cadence™

Virtuoso® Spectre® Circuit Simulator Reference

Product Version 7.2
December 2009

Printed in the United States of America.

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

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

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:

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.

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

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

`[ ]` 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.

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.

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.

Improvements over SPICE

The Spectre circuit simulator has many improvements over SPICE.

Improved Capacity

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

Improved Accuracy

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

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

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

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

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

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

- Superior time-step control algorithm

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

- More accurate simulation techniques

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

- User control of accuracy tolerances

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

Improved Speed

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

- Faster simulation of small circuits

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

- Faster simulation for large circuits

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

Improved Reliability

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

- Improved convergence

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

- Helpful error and warning messages

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

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

- Thorough testing

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

- Benchmark suite

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

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

Improved Models

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

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

Spectre Usability Features and Customer Service

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

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

Analog HDLs

The Spectre circuit simulator works with 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.

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

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

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

RF Capabilities

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

SpectreRF adds four types of analyses to the Spectre simulator. The first is periodic steady-state (PSS) analysis, a large-signal analysis that directly computes the periodic steady-state response of a circuit. With PSS, simulation times are independent of the time constants of the circuit, so PSS can quickly compute the steady-state response of circuits with long time constants, such as high-Q filters and oscillators.

You can also embed a PSS analysis in a sweep loop (referred to as an SPSS analysis in the Cadence analog design environment), which allows you to easily determine harmonic levels as a function of input level or frequency, making it easy to measure compression points, intercept points, and voltage-controlled oscillator (VCO) linearity.

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

analyses are limited to circuits with DC operating points. The periodic small-signal analyses can be applied to circuits with periodic operating points, such as the following:

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

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

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

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

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

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

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

Mixed-Signal Simulation

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

Environments

The Spectre circuit simulator is fully integrated into the Cadence® design framework II for the Cadence analog design environment and also into the Cadence analog workbench design 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.

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:

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.

Note: 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 device or analysis *`name`*. If *`name`* is `all`, the synopses for all components and analyses are given. You can use `-h` as an abbreviation of `-help`.
`-helpsort` *`name`*	Gives a synopsis of the device or analysis *`name`* and sorts all the parameters by name. You can use `-hs` as an abbreviation of `-helpsort`.
`-helpfull` *`name`*	Gives a full synopsis of the component or analysis *`name`*, including parameter types and range limits. You can use `-hf` as an abbreviation of `-helpfull`.

`-helpsortfull` *`name`*	Gives a full synopsis of component or analysis *`name`*, including parameter types and range limits. Sorts all parameters by name. You can use `-hsf` as an abbreviation of `-helpsortfull`.
`-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 all 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`, awb, `sst2`, `fsdb`, `wdf`, `uwi`, or `tr0ascii`.
`+rtsf`	Enables the fast waveform viewing mode for psf output. Requires -f psfbin or -f psfbinf format options.
`-uwifmt` *`name`*	User defined output format. To specify multiple formats use `:` as a delimeter.
`-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.
`+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`†	Restarts the simulation from a checkpoint file, if it exists. You can use `+rec` as an abbreviation of `+recover`.
`-cols` *N*†	Sets screen width in characters to *N*. You can use `-c` as an abbreviation of `-cols`. If not set, the Spectre simulator determines the screen width automatically.
`-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.
`-warn`†	Does not print warning messages.
`-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 may 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 may 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.
`-note`	Does not print notices.
`+info`†	Prints informational messages.
`-info`†	Does not print informational messages.
`+debug`†	Prints debugging messages.
`-debug`†	Does not print debugging messages.
`-slave <`*cmd*`>`	Starts the attached simulator using the command *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.
`-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.
`-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`†	Do not run the Spice netlist reader on the input file.
`+spp`†	Run the Spice netlist reader on the input file. Use `+spp -sppbin` on the command line option to read other spp binaries.
`-sppbin` *`path`*†	Specify the location for SPICE netlist reader. Default provided.
`+sensdata <`*`file`*`>`	Sends the sensitivity analyses data to *`file`*.
`+multithread`	Turns on multithread capability. Spectre automatically detects the number of processors and uses one thread for every CPU core, with a maximum of 4 threads for the baseline Spectre product, and 8 threads in Spectre Turbo mode. You can use `+mt` as an abbreviation of `+multithread`.
`+multithread=`*`N`*	Turns on multithread capability with *`N`* threads. Specify up to 4 threads for the baseline Spectre product, and up to 8 threads in Spectre Turbo mode. You can use `+mt` as an abbreviation of `+multithread`.
`-multithread`	Turns off multithread capability. By default, multithreading is turned off. You can use `-mt` as an abbreviation of `-multithread`.
`-interactive`	Run in the non-interactive mode, that is, process the input file and then return. You can use `-inter` as an abbreviation of `-interactive`.
`+interactive`	Run in the default interactive mode. You can use `+inter` as an abbreviation of `+interactive`.
`+interactive=`*`type`*	Run 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`.
`+mpshost=`*`sessionHost`*	The *`sessionHost`* for an interactive session using MPS.

`-mdlcontrol`	Do not run with the MDL control file. You can use `-mdl` as an abbreviation of `-mdlcontrol`.
`+mdlcontrol`	Runs with the default MDL control file. You can use `-mdl` as an abbreviation of `-mdlcontrol`.
`=mdlcontrol` *`file`*	Specifies the location of the MDL control file to run. You can use `-mdl` as an abbreviation of `-mdlcontrol`.
`-checklimitfile` *`file`*	Writes assert violations to *`file`*. In *`file`*, %C is replaced by the circuit name. You can use `-cl` as an abbreviation of `-checklimitfile`.
`-docheckli mit`	Turns off the checklimit capability. You can use `-docl` as an abbreviation of `-docheckli mit`.
`+docheckli mit`	Turns on the checklimit capability. You can use `+docl` as an abbreviation of `+docheckli mit`.
`+lqtimeout` *`value`*	Turns on license queuing. *`value`* specifies how many seconds to wait for a license. Specifying 0 means wait until license is available. You can use `+lqt` as an abbreviation of `+lqtimeout`.
`+lqsleep` *`value`*	Sleep time between two tries to check out a license when queuing. Setting the value to a positive number will override the default sleep time of 30 seconds. You can use `+lqs` as an abbreviation of `+lqsleep`.
`+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`.

`+lorder`	Spectre will check for a license in the specified order. Use `:` between the license feature names when defining the order. For example, `+lorder Virtuoso_Multi_mode_Simulation:Virtuoso_Spectre` specifies that the token license is checked before `Virtuoso_Spectre`. Spectre can run on the following licenses: `Virtuoso_Spectre`, `Virtuoso_Multi_mode_Simulation` and `32500`.
`-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. You may use `-bc` as an abbreviation of `-bsrccom`.
`-ahdlcom` *`value`*	Determines the Ahdl compiledC flow. Use `0` for fast compilation but potentially slower simulation and `1` for slow compilation but potentially faster simulation. You may use `-ac` as an abbreviation of `-ahdlcom`.
`+parasitics`	Enables reduction of netlist parasitics.
`+turbo`	Enables the Spectre Turbo mode.
`+turbo=`*`value`*	Enables the Spectre Turbo mode and overwrites `errpreset` in all transient analyses to the specified value. Possible values are `liberal`, `moderate`, or `conservative`.

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

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

Default Values

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

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

Default Parameter Values

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

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

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

In this example, the default values for nf, etchc, homotopy, and rawfile are a real number (1.0), the value of a different parameter (etch), an enumerated type (all), and a character string with a percent code and a colon modifier that gives Spectre instructions for creating the output filename ("%C:r.raw").

For more information about percent codes and colon modifiers, see "Description of Spectre Predefined Percent Codes," "Customizing Percent Codes," and "Creating Filenames from Parts of Input Filenames" in the Virtuoso Spectre Circuit Simulator User Guide.

3

Analysis Statements

This chapter discusses the following topics:

AC Analysis (ac)

Description

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

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

Definition

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

Parameters

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

Sweep interval parameters

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

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

Sweep variable parameters

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

State-file parameters

15 `readns`	File that contains estimate of DC solution (nodeset).
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`	Which set of initial conditions to use. Possible values are `none`, `node`, `dev`, or `all`.
19 `readforce`	File that contains initial conditions.
20 `skipdc=no`	Skip the DC analysis. Possible values are `no` or `yes`.

Output parameters

21 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
22 `nestlvl`	Levels of subcircuits to output.
23 `oppoint=no`	Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if operating point is computed in the previous analysis

and is unchanged.
Possible values are no, screen, logfile, or rawfile.

Convergence parameters

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

Annotation parameters

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

26 stats=no — Stats parameter is not supported. Please use annotate.
Possible values are no or yes.

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, or im2ds.

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 devices is calculated.

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

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

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

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

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

When you simulate the same circuit many times, we suggest that you use both the `write` and `readns` parameters and give the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.

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:

`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.

Once 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 following each parameter name indicates where to find the description of that parameter.

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

Alter a Circuit, Component, or Netlist Parameter (alter)

Description

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

Definition

```
Name alter parameter=value ...
```

Parameters

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

Alter Group (altergroup)

Description

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

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

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

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

Definition

```
Name altergroup parameter=value ...
```

Parameters

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

Example:

```
FastCorner altergroup {

    parameters p2=1 p3=p1+2

    model myres resistor r1=1e3 af=1

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

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

}
```

The list of public devices supported by `altergroup`:

```
assertbhtbht0bjt
bjt301bjt500bjt500tbjt503
bjt504bjt504tbjt3500bjt3500t
bjtd504bjtd504tbjtd3500bjtd3500t
bsim1bsim2bsim3bsim3v3
bsim4bsimsoibulkmgcapacitor
cccsccvsdio500diode
ekv ekv3ekv3_nqsekv3_r4
ekv3_rfekv3_sfracpolegaas
hbt hisim2hisim_ldmoshvmos
inductorintcapisourcejfet
juncapjuncap200juncap_eldoldmos
mos1mos2mos3mos30
mos40mos40tmos705mos902
mos903mos1000mos1100mos1100e
mos1101emos1101etmos1102emos1102et
mos2001mos2001emos2001etmos2001t
mos2002mos2002emos2002etmos2002t
mos3002mos3100mos3100tmos11010
mos11010tmos11011mos11011tmos11020
mos11020tmos11021mos11021tmosvar
mslinemutual_inductornodcappattern
pcccspccvsphy_resprint
psitftpsp102epsp1020psp1021
pspnqs102epspnqs1020pspnqs1021pvccs
pvcvsr2r3rdiff
resistorrlck_matrixsoimgspmos
tlinetom2tom3vbic
vccsvcvsvsource
```

The list of public devices not supported by `altergroup`:

```
a2d atftb3soipdbit
cktromcored2adelay
ibis_bufferiprobemos0mos15
mtlinenportportrelay
scccssccvssvccssvcvs
switchtransformerwindingzcccs
zccvszvccszvcvs
```

Check Parameter Values (check)

Description

The `check` analysis checks the values of component parameters to assure they are reasonable. This analysis reduces the cost of data entry errors. Various filters specify which

parameters are checked. You can perform checks on input, output, or operating-point parameters. Use this analysis in conjunction with the `+param` command line argument, which specifies a file that contains component parameter soft limits.

Definition

```
Name check parameter=value ...
```

Parameters

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

Checklimit Analysis (checklimit)

Description

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

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

Definition

```
Name checklimit parameter=value ...
```

Parameters

1	`enable=[...]`	Array of checks to be enabled. Default is all.
2	`disable=[...]`	Array of checks to be disabled. Default is none.
3	`severity`	Severity of the checks. Possible values are `none`, `notice`, `warning`, `error`, or `fatal`.
4	`title`	Analysis title.

5 `checkallasserts=yes`
If all checks should be enabled or disabled. CheckAllAsserts=no will disable all checks.
Possible values are `no` or `yes`.

Boundary parameters

6 `boundary_type=time`
Boundary type.
Possible values are `time` or `sweep`.

7 `start` Start time or sweep boundary of the checks.

8 `stop` Stop time or sweep boundary of the checks.

