be viewed in a Web browser.

This check is supported only by XPS.

Definition

```
Name static_highz parameter=value ...
```

Parameters

Design check parameters

1 `node="[...]"` Nodes to which the check is applied. Default is all (`node=*`).

Digitize parameters

2 `vlth=0.2 v` DC sources with voltage lower than `vlth` carry static 0. Default is `0.2 V`.

3 `vhth=0.8 v` DC sources with voltage higher than `vhth` carry static 1. Default is `0.8 V`.

4 `pwl_time`
Time for pwl src to be considered as constant vsrc.

Filtering parameters

5 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

6 `inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
7 `xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
8 `subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
9 `xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
10 `depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static MOSFET Voltage Check (static_mosv)

Description

Reports the MOSFET devices fulfilling the conditional expression on device voltages and device size (w, l).

Supported MOSFET variables are: v(g,s), v(g,d), v(g,b),v(d,s), v(d,b), v(s,b), v(g), v(d), v(s), v(b), l, and w

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !

The results are written to the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_mosv parameter=value ...
```

Parameters

Design check parameters

1	`model="[...]"`	MOSFET device model names to be checked.
2	`cond`	The conditional expression to be fulfilled. Default is `none`.

Digitize parameters

3	`vlth=0.2 v`	DC sources with voltage lower than `vlth` carry static 0. Default is `0.2 V`.
4	`vhth=0.8 v`	DC sources with voltage higher than `vhth` carry static 1. Default is `0.8 V`.
5	`pwl_time`	Time for pwl src to be considered as constant vsrc.

© 2003-2014 All Rights Reserved.

Filtering parameters

6 `error_limit=10000`
: Maximum number of errors reported. Default is `10000`.

Wildcard scoping

7 `inst=[...]`
: Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).

8 `xinst=[...]`
: Subcircuit instances to be excluded from the check. Default is `none`.

9 `subckt=[...]`
: The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).

10 `xsubckt=[...]`
: The instances of the specified subcircuits that are excluded from the check. Default is `none`.

11 `depth=8`
: Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static Forward Bias Bulk Check (static_nmosb)

Description

Reports the NMOS devices with a forward-biased bulk condition.

The results are written to the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_nmosb parameter=value ...
```

Parameters

Design check parameters

1 `model="[...]"` MOSFET device model names to be checked.

Digitize parameters

2 `vlth=0.2 V` DC sources with voltage lower than `vlth` carry static 0. Default is `0.2V`.

3 `vhth=0.8 V` DC sources with voltage higher than `vhth` carry static 1. Default is `0.8V`.

4 `pwl_time`
Time for pwl src to be considered as constant vsrc.

Filtering parameters

5 `error_limit=10000`
Maximum number of errors reported. Default is `10000`.

Wildcard scoping

6 `inst=[...]` Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).

© 2003-2014 All Rights Reserved.

7 `xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
8 `subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
9 `xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
10 `depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static Always Conducting MOSFET Check (static_nmosvgs)

Description

Reports the NMOS devices that are potentially always conducting due to connectivity problems.

The following conditions are checked, and an error is reported if they are fulfilled:
NMOS: min(Vg) > min (Vs/Vd) + abs(vt)

The results are written to the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_nmosvgs parameter=value ...
```

Parameters

Design check parameters

1 `model="[...]"` MOSFET device model names to be checked.

2 `vt=none` MOSFET voltage threshold. Default is `none`.

Digitize parameters

3 `vlth=0.2 V` DC sources with voltage below `vlth` carry static 0. Default is `0.2 V`.

4 `vhth=0.8 V` DC sources with voltage above `vhth` carry static 1. Default is `0.8 V`.

5 `pwl_time`
Time for pwl src to be considered as constant vsrc.

© 2003-2014 All Rights Reserved.

Filtering parameters

6 `error_limit=10000`
: Maximum number of errors reported. Default is `10000`.

Wildcard scoping

7 `inst=[...]`
: Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).

8 `xinst=[...]`
: Subcircuit instances to be excluded from the check. Default is `none`.

9 `subckt=[...]`
: The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).

10 `xsubckt=[...]`
: The instances of the specified subcircuits that are excluded from the check. Default is `none`.

11 `depth=8`
: Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static Forward Bias Bulk Check (static_pmosb)

Description

Reports the PMOS devices with a forward-biased bulk condition.

The results are written to the static.xml file, which can be viewed in a Web browser.

This check is supported only by XPS.

Definition