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

Parameter Index

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

```
boundary_type  6    disable  2          start  7

check_windows  9    enable  1           stop  8

checkallasserts     severity  3         title  4
5
```

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 or yes.. Possible values are `no` or `yes`.

DC Analysis (dc)

Description

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

Definition

```
Name dc parameter=value ...
```

Parameters

Sweep interval parameters

1 `start=0`	Start sweep limit.
2 `stop`	Stop sweep limit.
3 `center`	Center of sweep.
4 `span=0`	Sweep limit span.
5 `step`	Step size, linear sweep.
6 `lin=50`	Number of steps, linear sweep.
7 `dec`	Points per decade.
8 `log=50`	Number of steps, log sweep.
9 `values=[...]`	Array of sweep values.
10 `hysteresis=no`	Perform dc hysteresis sweep. When on, a reverse sweep will automatically be added to the dc sweep. Possible values are `no` or `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`	What should be used to force values for DC. Uses the values from the device and node ICs. Possible values are `none`, `node`, `dev`, or `all`.
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.

Output parameters

19 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
20 `nestlvl`	Levels of subcircuits to output.
21 `print=no`	Print node voltages. Possible values are `no` or `yes`.
22 `oppoint=no`	Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if sweep parameter `param` is set. Possible values are `no`, `screen`, `logfile`, or `rawfile`.
23 `check=yes`	Check operating point parameters against soft limits. Possible values are `no` or `yes`.

Convergence parameters

24 `homotopy=all`	Method used when no convergence on initial attempt of DC analysis. Possible values are `none`, `gmin`, `source`, `dptran`, `ptran`, `arclength`, or `all`.
25 `restart=yes`	Restart from scratch if any condition has changed. If not, use the previous solution as initial guess. Possible values are `no` or `yes`.
26 `maxiters=150`	Maximum number of iterations.
27 `maxsteps=10000`	Maximum number of steps used in homotopy method.

Annotation parameters

28 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, `steps`, `iters`, `detailed`, or `rejects`.
29 `title`	Analysis title.

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

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

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

When you simulate the same circuit many times, we suggest that you use both the `write` and `readns` parameters and give the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.

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

`force=none`: Any initial condition specifiers are ignored.

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

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

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

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

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

Parameter Index

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

DC Device Matching Analysis (dcmatch)

Description

The DCMATCH analysis performs DC device mis-matching analysis for a given output. It computes the deviation in the DC operating point of the circuit caused by mismatch in the devices. Users need to specify mismatch parameters in their model cards for each device

contributing to the deviation. The analysis uses the device mismatch models to construct equivalent mismatch current sources to all the devices that have mismatch modeled. These current sources will have zero mean and some variance. The variance of the current sources are computed according to mismatch models. It then computes the 3-sigma variance of dc voltages or curents at user specified outputs due to the mismatch current sources. The simulation results displays the devices rank ordered by their contributions to the outputs. In addition, for mosfet devices, it displays threshold voltage mismatch, current factor mismatch, gate voltage mismatch, and drain current mismatch. For bipolar devices, it displays base-emitter junction voltage mismatch. For resistors, it displays resistor mismatches.

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

The DCMATCH analysis is available for BJT, BSIM3v3, BSIM4, BSIMSOI, BSIM5, EKV, VBIC, resistor 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`, or `rawfile`.
3	`file`	File name for results to be printed if where=file is used.

Probe parameters

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

Port parameters

5	`portv`	Voltage across this probe port is output of the analysis.
6	`porti`	Current through this probe port is output of the analysis.

Sweep interval parameters

7 `start=0`	Start sweep limit.
8 `stop`	Stop sweep limit.
9 `center`	Center of sweep.
10 `span=0`	Sweep limit span.
11 `step`	Step size, linear sweep.
12 `lin=50`	Number of steps, linear sweep.
13 `dec`	Points per decade.
14 `log=50`	Number of steps, log sweep.
15 `values=[...]`	Array of sweep values.

Sweep variable parameters

16 `dev`	Device instance whose parameter value is to be swept.
17 `mod`	Model whose parameter value is to be swept.
18 `param`	Name of parameter to sweep.

State-file parameters

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

Output parameters

20 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
21 `nestlvl`	Levels of subcircuits to output.
22 `oppoint=no`	Should operating point information be computed, and if so, where should it be sent. Operating point information would not

be output if (1) operating point is computed in the previous analysis and is unchanged, or (2) sweep parameter `param` is set.
Possible values are `no`, `screen`, `logfile`, or `rawfile`.

Convergence parameters

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

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

Annotation parameters

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

26 `stats=no` — Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

27 `title` — Analysis title.

Miscellaneous parameters

28 `version=0` — If 1, using Bsim short channel mismatch equation for bsim3 or bsim4 devices.

The dcmatch analysis will find a dc operating point first. If the dc analysis fails, then the dcmatch analysis will fail also. The parameter mth is a threshold value relative to maximum contribution. Any device contribution less than (mth * maximum) will not be 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 users to select a current associated with a specific device given in oprobe as an output. This device, however, has to have its terminal currents as network variables,i.e. the device has to be an inductor, a vsource, a switch, a tline, a controlled voltage source, an iprobe, or other type of device which has current solution. Further, for inductor, vsource, switch, controlled voltage source and iprobe, porti can only be set to one, since these devices are two terminal devices (one port); and for tline porti can be set to one or two, since it is a four terminal device (two ports).

Parameter Index

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

Envelope Following Analysis (envlp)

Description

This analysis computes the envelope response of a circuit. The user specifies the analysis `clockname` or `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 otherwise given.

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

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. Thus, autonomous circuits generate non-constant waveforms even though they are not driven by a time-varying stimulus. Driven circuits require some time-varying stimulus to generate a time-varying response. The most common example of an autonomous circuit is an RF mixer with local oscillator.

When applied to autonomous circuits, Envelope following analysis requires the user 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 specified harmonic of the clock fundamental.

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, will overwrite 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 envlp starts shooting.

7 `period (s)` Period of the clock fundamental. If set, `clockname` can be ignored. It is 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 derived from `errpreset`.

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

12 `fixstepsize=no` Use this option to fix envelop step size for speeding up envlp analysis.
Possible values are `no` or `yes`.

13 `stepsize=4` — The number of cycles skipped for each step when fixstepsize is yes.

Initial-condition parameters

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

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

16 `readic` — File that contains initial transient condition.

Convergence parameters

17 `readns` — File that contains estimate of initial DC solution.

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

State-file parameters

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

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

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

Envelope Integration method parameters

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

Integration method parameters

23 `method=gear2only`	Inner transient integration method. Possible values are `euler`, `trap`, `traponly`, `gear2`, `gear2only`, or `trapgear2`.
24 `oscic=default`	Oscillator IC method. It determines how the starting values for the oscillator are determined. Possible values are `default` or `lin`.

Accuracy parameters

25 `errpreset=moderate`	Selects a reasonable collection of parameter settings. Possible values are `liberal`, `moderate` or `conservative`.
26 `relref`	Reference used for the relative convergence criteria. Default derived from `errpreset`. Possible values are `pointlocal`, `alllocal`, `sigglobal`, or `allglobal`.
27 `lteratio`	Ratio used to compute LTE tolerances from Newton tolerance. Default derived from `errpreset`.
28 `itres=1e-2`	Relative tolerance for linear solver.
29 `lnsolver=gmres`	Linear solver. Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.
30 `inexactNewton=no`	Inexact Newton method. Possible values are `no` or `yes`.
31 `steadyratio`	Ratio used to compute steady state tolerances from LTE tolerance. Default derived from `errpreset`.
32 `envlteratio`	Ratio used to compute envelope LTE tolerances. Default derived from `errpreset`.

Annotation parameters

33 `stats=no`	Stats parameter is not supported. Please use annotate. Possible values are `no` or `yes`.

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

Output parameters

36 `harms`	If `harmonicbalance` is no, it is the number of clock harmonics to output and the default value is 1. If `harmonicbalance` is yes, it is the maxharm of the clock fundamental and the default value is 3.
37 `harmsvec=[...]`	Array of desired output clock harmonics. Alternate form of `harms` that allows selection of specific harmonics. For multi-carrier envelope, each group of elements with the size equal to that of `funds` is a selection of specific harmonic combinations of fundamental frequencies.
38 `outputtype=both`	Output type. Possible values are `both`, `envelope` or `spectrum`.
39 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
40 `nestlvl`	Levels of subcircuits to output.
41 `compression=no`	Do data compression on output. See full description bellow. Possible values are `no` or `yes`.
42 `strobeperiod (s)`	The output strobe interval (in seconds of envelope following time). The actual strobe interval is rounded to an integer multiple of the clock period.

Newton parameters

43 `maxiters=5`	Maximum number of Newton iterations per transient integration time step.
44 `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.

45 `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

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

47 `fmspeedup=0` The level to speed up the envlp analysis for frequency modulated signal, default is 0 for standard envlp following, 1 for predescribed fmmod sources speedup.

48 `saveinit=no` If set, the waveforms for the initial transient(tstab) before envlp are saved.
Possible values are `no` or `yes`.

Harmonic Balance Envelope parameters

49 `funds=[...]` Array of fundamental frequency names for fundamentals to use for Harmonic Balance Envelope.

50 `maxharms=[...]` Array of number of harmonics of each fundamental to consider for each fundamental for Harmonic Balance Envelope.

51 `freqdivide` Large signal frequency division.

52 `harmonicbalance=no`
Use Harmonic Balance Envelope.
Possible values are `no` or `yes`.

53 `flexbalance=no` The same parameter as `harmonicbalance`.
Possible values are `no` or `yes`.

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

55 `oversample=[...]` Array of over sample factors for each tone for Harmonic Balance Envelope.

Tstab save/restart parameters

56 `saveperiod`	save the tran analysis periodically on the stab time.
57 `saveclock=1800 s`	save the tran analysis periodically on the wall clock time.
58 `savetime=[...]`	save the analysis states into files on the specified time points.
59 `savefile`	save the analysis states into the specified file.
60 `recover`	specify the file to be restored.

AMS-envlp co-sim parameters

61 `resetenv=no`	Use this option to reset envelop data after D2A/A2D events for AMS-envlp co-simulation.. Possible values are `no` or `yes`.
62 `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` or `yes`.
63 `trancycles=5`	The number of transient cycles for AMS-envlp cosimulation, this is the cycles around D2A/A2D events time point, default value is 5..

envlp-PAC parameters

64 `pacnames=[...]`	Names of pac, pnoise, psp, or pxf analyses to be performed at each time point in the `pactimes` array. (not for AMS).
65 `pactimes=[...] s`	Times when analyses specified in `pacname` array are performed. (not for AMS).

If `period` or `fund` are 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 frequencies are found, the greatest common factor of these frequencies is used as the clock frequency.

The maximum envelope step size is affected by many parameters. It can be directly limited by `envmaxstep`. It is also limited by `modulationbw`. The user gives an estimate of the modulation bandwidth. The simulator will put at least eight points within the modulation

period. The user is recommended to use strobeperiod to get equally spaced envelope points, to improve the noise floor in power spectrum density computation.

The `harms` and `harmsvec` parameters affect the simulation time in a insignificant way. The spectrum is calculated for all the specified harmonics for all sampled integration cycles as the envelope following analysis marches on. For each harmonic, a file is generated. If `harmonicbalance` is no, `harms` is set to 1 or 2 typically and high order harmonics will not be accurate.

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

For conservative autonomous envlp, default values for method and envmethod set to traponly to avoid numerical damping of oscillator.

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

Parameter defaults as a function of `errpreset`

---

errpreset	maxstep	envmaxstep	reltol	relref	steadyratio	envlteratio
liberal	T/20	Interval/10	0.01	sigglobal	0.1	0.35
moderate	T/20	Interval/25	0.001	sigglobal	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, since every signal has the same x value. If `compression=yes`, Spectre writes data to the output file only if the signal value changes by at least 2*the convergence criteria. In order to save data for each signal independently, x axis information corresponding to each signal must be saved. If the signals stay at constant values for large periods of the simulation time, setting `compression=yes` results in a smaller output data file. If the signals in your circuit move around a lot, setting `compression=yes` results in a larger output data file.

Parameter Index

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

Circuit Information (info)

Description

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

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

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

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

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

Definition

```
Name info parameter=value ...
```

Parameters

1	`what=oppoint`	What parameters should be printed. Possible values are `none`, `inst`, `models`, `input`, `output`, `nodes`, `all`, `terminals`, `oppoint`, `captab`, `parameters`, `primitives`, `subckts`, `assert`, `allparameters`, `netlist`, `options`, or `dumpall`.
2	`where=logfile`	Where parameters should be printed. Possible values are `nowhere`, `screen`, `file`, `logfile`, or `rawfile`.

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

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

5 `nestlvl`
Levels of subcircuits to output.

6 `extremes=yes`
Print minimum and maximum values.
Possible values are `no`, `yes` or `only`.

7 `title`
Analysis title.

8 `descriptions=no`
Print descriptions.
Possible values are `no` or `yes`.

Captab parameters

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

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

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

Parameter Index

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

Monte Carlo Analysis (montecarlo)

Description

The `montecarlo` analysis is a swept analysis with associated child analyses similar to the sweep analysis (see `spectre -h sweep`.) The Monte Carlo analysis refers to "statistics blocks" where statistical distributions and correlations of netlist parameters are specified. (Detailed information on statistics blocks is given below.) For each iteration of the Monte Carlo analysis, new pseudo-random values are generated for the specified netlist parameters (according to their specified distributions) and the list of child analyses are then executed.

Expressions are associated with the child analyses. These expressions, which are constructed as scalar calculator expressions by the user during Monte Carlo analysis set up, can be used to measure circuit metrics, such as the slew-rate of an op-amp. During a Monte Carlo analysis, these expression results will vary as the netlist parameters vary for each Monte Carlo iteration. The Monte Carlo analysis therefore becomes a tool that allows you to examine and predict circuit performance variations, which affect yield.

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

perform nominal run if requested

if any errors in nominal run then stop

foreach Monte Carlo iteration {

  if process variations specified then

    apply `process` variation to parameters

  if mismatch variations specified then

    foreach subcircuit instance {

      apply `mismatch` variation to parameters

    }

  foreach child analysis {

```
   run child analysis

    evaluate expressions

 }

}
```

Definition

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

Parameters

Analysis parameters

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

Saving Process Parameters

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

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

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

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

Flags

14 `donominal=yes` Whether or not to perform nominal run.
Possible values are `no` or `yes`.

15 `addnominalresults=no`
Whether or not to add nominal run results to MC run results.
Possible values are `no` or `yes`.

16 `paramdumpmode=no` Whether or not to full dump process/mismatch parameters information.
Possible values are `no` or `yes`.

17 `appendsd=no` Whether or not to append scalar data.
Possible values are `no` or `yes`.

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

Annotation parameters

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

Detailed Description and Examples:

numruns:(default=100)

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

seed:(no default)

seed for the random number generator. By always specifying the same seed, you can reproduce a previous experiment. If you do not specify a seed, then each time you run the analysis, you will get different results i.e. a different stream of pseudo-random numbers will be generated.

scalarfile="filename"

This parameter allows an ASCII file to be specified in which scalar data (results of expressions that resolve to scalar values) will be written. The data from this file can be read and plotted in histograms by ADE. For each iteration of each Monte Carlo child analyses, Spectre (through Artil) will write a line to this ASCII file which contains scalar data (one scalar expression per column e.g. slewrate or bandwidth.) The default name for this file will be of the form name.mcdata, where name is the name of the Monte Carlo analysis instance. This file contains only the matrix of numeric values. ADE Monte Carlo users 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 will merge the values of the statistically varying process parameters into this file containing 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, the user can see correlations between process parameter variations and circuit performances variations. This data merging will take place whenever the scalarfile and processscalarfile (see below) are written in the same directory.

paramfile="filename"

This file contains the titles, sweep variable values and the full expression for each of the columns in the scalarfile. ADE Monte Carlo users will be more familiar with the term `mcparam`

file for the paramfile.This file will be created in the psf directory by default, unless you specify some path information in the filename.

processscalarfile="filename"

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

processparamfile="filename"

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

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

firstrun:(default=1)

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

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

Whether to apply process (batch-to-batch) variations only, or mismatch (per-instance) variations only, or both together. This assumes that you have specified appropriate statistical

distributions in the statistics block. You cannot request that mismatch variations be applied unless you have specified mismatch statistics in the statistics block. You cannot request that process variations be applied unless you have specified process statistics in the statistics block. More details on statistics blocks are given below.

saveprocessvec=[rshsp TOX ...]

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

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

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

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

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

This parameter controls whether or not Spectre should add a nominal run results after the Monte Carlo run results into the data files.

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

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

savefamilyplots={yes,no}.

If yes, a data file (e.g. psf) is saved for each analysis for each Monte Carlo iteration, in addition to the expressions scalar results which are saved to the ASCII scalar data file at the end of each iteration. Saving the full data files between runs enables the cloud plotting feature (overlaid waveforms) in ADE. It also enables the user 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 number of saved quantities to a minimum. If this parameter is set to no, then data files are overwritten by each Monte Carlo iteration.

annotate={no,title,sweep,status}

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

Examples:

```
// do a Monte Carlo analysis, with process variations only
// useful for looking at absolute performance spreads
mc1 montecarlo variations=process seed=1234 numruns=200 {
  dcop1 dc      // a child analysis
  tran1 tran start=0 stop=1u    // another child analysis
  // expression calculations are sent to the scalardata file
  export slewrate=oceanEval("slewRate(v("vout"),10n,t,30n,t,10,90 )")
}
// do a Monte Carlo analysis, with mismatch variations only
// useful for detecting spreads in differential circuit
// applications, etc. Do not perform a nominal run.
mc2 montecarlo donominal=no variations=mismatch seed=1234 numruns=200 {
  dcop2 dc
  tran2 tran start=0 stop=1u
  export slewrate=oceanEval("slewRate(v("vout"),10n,t,30n,t,10,90 )")
}
// do both together...
mc3 montecarlo saveprocessparams=yes variations=all numruns=200 {
  dcop3 dc
  tran3 tran start=0 stop=1u
```

```
  export slewrate=oceanEval("slewRate(v("vout"),10n,t,30n,t,10,90 )")
}
```

Specifying Parameter Distributions using Statistics Blocks:

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

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

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

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

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

// such as sheet resistance and mismatch factors

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

// define statistical variations, to be used

// with a MonteCarlo analysis.
```

```
statistics {
    process {   // process: generate random number once per MC run
        vary rshsp dist=gauss std=12 percent=yes
        vary rshpi dist=gauss std=rshpi_std // rshpi_std is a parameter
        vary xxx dist=lnorm std=12
        vary uuu dist=unif N=10 percent=yes
    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 <type> keywords `gauss`, `lnorm` and `unif` respectively. For both the `gauss` and the `lnorm` distributions, you specify a standard deviation using the `std` keyword.

Gaussian Distribution:

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

Lognormal Distribution:

The lognormal distribution is denoted by

log(x) = 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 above, 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)

such that the mean value is the nominal value of the parameter x, and the parameter is varied about the mean with a range of +/- N. The standard deviation is not specified for the uniform distribution, but its value can be calculated from the formula: 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)) i.e. Normal (200, 24). It is not advised that you use `percent=yes` with the lognormal distribution.

Process and Mismatch Variations:

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

Correlation Statements:

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

Process Parameter Correlation:

The syntax of the process parameter correlation statement is:

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 will be randomly generated according to the specified distributions and correlations.

Mismatch Correlation (Matched Devices):

The syntax of the instance or mismatch correlation statement is:

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 (i.e. matched) and give the correlation coefficient. You can optionally specify exactly which parameters are to be correlated by giving a list of parameters (each of which must have had distributions specified for it in a mismatch block), or specify no parameter list, in which case all parameters with

mismatch statistics specified are correlated with the given correlation coefficient. The correlation coefficients are specified in the <value> field and must be between +/- 1.0.

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

Truncation Factor:

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

truncate tr=<value>.

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

NOTE(2): Parameter correlations can be affected by using small truncation factors.

NOTE(3): There is different truncate separately for process and mismatch block. If a truncate for process or mismatch block is not given, it will be set to the value of the truncate in statistic block.

Parameter Index

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

```
dut  9              processparamfile    saveprocessvec
                    12                  13
```

Noise Analysis (noise)

Description

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

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

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

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

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

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

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

Definition

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

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

Parameters

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

Sweep interval parameters

2	`start=0`	Start sweep limit.
3	`stop`	Stop sweep limit.

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

Sweep variable parameters

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

Probe parameters

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

State-file parameters

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

Output parameters

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

Convergence parameters

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

Annotation parameters

27 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
28 `stats=no`	Stats parameter is not supported. Please use annotate. Possible values are `no` or `yes`.
29 `title`	Analysis title.

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of

steps or the size of each step. You can give a step size parameter (`step`, `lin`, `log`, `dec`) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. All frequencies are in Hertz.

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

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

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

When you simulate the same circuit many times, we suggest that you use both the `write` and `readns` parameters and give the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.

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:

`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.

Once 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 following each parameter name indicates where to find the description of that parameter.

Immediate Set Options (options)

Description

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

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

In many cases a particular option can be controlled by either a command line or netlist specification. In the situation where both are given, 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=1e-06 V`	Voltage absolute tolerance convergence criterion.
4	`iabstol=1e-12 A`	Current absolute tolerance convergence criterion.

Temperature parameters

5	`temp=27 C`	Temperature.
6	`tnom=27 C`	Default component parameter measurement temperature.
7	`tempeffects=all`	Temperature effect selector. If tempeffect = vt, only thermal voltage varies with temperature; if tempeffect = tc, parameters that start with tc are active and thermal voltage is dependent on temperature; and if tempeffect = all, all built-in temperature models are enabled. Possible values are `vt`, `tc` or `all`.

Output parameters

8	`save=selected`	Signals to output. For more information see below. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
9	`nestlvl=∞`	Levels of subcircuits to output.

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

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

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

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

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

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

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

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

18 `ahdldomainerror=warning`
AHDL domain error handle manner selector. If ahdldomainerror = error, treat wrong AHDL domain input as error; If ahdldomainerror = warning, treat wrong AHDL domain input as warning; If ahdldomainerror = none, ignore AHDL domain error.
Possible values are `error`, `warning`, `none`, `erroriter`, or `warniter`.

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

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

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

22 `uwifmt` User defined output format. To specify multiple formats use `:` as a delimeter.

23 `uwilib` Absolute path to the user defined output format library. This option is used together with `uwifmt`. Use `:` to specify more than one library.

Convergence parameters

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

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

26 `gmethod=dev` Stamp gdev, gnode or both in the homotopy methods. See below for more information.
Possible values are `dev`, `node` or `both`.

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

Multithreading parameters

28 `multithread=off` This option turns on/off multithread capability. When multithreading is turned on but the number of threads (`nthreads`) is not specified, Spectre will automatically detect the number of processors and use one thread for every CPU core, with a maximum of 4 threads for the baseline Spectre

product, and 8 threads in Spectre Turbo mode.
(See important note below about using multithreading).
Possible values are `off` or `on`.

29 `nthreads` — Specifies the number of threads for multithreading. Specify up to 4 threads for the baseline Spectre product, and up to 8 threads in Spectre Turbo mode.

Component parameters

30 `scalem=1` — Model scaling factor.

31 `scale=1` — Device instance scaling factor.

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

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

34 `macromodels=no` — Circuit contains macromodels; sometimes helps performance.
Possible values are `no` or `yes`.

35 `auto_minductor=no`
Automatic insertion of missing mutual inductor coupling, for more information see below.
Possible values are `no` or `yes`.

Error-checking parameters

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

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

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

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

40 `checklimitfile` File to which assert violations will be written to.

41 `docheckli mit=yes` Check asserts in the netlist.
Possible values are `no` or `yes`.

42 `checklimitdest=file`
Destionation(s) where violations will be written to.
Possible values are `file`, `psf` or `both`.

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

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

Resistance parameters

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

46 `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`.

47 `rforce=1 Ω` Resistance used when forcing nodesets and node-based initial conditions.