```
Name static_pmosb parameter=value ...
```

Parameters

Design check parameters

1	`model="[...]"`	MOSFET device model names to be checked.

Digitize parameters

2	`vlth=0.2 V`	DC sources with voltage below vlth carry static 0. Default is `0.2 V`.
3	`vhth=0.8 V`	DC sources with voltage above vhth carry static 1. Default is `0.8 V`.
4	`pwl_time`	Time for pwl src to be considered as constant vsrc.

Filtering parameters

5	`error_limit=10000`	Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

6 `inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
7 `xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
8 `subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
9 `xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
10 `depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static Always Conducting MOSFET Check (static_pmosvgs)

Description

Reports the PMOS devices potentially always conducting due to connectivity problems.

The following conditions are checked, and an error is reported if they are fulfilled:
PMOS: max(Vg) < max (Vs/Vd) - abs(vt)

The results are written to the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_pmosvgs parameter=value ...
```

Parameters

Design check parameters

1	`model="[...]"`	MOSFET device model names to be checked.
2	`vt=0.0 v`	MOSFET voltage threshold. Default is `none`.

Digitize parameters

3	`vlth=0.2 v`	DC sources with voltage below vlth carry static 0. Default is 0.2 V.
4	`vhth=0.8 v`	DC sources with voltage above vhth carry static 1. Default is 0.8 V.
5	`pwl_time`	Time for pwl src to be considered as constant vsrc.

Filtering parameters

6	`error_limit=10000`	Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

7 `inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (inst=*).
8 `xinst=[...]`	Subcircuit instances to be excluded from the check. Default is none.
9 `subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).
10 `xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is none.
11 `depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static Resistor Check (static_resistor)

Description

Reports all resistors within or outside the range of `rmin` and `rmax`. It can also generate a distribution list of all the resistors in a circuit.

If the `type` parameter is set to range, all the resistors outside the range of `rmin` and `rmax` will be reported. If the `type` parameter is set to `print`, all the resistors between `rmin` and `rmax` will be reported. In the report, the resistor names can be sorted by clicking on the `Device name` header in the Web browser.

If the `type` parameter is set to `distr`, a distribution list will be generated for all resistors. There are a maximum of 12 bins: -Inf - 0, 0 - 1m, 1m - 10m, 10m - 0.1, 0.1 - 1, 1 - 10, 10 - 100, 100 - 1k, 1k - 10k, 10k - 100k, 100k - 1Meg , and 1Meg - Inf

However, if two or more consecutive bins are empty, they will merge into one bin, reducing the number of bins. If type is set to `distr`, the parameters `rmin`, `rmax` and `error_limit` are ignored.

The results are reported into the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_resistor parameter=value ...
```

Parameters

Design check parameters

1	`type=print`	Checking types. Possible values are `range`, `distr`, and `print`.
2	`rmin=-1000E9` Ω	Minimum resistor value.
3	`rmax=1000E9` Ω	Maximum resistor value.

© 2003-2014 All Rights Reserved.

Filtering parameters

4 `error_limit=10000`

Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Static Resistor Voltage Check (static_resv)

Description

Reports the resistor devices fulfilling the conditional expression on device voltages.

Supported resistor variables are: v(1,2), v(1), and v(2)

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !

The results are written to the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_resv parameter=value ...
```

Parameters

Design check parameters

1	`cond`	The conditional expression to be fulfilled. Default is none.

Digitize parameters

2	`vlth=0.2 v`	DC sources with voltage lower than `vlth` carry static 0. Default is `0.2 V`.
3	`vhth=0.8 v`	DC sources with voltage higher than `vhth` carry static 1. Default is `0.8 V`.
4	`pwl_time`	Time for pwl src to be considered as constant vsrc.

Filtering parameters

5	`error_limit=10000`	Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

6 `inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
7 `xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
8 `subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
9 `xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
10 `depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

Static Transmission Gate Check (static_tgate)

Description

Reports the transmission gates which cause potential leakage currents between power supplies. These gates can be characterized by their node connectivity, based on the following:

- nodes which connect to the gate and NMOS drain/source terminals, but not to the PMOS drain/source terminals

- nodes which connect to the gate and PMOS drain/source terminals, but not to the NMOS drain/source terminals

The results are written to the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_tgate parameter=value ...
```

Parameters

Design check parameters

1 `node="[...]"` Nodes to which the check is applied. Default is all (`node=*`).

Filtering parameters

2 `error_limit=10000`
Maximum number of errors reported. Default is 10000.

Wildcard scoping

3 `inst=[...]` Subcircuit instances to which the check is applied. Default includes all instances (inst=*).

4 `xinst=[...]` Subcircuit instances to be excluded from the check. Default is none.

© 2003-2014 All Rights Reserved.

5	`subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).
6	`xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is none.
7	`depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.

© 2003-2014 All Rights Reserved.

Static Voltage Domain Device Check (static_voltdomain)

Description

Reports the low voltage MOSFET devices that are wrongly connected to the high voltage domain, or high voltage MOSFET devices wrongly connecting to a low voltage domain.

The results are written to the static.xml file, which can be viewed in a Web browser. This check is supported only by XPS.

Definition