48 `rthresh=0.001 Ω` All instance resistors that have a smaller resistance 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 can not be changed afterwards, so there is no use to alter the value of rthresh. You have to start a new run if you want a different rthresh for your circuit.

49 `rclamp (Ω)` — When rclamp=value is given, all instance resistors with R<value are clamped to value.

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

Quantity parameters

51 `value="V"` — Default value quantity.

52 `flow="I"` — Default flow quantity.

53 `quantities=no` — Print quantities. If quantities=min, spectre will print out all defined quantities; if quantities=full, spectre will also print a list of nodes and their quantities.
Possible values are `no`, `min` or `full`.

Annotation parameters

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

55 `inventory=detailed` — Print summary of components used.
Possible values are `no`, `brief` or `detailed`.

56 `narrate=yes` — Narrate the simulation.
Possible values are `no` or `yes`.

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

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

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

60 `maxnotes=5` — Maximum number of times any notice will be issued per analysis. Note that this option has no effect on notices issued as part of parsing the netlist. Please use the -maxnotes command line option to control the number of all notices issued.

61 `warn=yes`	Give warning messages. Possible values are `no` or `yes`.
62 `maxwarns=5`	Maximum number of times any warning message will be issued per analysis. Note that this option has no effect on warnings issued as part of parsing the netlist. Please use the -maxwarns command line option to control the number of all warnings issued.
63 `maxwarnstologfile=5`	Maximum number of times any warning message will be printed to the log file per analysis. Note that this option has no effect on warnings printed as part of parsing the netlist. Please use the -maxwarnstolog command line option to control the number of all warnings printed to the log file.
64 `maxnotestologfile=5`	Maximum number of times any notice message will be printed to the log file per analysis. Note that this option has no effect on notices printed as part of parsing the netlist. Please use the -maxnotestolog command line option to control the number of all notices printed to the log file.
65 `error=yes`	Give error messages. Possible values are `no` or `yes`.
66 `digits=5`	Number of digits used when printing numbers.
67 `measdgt=0`	Number of decimal digits in floating point numbers in measurement output in mt0 fortmat.
68 `ingold=sci`	
69 `notation=eng`	When printing real numbers to the screen, what notation should be used. Possible values are `eng`, `sci` or `float`.
70 `cols=80`	Width of screen in characters.
71 `colslog=80`	Width of log-file in characters.
72 `title`	Circuit title.

Matrix parameters

73 `pivotdc=no`	Use numeric pivoting on every iteration of DC analysis. Possible values are `no` or `yes`.
74 `pivrel=0.001`	Relative pivot threshold.
75 `pivabs=0`	Absolute pivot threshold.
76 `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 are: designs with very large number of small resistors or large number of behavioral instances containing voltage based equations.. Possible values are `partial` or `full`.
77 `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`.
78 `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`.
79 `matrixtype=flat`	Matrix type. If matrix = flat, use flat matrix; If matrix = bbd, use BBD matrix. Possible values are `flat` or `bbd`.
80 `numparts=1`	Number partitions.
81 `parttype=1`	Partitioner Type.

Miscellaneous parameters

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

83 `param_topchange=no`
: If yes, Spectre will support parametric topology change which is caused by device internal node changes when device, model or netlist parameter value is altered..
Possible values are `no` or `yes`.

84 `redefinedparams=error`
: Allow redefined parameters in netlist. Value of the last entry will be used for simulation..
Possible values are `error`, `ignore`, `warning`, or `warn`.

Sensitivity parameters

85 `sensfile`
: Output sensitivity data file name.

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

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

88 `sensfileonly=no`
: Enable or disable raw output of sens. results.
Possible values are `no` or `yes`.

89 `paramrangefile`
: Parameter range file.

Performance parameters

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

91 `verilogalang=relax`
: AHDL Verilog-A language syntax check mode selector. If verilogalang = strict, archaic syntax input as error during verilog-A parsing; If verilogalang = relax, archaic syntax input as warning

during verilog-A parsing.
Possible values are `strict` or `relax`.

Model parameters

`92 ignore_unsupported_altergroup_constructs=no`
If set to yes, it means ignore statistics block in altergroup.
Possible values are `no` or `yes`.

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

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

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

senstype parameter:

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

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

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

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

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

$$\frac{dF}{} \quad \frac{dF}{}$$

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

and

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

When both F and p are zero, the 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.

Use topcheck=fixall to attach gmin to all types of isolated nodes. Including the ones found 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 will not be done.

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 note about using multithreading:

Currently, multithreading is only available for devices evaluation for BSIM3v3, and BSIM4. Multithreading does not work with table model. If there is an instance of a primitive using table model, multithreading would not be applied to all instances of that primitive.

Multithreading can be turned on/off by command line option, or by the multithread parameter in the options statement from the input file. If both options are specified, command line option will make the final determination on the number of threads to use.

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

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

Spectre device multithreading on hyperthreading enabled system:

On hyperthreading enabled system, Spectre device multithreading 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 multiprocessors system with hyperthreading enabled, Spectre device multithreading would allow one extra thread for each additional physical processor. How the device evaluation threads distribute to the processors is controlled by the OS (operating system). Multithreading performance depends on Spectre, but also on how well OS manages multi-threads and multi-processes.

gmethod:

The option `gmethod` controls how conductance is stamped in the homotopy methods. If `gmethod=gnode` the conductance is added from node to ground. In case of `gmethod=dev` the conductance is stamped in the devices. If `gmethod=both`, the stamping is done in the devices as well as from every node to ground.

auto_minductor:

When the option is set to `yes` Spectre 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.

Example: Given two mutual inductors K1 and K2

K1 mutual_inductor coupling=.65 ind1=L1 ind2=L2

K2 mutual_inductor coupling=.65 ind1=L2 ind2=L3

Spectre automatically inserts coupling between L1 and L3 if missing and the coupling coefficient is 0.65*0.65=0.4225.

save option:

If `save=nooutput` is specified Spectre will not output any simulation results other than measurements. For `save=nooutput` to have effect it needs to be defined globally.

Parameter Index

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

Periodic AC Analysis (pac)

Description

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

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

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

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

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

Definition

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

Parameters

Sweep interval parameters

1 `start=0` Start sweep limit.

2 `stop` Stop sweep limit.

3 `center` Center of sweep.

4 `span=0` Sweep limit span.

5 `step` Step size, linear sweep.

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

7 `dec` Points per decade.

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

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

10 `sweeptype=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` or `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` or `sampled`.

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

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

15 `crossingdirection=all`
Specifies for which transitions to do the sampling.
Possible values are `all`, `rise`, `fall`, or `ignore`.

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

17 `sampledelay=[...]` Sampled measurement is done with delay from the time of the crossing event on the conrol signal or probe. The first value corresponds to the rising edge and the second to the falling transition.

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

Output parameters

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

20 `maxsideband=0` An alternative to the `sidebands` array specification, which automatically generates the array: [-maxsideband ... 0 ... +maxsideband]. It is ignored in HB small signal when its larger than the harms/maxharms of large signal.

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

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

23 `nestlvl` Levels of subcircuits to output.

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

Convergence parameters

25 `tolerance` Relative tolerance for linear solver, default value is 1.0e-9 for shooting-based solver; 1.0e-6 for driven and 1.0e-4 for autonomous for flexbalance-based solver.

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

27 `solver=turbo` Solver type.
Possible values are `std` or `turbo`.

28 `oscsolver=turbo` Oscillator solver type. Suggest to use ira for huge circuit.
Possible values are `std`, `turbo` or `ira`.

29 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.

30 `resgmrescycle=short`
restarted gmres cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

Annotation parameters

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

32 `stats=no` Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

33 `title` Analysis title.

Modulation conversion parameters

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

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

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

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

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

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

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

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

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

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

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

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

46 `rfmag=0` RF source magnitude.

47 `rfdbm=0` RF source dBm.

48 `rf1_src="NULL"` Array of RF1 source names for IP3/IP2/IM2.

49 `rf2_src="NULL"` Array of RF2 source names for IP3/IP2/IM2.

You can select the set of periodic small-signal output frequencies of interest by setting either the `maxsideband` or the `sidebands` parameters. For a given set of n integer numbers representing the sidebands K1, K2, ... Kn, the output frequency at each sideband is computed as f(out)= f(in) + 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. 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.

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

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

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

Parameter Index

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

crossingdirection 15
maxsideband 20
resgmrescycle 30
step 5
dec 7
modsource 38
rf1_src 48
stop 2
extrasampletimepoints 18
modulated 34
rf2_src 49
sweeptype 10
fim_out 41
moduppersideband 37
rfdbm 47
thresholdvalue 14
flin_out 40
nestlvl 23
rfmag 46
title 33
freqaxis 21
oscsolver 28
sampledelay 17
tolerance 25
gear_order 26
out1 42
sampleprobe 13
values 9
inmodharmnum 35
out2 43
save 22
lin 6
outmodharmvec 36
sidebands 19
lnsolver 29
outputperiod 24
solver 27

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 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, 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 with all signals activated is run. At last, 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.

As 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). As one consequence, the efficiency of the algorithm depends significantly on the number of harmonics required to model the responses of moderate fundamentals. As another consequence, the number of harmonics of the large fundamental does not significantly affect the efficiency of the shooting algorithm. The boundary conditions of a shooting interval are such that the time domain integrations are consistent with a frequency domain transformation with a shift of one large signal period.

QPSS inherits 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 autonomoussimulation and the QPSS. To do the semi-autonomous simulation, users need to

specifyan initial frequency guess for the oscillator inside the circuit, and two oscillatorterminals, 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 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. Possible values are box, diamond, funnel or axis. Default is diamond when `maximorder` or `boundary` is set; otherwise, default is box.
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 `stabcycles=2`	Stabilization cycles with both large and moderate sources enabled..
11 `tstart=0.0 s`	Initial transient analysis start time.

Time-step parameters

12 `maxstep (s)`	Maximum time step. Default derived from `errpreset`.
13 `maxacfreq`	Maximum frequency requested in a subsequent periodic small-signal analysis. Default derived from `errpreset` and `harms`. This parameter is valid only for shooting.
14 `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

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

Convergence parameters

18 `readns`	File that contains estimate of initial transient solution.
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`, or `nooutput`.
21 `nestlvl`	Levels of subcircuits to output.
22 `oppoint=no`	Should operating point information be computed for initial timestep, and if so, where should it be sent. Possible values are `no`, `screen`, `logfile`, or `rawfile`.
23 `skipstart=starttime s`	The time to start skipping output data.
24 `skipstop=stoptime s`	The time to stop skipping output data.
25 `skipcount`	Save only one of every skipcount points.
26 `strobeperiod (s)`	The output strobe interval (in seconds of transient time).
27 `strobedelay=0 s`	The delay (phase shift) between the skipstart time and the first strobe point.
28 `compression=no`	Do data compression on output. See full description below. Possible values are `no` or `yes`.
29 `saveinit=no`	If set, the waveforms for the initial transient before steady state are saved. Possible values are `no` or `yes`.

State-file parameters

30 `write`	File to which initial transient solution (before steady-state) is to be written.
31 `writefinal`	File to which final transient solution in steady-state is to be written. This parameter is now valid only for shooting.
32 `swapfile`	Temporary file to hold steady-state information. Tells Spectre to use a regular file rather than virtual memory to hold the periodic operating point. Use this option if Spectre complains about not

having enough memory to complete the analysis. This parameter is now valid only for shooting.

Integration method parameters

33 `method` — Integration method. Default derived from `errpreset`. This parameter is valid only for shooting.
Possible values are `euler`, `trap`, `traponly`, `gear2`, or `gear2only`.

Accuracy parameters

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

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

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

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

38 `maxperiods` — Maximum number of simulated periods to reach steady-state.

39 `lnsolver=gmres` — Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.

40 `resgmrescycle=short` — restarted gmres cycle.For large signal analysis, ignore recycleinstant,recycleshort and recyclelong options.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

41 `itres=1e-4 for shooting, 0.9 for HB` — Relative tolerance for linear solver. The value is between [0,1]..

42 `inexactNewton=no` — Inexact Newton method.
Possible values are `no` or `yes`.

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

Harmonic Balance parameters

44 `harmonicbalance=no`
Use Harmonic Balance engine instead of time-domain shooting.
Possible values are `no` or `yes`.

45 `flexbalance=no` Same parameter as `harmonicbalance`.
Possible values are `no` or `yes`.

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

47 `hbpartition_fundnames=[...]`
Specify HB partition fundamental frequency names.

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

49 `oversamplefactor=1`
Oversample device evaluations.

50 `oversample=[...]` Array of oversample factors for each tone. It overrrides oversamplefactor.

51 `hbhomotopy` Name of flexbalance homotopy selection methods. Possible values are tstab or source. Default is is tstab.Not applicable for autonomous circuit.

52 `backtracking=no` This parameter is used to activate the backtracing utility of Newtons Method.Default is no.
Possible values are `no` or `yes`.

Annotation parameters

53 `stats=no` Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

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

55 `title` — Analysis title.

Newton parameters

56 `maxiters=5` — Maximum number of iterations per time step.

57 `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.
Possible values are `no` or `yes`.

Circuit age

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

59 `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. This parameter is now valid only for shooting.

60 `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. This parameter is now valid only for shooting.

Tstab save/restart parameters

61 `ckptperiod` — Checkpoint the analysis periodically using the specified period.

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

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

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

65 `savefile` — Save the analysis states into the specified file.

66	`recover`	Specify the file to be restored.
67	`semiauto=no`	This option is used to activate semi-autonomous simulation. Possible values are `no` or `yes`.
68	`oscic=default`	Oscillator IC method. It determines how the starting values for the oscillator are determined. Possible values are `default` or `lin`.

Most of QPSS analysis parameters are inherited from PSS analysis, and their meanings remain essentially unchanged. Two new important parameters are `funds` and `maxharms`. They replace and extend the role of `fund` and `harms` parameters of PSS analysis. One important difference is that `funds` accepts a list of fundamental names instead of actual frequencies. The frequencies associated with fundamentals are figured out automatically by the simulator. An important feature is that each input signal can be a composition of more than one sources. However, these sources must have the same fundamental name. For each fundamental name, its fundamental frequency is the greatest common factor of all frequencies associated with the name. 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 might set errpreset to `liberal`. If have some concern for accuracy, you might set errpreset to `moderate`. If accuracy is your main interest, you might set errpreset to `conservative.`

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

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

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

Parameter Index

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

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 is it useful for predicting the noise behavior of mixers, switched-capacitor filters, and other periodically driven circuits. It is particularly useful for predicting the phase noise of autonomous circuits, such as oscillators.

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

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

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

If `port` is specified as the input probe, then both input-referred noise and gain are refered 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 tothe 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.

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)

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.

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.

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

Definition

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

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

Parameters

Sweep interval parameters

1	`start=0`	Start sweep limit.
2	`stop`	Stop sweep limit.
3	`center`	Center of sweep.
4	`span=0`	Sweep limit span.
5	`step`	Step size, linear sweep.

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

7 `dec` Points per decade.

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

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

10 `sweeptype=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` or `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 for which transitions to do the sampling.
Possible values are `all`, `rise`, `fall`, or `ignore`.

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

18 `externalsourcedata`
Name of PXF analysis that provide information to compute the contribution of external jitter sources.

Output parameters

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

20 `maxsideband=7` Maximum sideband included when computing noise either up-converted or down-converted to the output by the periodic drive signal. It is ignored in HB small signal when its larger than the harms/maxharms of large signal.

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

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

23 `nestlvl` Levels of subcircuits to output.

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

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

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

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

28 `numberofpoints=5` Number of time points of interest in the period where to calculate time domain PSD. Simulator divides the period evenly into N segments (N=numberofpoints) and calculates time domain PSD

on the starting time point of each segment. When < 0, the parameter is ignored.

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

30 `separatenoise=no` Separate Noise into sources and transfer functions.
Possible values are `no` or `yes`.

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

Convergence parameters

32 `tolerance` Relative tolerance for linear solver, default value is 1.0e-9 for shooting-based solver; 1.0e-6 for driven and 1.0e-4 for autonomous for flexbalance-based solver.

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

34 `solver=turbo` Solver type.
Possible values are `std` or `turbo`.

35 `oscsolver=turbo` Oscillator solver type. Suggest to use ira for huge circuit.
Possible values are `std`, `turbo` or `ira`.

36 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.

37 `resgmrescycle=short`
restarted gmres cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

38 `ppv=no` If yes, save the oscillator PPV after doing noise analysis.
Possible values are `no` or `yes`.

39 `ppvfile` File to which the PPV of oscilator is to be written.

40 `augmented=no` If yes, the frequency-aware PPV method (instead of the Floquet deflation) is used to do pnoise analysis for oscillators. This option

	is only avaliable for frequency domain analysis now. Possible values are `no`, `yes` or `ppvonly`.

Annotation parameters

41 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
42 `stats=no`	Stats parameter is not supported. Please use annotate. Possible values are `no` or `yes`.
43 `title`	Analysis title.

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

f(noise_source) = f(out) + Ki * fund(pss)

The `maxsideband` parameter specifies the maximum |Ki| included in the PNoise calculation. Thus, 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 are included in the calculation. Care should be taken when specifying the sidebands because the results will be in error if you do not include a sideband that contributes significant noise to the output.

The number of requested sidebands does not change substantially the simulation time. However, the `maxacfreq` of the corresponding PSS analysis should be set to guarantee that |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 the majority of the simulations, however, this is not an issue, because `maxacfreq` is never allowed to be smaller than 40x the PSS fundamental.

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

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

Parameter Index

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

Periodic S-Parameter Analysis (psp)

Description

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

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

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

Definition

```
Name psp parameter=value ...
```

Parameters

Sweep interval parameters

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

4 `span=0` Sweep limit span.

5 `step` Step size, linear sweep.

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

7 `dec` Points per decade.

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

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

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` or `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 output versus the input frequency, the output frequency, or the absolute value of the input frequency. Default is `in`.
Possible values are `absin`, `in` or `out`.

Noise parameters

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

Probe parameters

16 `maxsideband=7` Maximum sideband included when computing noise either up-converted or down-converted to the output by the periodic drive signal. It is ignored in HB small signal when its larger than the harms/maxharms of large signal.

Convergence parameters

17 `tolerance` Relative tolerance for linear solver, default value is 1.0e-9 for shooting-based solver; 1.0e-6 for driven and 1.0e-4 for autonomous for flexbalance-based solver.

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

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

20 `oscsolver=turbo` Oscillator solver type. Suggest to use ira for huge circuit.
Possible values are `std`, `turbo` or `ira`.

21 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.

22 `resgmrescycle=short`
restarted gmres cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

Annotation parameters

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

24 `stats=no` Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

25 `title` Analysis title.

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

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

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

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

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

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

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

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

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

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

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, not the gain from the internal port voltage source to the output.

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

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

Parameter Index

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

```
donoise  15        maxsideband  16    start  1          values  9

freqaxis  14       oscsolver  20      stats  24

gear_order  18     portharmsvec  12   step  5

harmsvec  13       ports  11          stop  2
```

Periodic Steady-State Analysis (pss)

Description

This analysis computes the periodic steady-state (PSS) response of a circuit 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. Also, 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 weakly nonlinear circuits while shooting is more suitable for highly nonlinear 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 the user to specify a pair of nodes, p and n. In fact this is how PSS analysis determines whether it is being applied to an autonomous or a driven circuit. If the pair of nodes is supplied, PSS assumes the circuit is autonomous; if not, the circuit is assumed to be driven.

With driven circuits the user specifies the analysis `period`, or its corresponding fundamental frequency `fund`. The `period` must be an integer multiple of the period of the drive signal or signals. Autonomous circuits have no drive signal and the actual period of oscillation is not known precisely by the user in advance. Instead, the user specifies an estimate of the

oscillation period and PSS analysis computes the precise period along with the periodic solution waveforms.

A 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 4x `period` for autonomous circuits. This interval has a special use for the autonomous PSS analysis, i.e., the PSS analysis monitors the waveforms in the circuit and develops a better estimate of the oscillation period. Once the initial transient phase is complete, the shooting or HB phase begins. In this phase, the circuit is iteratively solved 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`	Possible values are no and yes. If the value is yes, the program will ignore period/fund value and calculate the fundamental frequency automatically from source information. The default value is no.. Possible values are `no` or `yes`.
4	`tstab=0.0 s`	Extra stabilization time after the onset of periodicity for independent sources.
5	`tstart=0.0 s`	Initial transient analysis start time.

6 `tstabenvlp=no` Determine will envelope method be used for tstab. If the value be set to yes, envelope method will be used for tstab. Default value is no..
Possible values are `no` or `yes`.

7 `envlpname` Name of envlp analysis to be performed at tstab for pss..

Time-step parameters

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

9 `maxacfreq` Maximum frequency requested in a subsequent periodic small-signal analysis. Default derived from `errpreset` and `harms`. This parameter is valid only for shooting.

10 `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

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

12 `skipdc=no` If yes, there is no dc analysis for initial transient.
Possible values are `no`, `yes` or `sigrampup`.

13 `readic` File that contains initial condition.

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

Convergence parameters

15 `readns` File that contains estimate of initial transient solution.

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

Output parameters

17 `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.

18 `harmsvec=[...]` Array of desired harmonics. Alternate form of `harms` that allows selection of specific harmonics. This parameter is valid only for shooting.

19 `outputtype= all''`
Output type.
Possible values are `all`, `time` or `freq`.

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

21 `nestlvl` Levels of subcircuits to output.

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

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

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

25 `skipcount` Save only one of every skipcount points.

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

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

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

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

State-file parameters

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

31 `writefinal` — File to which final transient solution in steady-state is to be written. This parameter is now valid only for shooting.

32 `swapfile` — Temporary file to hold steady-state information. Tells Spectre to use a regular file rather than virtual memory to hold the periodic operating point. Use this option if Spectre complains about not having enough memory to complete the analysis. This parameter is now valid only for shooting.

33 `writepss` — File to which the converged steady-state solution is to be written. This parameter is now valid only for shooting.

34 `readpss` — File from which a previously converged steady-state solution is to be read. PSS loads the solution and checks the residue of the circuit equations only. The solution is re-used if the residue is satisfying. Otherwise, the solution is re-converged using the finite difference method. This parameter is now valid only for shooting.

35 `checkpss=yes` — If yes, the previous PSS results (from readpss file) are checked and redo PSS+MIC if any condition has changed. If no, the program assumes nothing has change and uses the solution from the file without check/redo PSS+MIC. This parameter is now valid only for shooting.
Possible values are `no` or `yes`.

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`.

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

Accuracy parameters

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

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

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

41 `steadyratio` — Ratio used to compute steady state tolerances from LTE tolerance. Default derived from `errpreset`.

42 `maxperiods` — Maximum number of simulated periods to reach steady-state.

43 `lnsolver=gmres` — Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.

44 `resgmrescycle=short` — restarted gmres cycle.For large signal analysis, ignore recycleinstant,recycleshort and recyclelong options.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

45 `itres=1e-4 for shooting, 0.9 for HB` — Relative tolerance for linear solver. The value is between [0,1]..

46 `inexactNewton=no` — Inexact Newton method.
Possible values are `no` or `yes`.

47 `finitediff` — Options for finite difference method refinement after shooting method for driven circuits.
Possible values are `no`, `yes` or `refine`.

48 `highorder` — Perform a high-order refinement after low-order convergence. The Multi-Interval Chebyshev polynomial spectral algorithm will be used. This parameter is only valid for shooting.
Possible values are `no` or `yes`.

49 `psaratio=1` — Ratio used to compute high-order polynomial spectral accuracy from Newton tolerance. This parameter is now only valid for shooting.

50 `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; 12 for autonomous circuits. This parameter is now only valid for shooting.

51 `fullpssvec` — Use the full vector containing solutions at all PSS time steps in the linear solver. Default derived from the size of the equation and the property of the PSS time steps. This parameter is only valid for shooting.
Possible values are `no` or `yes`.

52 `fdharms=10` — Number of harmonics considered for distributed (frequency-domain) components, such as nport, delay, mtline, delayed controlled sources. This parameter is valid only for shooting and only for those components whose Fmax parameter of neither model nor instance is set. .