```
Name static_voltdomain parameter=value ...
```

Parameters

Design check parameters

1	`model=[...]`	MOSFET device model names to be checked.

Digitize parameters

2	`vlth=0.2 v`	Low-voltage DC sources with voltage higher than `vlth` are checked.
3	`vhth=0.8 v`	High-voltage DC sources with voltage lower than `vhth` are checked.
4	`pwl_time`	Time for pwl src to be considered as constant vsrc.

Filtering parameters

5	`error_limit=10000`	Maximum number of errors reported. Default is `10000`.

© 2003-2014 All Rights Reserved.

Wildcard scoping

6 `inst=[...]`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
7 `xinst=[...]`	Subcircuit instances to be excluded from the check. Default is `none`.
8 `subckt=[...]`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (`subckt=*`).
9 `xsubckt=[...]`	The instances of the specified subcircuits that are excluded from the check. Default is `none`.
10 `depth=8`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is `8`.

Parameter Index

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

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

A

References

This section gives additional details about the source documents referred to in the text.

- *[antognetti88] Paolo Antognetti, Giuseppe Massobrio. Semiconductor Device Modeling with SPICE. McGraw-Hill, New York, 1988.*
- *[gear71] C. William Gear. Numerical Initial Value Problems in Ordinary Differential Equations. Prentice-Hall, 1971.*
- *[hammerstad80] E. Hammerstad, O. Jensen. "Accurate models for microstrip computer-aided design." IEEE MTT-S 1980 International Microwave Symposium Digest, pages 407-409.*
- *[jansen83] Rolf H. Jansen, Martin Kirschning. "Arguments and an accurate model for the power-current formulation of microstrip characteristic impedance." Arch. Elek. Ubertragung (AEU), vol. 37, 1983, pages 108-112.*
- *[kirschning82] M. Kirschning, R. H. Jansen. "Accurate model for effective dielectric constant of microstrip with validity up to millimetre-wave frequencies." Electronic Letters, vol. 18, no. 6, 18 March 1982, pages 272-273.*
- *[kundert90]Kenneth S. Kundert, Jacob K. White, Alberto Sangiovanni-Vincentelli. Steady-State Methods for Simulating Analog and Microwave Circuits. Kluwer Academic Publishers, 1990.*
- *[nagel75]Laurence W. Nagel. SPICE2: A Computer Program to Simulate Semiconductor Circuits. Ph. D. dissertation, University of California at Berkeley, May 1975. Available through Electronics Research Laboratory Publications, U. C. B., 94720; Memorandum No. UCB/ERL M520.*
- *[quarles89] Thomas L. Quarles. Analysis of Performance and Convergence Issues for Circuit Simulation. Ph. D. dissertation, University of California at Berkeley, April 1989. Extensively documents the Spice3 program. Available through Electronics Research Laboratory Publications, U. C. B., 94720; Memorandum No. UCB/ERL M89/42.*

© 2003-2014 All Rights Reserved.

- *[statz87]Hermann Statz, Paul Newman, Irl W. Smith, Robert A. Pucel, Hermann A. Haus. “GaAs FET device and circuit simulation in SPICE.” IEEE Transactions on Electron Devices, vol. ED-34, no. 2, pages 160-169, February 1987.*
- *[vladimirescu81]A. Vladimirescu, Kaihe Zhang, A. R. Newton, D. O. Pederson, A. Sangiovanni-Vincentelli. SPICE Version 2G User’s Guide, August 1981. Available through Industrial Liaison Program Software Distribution office, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 94720.*
- *[yang82]Ping Yang, Pallab K. Chatterjee. “SPICE modeling for small geometry MOSFET circuits.” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. CAD-1, no. 4, pages 169-182, October 1982.*

© 2003-2014 All Rights Reserved.

Index

Symbols

A

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

B

C

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

D

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

E

F

G

H

© 2003-2014 All Rights Reserved.

I

K

L

M

© 2003-2014 All Rights Reserved.

N

O

© 2003-2014 All Rights Reserved.

P

© 2003-2014 All Rights Reserved.

Q

R

S

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

© 2003-2014 All Rights Reserved.

T

© 2003-2014 All Rights Reserved.

U

V

W

X

© 2003-2014 All Rights Reserved.