Harmonic Balance parameters

53 `harmonicbalance=no` — Use Harmonic Balance engine instead of time-domain shooting.
Possible values are `no` or `yes`.

54 `flexbalance=no` — Same parameter as `harmonicbalance`.
Possible values are `no` or `yes`.

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

56 `hbpartition_defs=[...]` — Define HB partitions.

57 `hbpartition_funds=[...]` — Specify HB partition fundamental frequencies.

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

59 `oversamplefactor=1`
Oversample device evaluations.

60 `oversample=[...]` Array of oversample factors for each tone. It overrrides oversamplefactor.

61 `oscmethod` Osc Newton method for autonomous HB: onetier (default) or twotier.

62 `newphaseeqn=default`
This option is used to activate the new pinning method for oscillators in harmonic balance analysis.
Possible values are `default` or `yes`.

63 `hbhomotopy` Name of flexbalance homotopy selection methods. Possible values are tstab or source. Default is is tstab.Not applicable for autonomous circuit.

64 `backtracking=no` This parameter is used to activate the backtracing utility of Newtons Method.Default is no.
Possible values are `no` or `yes`.

65 `hbhighq=no` This parameter is used to activate the harmonic balance based algorithm for high Q oscillators simulation.Default is no.For low Q or moderate Q oscillators, use default value.
Possible values are `no` or `yes`.

Annotation parameters

66 `stats=no` Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

67 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, `iters`, `detailed`, `rejects`, or `alliters`.

68 `title` Analysis title.

Newton parameters

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

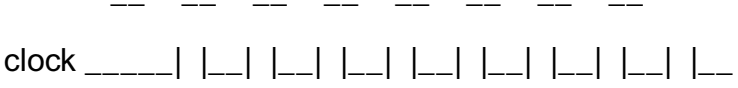
Circuit age

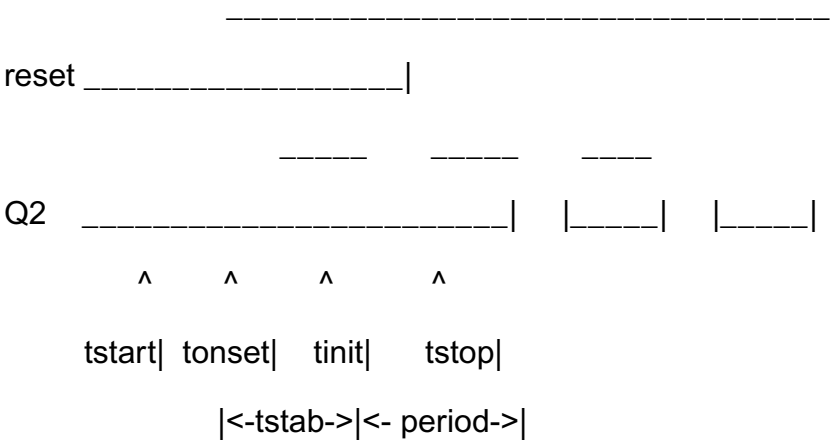
71 `circuitage (Years)`	Stress Time. Age of the circuit used to simulate hot-electron degradation of MOSFET and BSIM circuits.

Tstab save/restart parameters

72 `ckptperiod`	Checkpoint the analysis periodically using the specified period.
73 `saveperiod`	Save the tran analysis periodically on the simulation time.
74 `saveclock=1800 s`	Save the tran analysis periodically on the wall clock time.
75 `savetime=[...]`	Save the analysis states into files on the specified time points.
76 `savefile`	Save the analysis states into the specified file.
77 `recover`	Specify the file to be restored.
78 `ppv=no`	If yes, save the oscillators perturbation projection vector or PPV,representing the oscillators phase sensitivity to perturbations in the voltage or current at the nodes of the oscillator. Possible values are `no` or `yes`.

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





In the figure above, 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 the 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 bag 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, then it is likely the opamps will undergoing 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 initialization 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 using initial conditions, or 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.

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, 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 the exact same fashion 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, error will be caused by aliasing effect and 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 error, or the 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 node voltages or current branches of the steady-stateThis value is multiplied by the `lteratio` and `reltol` to determine the convergence criterion. The relative convergence norm is printed out along with the actual mismatch value at the end of each iteration, thus indicating the progress of the steady-state iteration.

The parameter `maxperiods` controls the maximum number of shooting iterations for PSS analysis. Its default value is set to 20 for driven PSS and 50 for autonomous PSS in shooting and 50 for both in HB.

The `finitediff` parameter allows the use of finite difference (FD) after shooting. Usually this will eliminate the above mismatch in node voltages or current branches. It can also refine the grid of time steps. In some cases, numerical error of the linear solver still introduces a

mismatch. One can set `steadyratio` to a smaller value to activate a tighter tolerance for the iterative linear solver. If `finitediff` is set to no, FD method is turned off. If it is set to yes, pss applies FD method and trying to improve the beginning small time steps if necessary. If it is set to refine, pss applies FD method and tries to refine the time steps. When the simulation uses 2nd-order method, uniform 2nd order gear is used. `finitediff` is changed from no to yes automatically when `readpss` and `writepss` are used to re-use PSS results

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

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

`relref=pointlocal`: Compares the relative errors in quantities at each node to that node alone.

`relref=alllocal`: Compares the relative errors at each node to the largest values found for that node alone for all past time.

`relref=sigglobal`: Compares relative errors in each of the circuit signals to the maximum for all signals at any previous point in time.

`relref=allglobal`: Same as `relref=sigglobal` except that it also compares the residues (KCL error) for each node to the maximum of that nodes past history.

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

Guidelines for using errpreset in driven circuits in shooting are described in the following. If the circuit contains only one periodic tone and you are only interested in obtaining the periodic operating point, you might set errpreset to `liberal`, which gives a reasonably accurate result and the fastest simulation speed. If the circuit contains more than one periodic tone and you are interested in intermodulation results, you might set errpreset to `moderate`, which gives a very accurate result. If you want a very low noise floor in your simulation result and accuracy is your main interest, you might set errpreset to `conservative`.

The effect of `errpreset` on other parameters for driven circuits is shown in the following table.

Parameter defaults and estimated numerical noise floor in simulation

result as a function of `errpreset`

---

errpreset reltol relref method lteratio steadyratio maxstep

---

liberal 1e-3 sigglobal traponly 3.5 0.001 period/50

moderate 1e-3 alllocal gear2only 3.5 0.001 period/200

conservative 1e-4 alllocal gear2only * 0.01 period/200

* : lteratio=10.0 for conservative `errpreset`. Only if user 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 actually an upper limit. An increase of `reltol` above default will be ignored by the simulator. User can decrease this value in the options statement. The only way to increase `reltol` is to `relax errpreset`.

Estimated numerical noise floor for a weakly nonlinear 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 will drop 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 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. 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 value of `errpreset`.

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

The effect of `errpreset` on other parameters for autonomous circuits is shown in the following table.

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

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

A long stabilization (by specifying a large tstab) can help with the PSS convergence. However it can slow down simulation. By default, in the stabilization stage, reltol=1e-3; maxstep=period/25; relref=sigglobal; and method=traponly. They 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 may specify the initial condition for the transient analysis by using the `ic` statement or the `ic` parameter on the capacitors and inductors. If you do not specify the initial condition, the DC solution is used as the initial condition. The `ic` parameter on the transient analysis controls the interaction of various methods of setting the initial conditions. The effects of individual settings are:

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

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

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

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

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

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

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

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

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

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

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

`skipdc=sigrampup`: Independent source values start at 0 and ramp up to their initial values in the first phase of the simulation. The waveform production in the time-varying independent source is enabled after the rampup phase.The rampup simulation is from

`tstart` to time=0 s, and the main simulation is from time=0 s to `tstab`. If the `tstart` parameter is not specified,the default `tstart` time is set to -0.1*`tstab`.

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

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

When you simulate the same circuit many times, we suggest that you use both the `write` and `readns` parameters and give the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.

Nodesets and initial conditions have similar implementation but produce different effects. Initial conditions actually define the solution, whereas nodesets only influence it. When you simulate a circuit with a transient analysis, Spectre forms and solves a set of differential equations. However, differential equations have an infinite number of solutions, and a complete set of initial conditions must be specified in order to identify the desired solution. Any initial conditions you do not specify are computed by the simulator to be consistent. The transient waveforms then start from initial conditions. Nodesets are usually used as a convergence aid and do not affect the final results. However, in a circuit with more than one solution, such as a latch, nodesets bias the simulator towards finding the solution closest to the nodeset values.

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

`method=euler`: Backward-Euler is used exclusively.

`method=traponly`: Trapezoidal rule is used almost exclusively.

`method=trap`: Backward-Euler and the trapezoidal rule are used.

`method=gear2only`: Gears second-order backward-difference method is used almost exclusively.

`method=gear2`: Backward-Euler and second-order Gear are used.

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

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

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

The skipping method is controlled by `skipcount`. If this is set to N, then only every Nth point is saved.

The parameters `skipstart` and `skipstop` apply to both data reduction methods. Before `skipstart` and after `skipstop`, Spectre saves all computed data.

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

Parameter Index

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

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 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 a normal part of the PSS analysis, the periodically time-varying representation of the circuit is computed and saved for later use. In the second, 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` 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. For the time-varying property of the circuit the loop gain at different places can be different but all 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 only be used on individual feedback loops to ensure 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 ...
```

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.

Output parameters

11 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
12 `nestlvl`	Levels of subcircuits to output.

Convergence parameters

13 `tolerance`	Relative tolerance for linear solver, default value is 1.0e-9 for shooting-based solver; 1.0e-6 for driven and 1.0e-4 for autonomous for flexbalance-based solver.
14 `gear_order=2`	Gear order used for small-signal integration.
15 `solver=turbo`	Solver type. Possible values are `std` or `turbo`.
16 `oscsolver=turbo`	Oscillator solver type. Suggest to use ira for huge circuit. Possible values are `std`, `turbo` or `ira`.

17 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.

18 `resgmrescycle=short`
restarted gmres cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

Annotation parameters

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

20 `stats=no` Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

21 `title` Analysis title.

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

Parameter Index

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

```
dec  7              oscsolver  16       start  1           values  9

gear_order  14      probe  10           stats  20

lin  6              resgmrescycle  18   step  5

lnsolver  17        save  11            stop  2
```

Periodic Transfer Function Analysis (pxf)

Description

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

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

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

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

Definition

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

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

Parameters

Sweep interval parameters

1 `start=0` Start sweep limit.

2 `stop` Stop sweep limit.

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` 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 will trigger the sampled small signal computation.

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

16 `crossingdirection=all`
Specifies for which transitions to do the sampling.
Possible values are `all`, `rise`, `fall`, or `ignore`.

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

18 `sampledelay=[...]` Sampled measurement is done with delay from the time of the crossing event on the conrol signal or probe. The first value corresponds to the rising edge and the second to the falling transition.

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

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=0`	An alternative to the `sidebands` array specification, which automatically generates the array: [-maxsideband ... 0 ... +maxsideband]. It is ignored in HB small signal when its 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 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.

Convergence parameters

31 `tolerance`	Relative tolerance for linear solver, default value is 1.0e-9 for shooting-based solver; 1.0e-6 for driven and 1.0e-4 for autonomous for flexbalance-based solver.
32 `gear_order=2`	Gear order used for small-signal integration.
33 `solver=turbo`	Solver type. Possible values are `std` or `turbo`.
34 `oscsolver=turbo`	Oscillator solver type. Suggest to use ira for huge circuit. Possible values are `std`, `turbo` or `ira`.
35 `lnsolver=gmres`	Linear solver. Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.
36 `resgmrescycle=short`	restarted gmres cycle.

Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

Annotation parameters

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

38 `stats=no` Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

39 `title` Analysis title.

Modulation conversion parameters

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

41 `outmodharmnum=1` Harmonic for the PXF output modulation.

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

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

The variable of interest at the output can be voltage or current, and its frequency is not constrained by the period of the large periodic solution. While sweeping the selected output frequency, you can select the periodic small-signal input frequencies of interest by setting either the `maxsideband` or the `sidebands` parameters. For a given set of n integer numbers representing the sidebands K1, K2, ... Kn, the input signal frequency at each sideband is computed as f(in)= f(out) + Ki * fund(pss), where f(out) represent 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 the majority of the simulations, however, this is not an issue, because maxacfreq is never allowed to be smaller than 40x the PSS fundamental.

With PXF the frequency of the stimulus and of the response are usually different (this is an important way in which PXF differs from XF). The freqaxis parameter is used to specify whether the results should be output versus the input frequency (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 what is used for the inputs for the transfer functions. There are two choices. stimuli=sources indicates that the sources present in the circuit should be used. The xfmag parameters provided by the sources may be used to adjust the computed gain to compensate for gains or losses in a test fixture. One can limit the number of sources in hierarchical netlists by using the save and nestlvl parameters. stimuli=nodes_and_terminals indicates that all possible transfer functions should be computed.

This is useful when it is not known in advance which transfer functions are interesting. Transfer functions for nodes are computed assuming that a unit magnitude flow (current) source is connected from the node to ground. Transfer functions for terminals are computed assuming that a unit magnitude value (voltage) source is connected in series with the terminal. By default, the transfer functions from a small set of terminals are computed. If transfer functions from specific terminals are desired, specify the terminals in the save statement. You must use the :probe modifier (ex. Rout:1:probe) or specify useprobes=yes on the options statement. If transfer functions from all terminals are desired, specify currents=all and useprobes=yes on the options statement.

Modulated small signal measurements are possible using the Analog Artist(ADE) environment. The modulated option for PXF and other modulated parameters are set by Artist. PXF analyses with this option will produce results which could have limited use outside such environment. Direct Plot is configured to analyze these results and combine several

wave forms to measure AM and PM transfer function from single sideband or modulated stimuli to the specified output. For details, please see the SpectreRF User Guide.

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

Parameter Index

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

```
extrasampletimepo   nestlvl  30         solver  33          values  9
ints  19

freqaxis  28        oscsolver  34       span  4
```

PZ Analysis (pz)

Description

The PZ analysis linearizes the circuit about the DC operating point and computes the poles and zeros of the linearized network. To compute zeros, users need to specify input sources and output voltages or currents. If no input or output are given, then only poles are computed. In case there are frequency dependent components, poles and zeros are computed by approximating those components as equivalent conductances and capacitances evaluated at 1Hz. The PZ analysis uses default direct solver (method=qz) for better accuracy. Small to medium circuit size will achieve better performance. For larger circuits, a Krylov subspace iterative solver (method=arnoldi) can be used for better performance but at lesser accuracy.

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

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

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

Definition

```
Name ... pz parameter=value ...
```

Parameters

Probe parameters

1 `iprobe`	Input probe for zeros of the transfer function.
2 `oprobe`	Output probe for zeros of the transfer function.

Port parameters

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

Sweep interval parameters

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

Sweep variable parameters

14 `dev`	Device instance whose parameter value is to be swept.
15 `mod`	Model whose parameter value is to be swept.

16 `param`	Name of parameter to sweep.
17 `freq (Hz)`	Frequency at which components will be evaluated in setting up the linearized network.

State-file parameters

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

Output parameters

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

Filtering parameters

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

Convergence parameters

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

Annotation parameters

27 `stats=no`	Analysis statistics. Possible values are `no` or `yes`.
28 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
29 `title`	Analysis title.

Miscellaneous parameters

30 `method=qz`	Method to perform pz analysis. Possible values are `qz` or `arnoldi`.
31 `numpoles`	Maximum number of poles requested, for arnoldi method only.
32 `numzeros`	Maximum number of zeros requested, for arnoldi method only.
33 `sigmar=0.1`	root finding control parameter, for arnoldi method only.
34 `sigmai=0.0`	root finding control parameter, for arnoldi method only.

Examples:

mypz pz

- pole analysis will be performed.

mypz2 (n1 n2) pz iprobe=VIN

- input is VIN, output is voltage difference between nodes n1 and n2.
- both pole and zero analyses will be performed.

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
```

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

Parameter Index

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

Quasi-Periodic AC Analysis (qpac)

Description

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

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

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

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

Definition

```
Name qpac parameter=value ...
```

Parameters

Sweep interval parameters

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

4 `span=0`	Sweep limit span.
5 `step`	Step size, linear sweep.
6 `lin=50`	Number of steps, linear sweep.
7 `dec`	Points per decade.
8 `log=50`	Number of steps, log sweep.
9 `values=[...]`	Array of sweep values.
10 `sweeptype`	Specifies if the sweep frequency range is an absolute frequency, i.e. actual frequency; or if it is relative to the "relharmvec" sideband frequency. 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=0`	An alternative to the `sidevec` array specification, which automatically generates the array: [-clockmaxharm ... 0 ... +clockmaxharms][-maxharms(QPSS)[2]...0...maxharms(QPSS)[2]][...].
14 `freqaxis`	Specifies whether the results should be output versus the input frequency, the output frequency, or the absolute value of the output frequency. Default is `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.

Convergence parameters

17 `tolerance` Relative tolerance for linear solver, default value is 1.0e-9 for shooting-base solver; 1.0e-6 for flexbalance-based solver.

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

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

20 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.

21 `resgmrescycle=short`
restarted gmres cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

Annotation parameters

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

23 `stats=no` Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

24 `title` Analysis title.

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

K1_1 * fund(large tone of QPSS) + K1_2 * fund(moderate tone of QPSS)

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

K1 = { K1_1,...K1_j..., K1_L} , K2, ... Kn. The output frequency at each sideband is computed as

f(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}. User would enter `sidevec` as a sequence of integer numbers, separated by spaces. The set of vectors {1 1 0} {1 -1 0} {1 1 1} becomes sidevec=[1 1 0 1 -1 0 1 1 1]. For `clockmaxharm`, only the large tone - first fundamental will be affected by this entry, all the rest - moderate tones - will be limited by `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 way in which QPAC differs from AC). The `freqaxis` parameter is used to specify whether the results should be output versus the input frequency (`in`), the output frequency (`out`), or the absolute value of the output frequency (`absout`).

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

Parameter Index

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

dec 7	nestlvl 16	span 4	title 24
freqaxis 14	relharmvec 11	start 1	tolerance 17
gear_order 18	resgmrescycle 21	stats 23	values 9

Quasi-Periodic Noise Analysis (qpnoise)

Description

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

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

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

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

If `port` is specified as the input probe, then both input-referred noise and gain are refered 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 the input at the individual quasi-periodic sideband specified. Note that Different sidebands can lead to the same frequency.

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

Ki_1 * 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)

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 give the terminals, then you must specify the output with a probe component.

Parameters

Sweep interval parameters

1 `start=0` Start sweep limit.

2 `stop`	Stop sweep limit.
3 `center`	Center of sweep.
4 `span=0`	Sweep limit span.
5 `step`	Step size, linear sweep.
6 `lin=50`	Number of steps, linear sweep.
7 `dec`	Points per decade.
8 `log=50`	Number of steps, log sweep.
9 `values=[...]`	Array of sweep values.
10 `sweeptype`	Specifies if the sweep frequency range is an absolute frequency, i.e. actual frequency; or if it is relative to the "relharmvec" sideband frequency. Possible values are `absolute` or `relative`.
11 `relharmvec=[...]`	Sideband - vector of QPSS harmonics - to which relative frequency sweep should be referenced.

Probe parameters

12 `oprobe`	Compute total noise at the output defined by this component.
13 `iprobe`	Refer the output noise to this component.
14 `refsideband=[...]`	Conversion gain associated with this sideband is used when computing input-referred noise or noise figure.
15 `refsidebandoption=individual`	Whether to view the sideband as a specification of a frequency or a specification of an individual sideband. Possible values are `freq` or `individual`.

Output parameters

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

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

18 `save` Signals to output.
Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `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, default value is 1.0e-9 for shooting-base solver; 1.0e-6 for flexbalance-based solver.

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

24 `solver=turbo` Solver type.
Possible values are `std` or `turbo`.

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

26 `resgmrescycle=short`
restarted gmres cycle.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

Annotation parameters

27	`annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
28	`stats=no`	Stats parameter is not supported. Please use annotate. Possible values are `no` or `yes`.
29	`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 only includes 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 will be represented as [K1_1 K1_2]. Corresponding frequency shift is

K1_1 * fund(large tone of QPSS) + K1_2 * fund(moderate tone of QPSS)

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

K1 = { K1_1,...K1_j..., K1_L} ,

K2, ... Kn.

If Ki represents sideband i, then

f(noise_source) = f(out) + SUM_j=1_to_L{ Ki_j * fund_j(qpss) },

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

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

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

parameter should take using the `values` parameter. If you give both a specific set of values and a set specified using a sweep range, the two sets are merged and collated before being used. All frequencies are in Hertz.

Parameter Index

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

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 in QPAC and QPXF, the circuit is first linearized about a quasiperiodically varying operating point as opposed to a simple DC operating point. Linearizing about a quasiperiodically time-varying operating point allows the computation of S-parameters between circuit ports that convert signals from one frequency band to another.

QPSP can also calculate noise parameters in frequency-converting circuits. QPSP computes noise figure (both single-sideband and double-sideband), input referred noise, equivalent noise parameters, and noise correlation matrices. As in QPNOISE, but unlike SP, the noise features of the QPSP analysis include noise folding effects due to the periodically time-varying nature of the circuit.

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

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

Definition

```
Name qpsp parameter=value ...
```

Parameters

Sweep interval parameters

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

9 `values=[...]`	Array of sweep values.
10 `sweeptype`	Specifies if the sweep frequency range is an absolute frequency, i.e. actual frequency; or if it is relative to the "relharmvec" sideband frequency. In QPSP, relative means relative to the input port frequency. Possible values are `absolute` 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. Default is `in`. Possible values are `absin`, `in` or `out`.

Noise parameters

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

Probe parameters

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

Convergence parameters

17 `tolerance`	Relative tolerance for linear solver, default value is 1.0e-9 for shooting-base solver; 1.0e-6 for flexbalance-based solver.
18 `gear_order=2`	Gear order used for small-signal integration, 1 or 2.
19 `solver=turbo`	Solver type. Possible values are `std` or `turbo`.
20 `lnsolver=gmres`	Linear solver. Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.
21 `resgmrescycle=short`	restarted gmres cycle. Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

Annotation parameters

22 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
23 `stats=no`	Stats parameter is not supported. Please use annotate. Possible values are `no` or `yes`.
24 `title`	Analysis title.

To specify the QPSP analysis the port and sideband combinations must be specified. You can select the ports of interest by setting the `port` parameter and the set of periodic small-signal output frequencies of interest by setting `portharmsvec` or `harmsvec` parameters. Sidebands are vectors in QPSP. Assume we have one large tone and one moderate tone in QPSS. A sideband K1 will be represented as [K1_1 K1_2]. Corresponding frequency is

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

We assume that there are L (1 large plus L-1 moderate) tones in QPSS analysis and a given set of n integer vectors representing the sidebands

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

If we specify the relative frequency then the scattering parameters at each port are computed at the 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 fund_j(qpss) represents the fundamental frequency used in the corresponding QPSS analysis.

In anlysis of a down-converting mixer with a blocker and the signal in the upper sideband, we sweep the input frequency of the signal coming into RF port. The most relevant sideband for this input is Ki= {1 , 0} - and for IF output Ki= {0 , 0}. Hence we can associate K1={1 , 0} with the RF port and K2={0 , 0} with the IF port. S21 will represent the transmission of signal from the RF to IF, and S11 the reflection of signal back to the RF port. If the signal was in the lower sideband, then a choice of K1={-1 , 0} would be more appropriate.

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

With QPSP the frequency of the input and of the response are usually different (this is an important way in which QPSP differs from SP). Because the QPSP computation involves inputs and outputs at frequencies that are relative to multiple sidebands, the `freqaxis` and `sweeptype` parameters behave somewhat differently in QPSP than in QPAC and QPXF.

The `sweeptype` parameter controls the way the frequencies in the QPSP analysis are swept. Specifying a `relative` sweep indicates to sweep relative to the port sideband (not the QPSS fundamental) and an `absolute` sweep is a sweep of the absolute input source frequency. For example, with a QPSS fundamentals of 1000MHz (LO) and 966MHz (blocker in RF channel), `portharmsvec` could be set to [0 1 -1 1] to examine a downconverting mixer. Lets set `sweeptype=relative` and a sweep range of f(rel)=-10MHz<->10MHz. Then S21 would represent the strength of the signal transmitted from the input port in the range 956->976MHz to the output port to the frequencies 24<->44MHz. Using `sweeptype=absolute` and sweeping the frequency from 966<->976MHz would calculate the same quantities, since f(abs)=956<->976MHz, and f(rel) = f(abs) - (K1_1 * fund_1(qpss) + K1_2 * fund_2(qpss) = -10MHz<->10MHz, because K1_1=0, K1_2=1 and fund_1(qpss) = 1000MHz, fund_2(qpss) = 966MHz.

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

An increase in the number of requested ports will increase the simulation time substantially. The same will happen if we increase the number of sidebands to be included into the noise computations.

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

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 QPSP is the voltage gain from the actual circuit input to the circuit output, not the gain from the internal port voltage source to the output.

To ensure accurate noise calculations, the `clockmaxharm` parameters must be set to include the relevant noise folding effects. `clockmaxharm` is only relevant to the noise computation features of QPSP.

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

Parameter Index

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

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 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 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, 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 with all signals activated is run. At last, 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.

As 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). As one consequence, the efficiency of the algorithm depends significantly on the number of harmonics required to model the responses of moderate fundamentals. As another consequence, the number of harmonics of the large fundamental does not significantly affect the efficiency of the shooting algorithm. The boundary conditions of a shooting interval are such that the time domain integrations are consistent with a frequency domain transformation with a shift of one large signal period.

QPSS inherits 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 autonomoussimulation and the QPSS. To do the semi-autonomous simulation, users need to specifyan initial frequency guess for the oscillator inside the circuit, and two oscillatorterminals, 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. Possible values are box, diamond, funnel or axis. Default is diamond when `maximorder` or `boundary` is set; otherwise, default is box.
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 `stabcycles=2`	Stabilization cycles with both large and moderate sources enabled..
11 `tstart=0.0 s`	Initial transient analysis start time.

Time-step parameters

12 `maxstep (s)`	Maximum time step. Default derived from `errpreset`.
13 `maxacfreq`	Maximum frequency requested in a subsequent periodic small-signal analysis. Default derived from `errpreset` and `harms`. This parameter is valid only for shooting.
14 `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

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

16 `skipdc=no` — If yes, there is no dc analysis for initial transient. Possible values are `no`, `yes` or `sigrampup`.

17 `readic` — File that contains initial condition.

Convergence parameters

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

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`, or `nooutput`.

21 `nestlvl` — Levels of subcircuits to output.

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

23 `skipstart=starttime s` — The time to start skipping output data.

24 `skipstop=stoptime s` — The time to stop skipping output data.

25 `skipcount` — Save only one of every skipcount points.

26 `strobeperiod (s)` — The output strobe interval (in seconds of transient time).

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

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

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

State-file parameters

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

31 `writefinal` — File to which final transient solution in steady-state is to be written. This parameter is now valid only for shooting.

32 `swapfile` — Temporary file to hold steady-state information. Tells Spectre to use a regular file rather than virtual memory to hold the periodic operating point. Use this option if Spectre complains about not having enough memory to complete the analysis. This parameter is now valid only for shooting.

Integration method parameters

33 `method` — Integration method. Default derived from `errpreset`. This parameter is valid only for shooting. Possible values are `euler`, `trap`, `traponly`, `gear2`, or `gear2only`.

Accuracy parameters

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

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

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

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

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

39 `lnsolver=gmres` Linear solver.
Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.

40 `resgmrescycle=short`
restarted gmres cycle.For large signal analysis, ignore recycleinstant,recycleshort and recyclelong options.
Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

41 `itres=1e-4 for shooting, 0.9 for HB`
Relative tolerance for linear solver. The value is between [0,1]..

42 `inexactNewton=no` Inexact Newton method.
Possible values are `no` or `yes`.

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

Harmonic Balance parameters

44 `harmonicbalance=no`
Use Harmonic Balance engine instead of time-domain shooting.
Possible values are `no` or `yes`.

45 `flexbalance=no` Same parameter as `harmonicbalance`.
Possible values are `no` or `yes`.

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

47 `hbpartition_fundnames=[...]`
Specify HB partition fundamental frequency names.

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

49 `oversamplefactor=1`
Oversample device evaluations.

50 `oversample=[...]` Array of oversample factors for each tone. It overrrides oversamplefactor.

51 `hbhomotopy` Name of flexbalance homotopy selection methods. Possible values are tstab or source. Default is is tstab.Not applicable for autonomous circuit.

52 `backtracking=no` This parameter is used to activate the backtracing utility of Newtons Method.Default is no.
Possible values are `no` or `yes`.

Annotation parameters

53 `stats=no` Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

54 `annotate=sweep` Degree of annotation.
Possible values are `no`, `title`, `sweep`, `status`, `steps`, `iters`, `detailed`, `rejects`, or `alliters`.

55 `title` Analysis title.

Newton parameters

56 `maxiters=5` Maximum number of iterations per time step.

57 `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.
Possible values are `no` or `yes`.

Circuit age

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

59 `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. This parameter is now valid only for shooting.
60 `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. This parameter is now valid only for shooting.

Tstab save/restart parameters

61 `ckptperiod`	Checkpoint the analysis periodically using the specified period.
62 `saveperiod`	Save the tran analysis periodically on the simulation time.
63 `saveclock=1800 s`	Save the tran analysis periodically on the wall clock time.
64 `savetime=[...]`	Save the analysis states into files on the specified time points.
65 `savefile`	Save the analysis states into the specified file.
66 `recover`	Specify the file to be restored.
67 `semiauto=no`	This option is used to activate semi-autonomous simulation. Possible values are `no` or `yes`.
68 `oscic=default`	Oscillator IC method. It determines how the starting values for the oscillator are determined. Possible values are `default` or `lin`.

Most of QPSS analysis parameters are inherited from PSS analysis, and their meanings remain essentially unchanged. Two new important parameters are `funds` and `maxharms`. They replace and extend the role of `fund` and `harms` parameters of PSS analysis. One important difference is that `funds` accepts a list of fundamental names instead of actual frequencies. The frequencies associated with fundamentals are figured out automatically by the simulator. An important feature is that each input signal can be a composition of more than one sources. However, these sources must have the same fundamental name. For each fundamental name, its fundamental frequency is the greatest common factor of all frequencies associated with the name. 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 might set errpreset to `liberal`. If have some concern for accuracy, you might set errpreset to `moderate`. If accuracy is your main interest, you might set errpreset to `conservative.`

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

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

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

Parameter Index

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

Quasi-Periodic Transfer Function Analysis (qpxf)

Description

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

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

As with a QPAC, QPSP, and QPNOISE analyses, a QPXF analysis must follow a QPSS analysis.

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

Definition

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

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

Parameters

Sweep interval parameters

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

Convergence parameters

19 `tolerance`	Relative tolerance for linear solver, default value is 1.0e-9 for shooting-base solver; 1.0e-6 for flexbalance-based solver.
20 `gear_order=2`	Gear order used for small-signal integration, 1 or 2.
21 `solver=turbo`	Solver type. Possible values are `std` or `turbo`.
22 `lnsolver=gmres`	Linear solver. Possible values are `gmres`, `qmr`, `bicgstab`, or `resgmres`.
23 `resgmrescycle=short`	restarted gmres cycle. Possible values are `instant`, `short`, `long`, `recycleinstant`, `recycleshort`, or `recyclelong`.

Annotation parameters

24 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
25 `stats=no`	Stats parameter is not supported. Please use annotate. Possible values are `no` or `yes`.
26 `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, you can select the periodic small-signal input frequencies of interest by setting either the `clockmaxharm` or the `sidevec` parameters. Sidebands are vectors in QPXF. Assume we have one large tone and one moderate tone in QPSS. A sideband K1 will be represented as [K1_1 K1_2]. Corresponding frequency is

K1_1 * fund(large tone of QPSS) + K1_2 * fund(moderate tone of QPSS)

We assume that there are L (1 large plus L-1 moderate) tones in QPSS analysis and a given set of n integer vectors representing the sidebands

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

The input signal frequency at each sideband is computed as

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

where f(out) represent 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.

User would enter `sidevec` as a sequence of integer numbers, separated by spaces. The set of vectors {1 1 0} {1 -1 0} {1 1 1} becomes sidevec=[1 1 0 1 -1 0 1 1 1]. For `clockmaxharm`, only the large tone - first fundamental will be affected by this entry, all the rest - moderate tones - will be limited by `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 way in which QPXF 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 QPXF analysis statement.

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

The `stimuli` parameter specifies what is used for the inputs for the transfer functions. There are two choices. `stimuli=sources` indicates that the sources present in the circuit should be used. The `xfmag` parameters provided by the sources may be used to adjust the computed gain to compensate for gains or losses in a test fixture. One can limit the number of sources in hierarchical netlists by using the `save` and `nestlvl` parameters. `stimuli=nodes_and_terminals` indicates that all possible transfer functions should be computed.

This is useful when it is not known in advance which transfer functions are interesting. Transfer functions for nodes are computed assuming that a unit magnitude flow (current) source is connected from the node to ground. Transfer functions for terminals are computed assuming that a unit magnitude value (voltage) source is connected in series with the terminal. By default, the transfer functions from a small set of terminals are computed. If transfer functions from specific terminals are desired, specify the terminals in the save statement. You must use the `:probe` modifier (ex. Rout:1:probe) or specify `useprobes=yes` on the options statement. If transfer functions from all terminals are desired, specify `currents=all` and `useprobes=yes` on the options statement.

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

Parameter Index

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

Deferred Set Options (set)

Description

The deferred set options statement sets or changes various program control options. You can set the options in any order and, once set, the options retain their value until reset. The set statement is queued with all analyses and is executed sequentially (The changes made to these options are deferred until the statement setting them is encountered). To set `temp`, `tnom`, `scalem`, or `scale`, use the alter statement. For further options, see individual analyses.

Definition

```
Name set parameter=value ...
```

Parameters

Tolerance parameters

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

Temperature parameters

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

Convergence parameters

6	`homotopy=all`	Method used when no convergence on initial attempt of DC analysis. Possible values are `none`, `gmin`, `source`, `dptran`, `ptran`, `arclength`, or `all`.
7	`limit=dev`	Limiting algorithms to aid DC convergence. Possible values are `delta`, `log` or `dev`.
8	`gmethod=dev`	Stamp gdev, gnode or both in the homotopy methods. See below for more information. Possible values are `dev`, `node` or `both`.
9	`try_fast_op=yes`	This feature often speeds up the DC solution. For hard to converge designs, this feature quietly fails and other methods are applied. In corner cases, this feature may have negative effects. If the DC analysis is unusually slow or the processes memory usage keeps growing or DC just gets stuck even before homotopy methods start, try setting this option to no.. Possible values are `no` or `yes`.

Component parameters

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

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

Error-checking parameters

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

13 `opptcheck=yes`
: Check operating point parameters against soft limits.
Possible values are `no` or `yes`.

Resistance parameters

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

15 `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`.

16 `rforce=1 Ω`
: Resistance used when forcing nodesets and node-based initial conditions.

Quantity parameters

17 `quantities=no`
: Print quantities. If quantities=min, spectre will print out all defined quantities; if quantities=full, spectre will also print a list of nodes and their quantities.
Possible values are `no`, `min` or `full`.

Annotation parameters

18 `narrate=yes` Narrate the simulation.
Possible values are `no` or `yes`.

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

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

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

22 `maxnotes=5` Maximum number of times any notice will be issued per analysis. Note that this option has no effect on notices issued as part of parsing the netlist. Please use the -maxnotes command line option to control the number of all notices issued.

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

24 `maxwarns=5` Maximum number of times any warning message will be issued per analysis. Note that this option has no effect on warnings issued as part of parsing the netlist. Please use the -maxwarns command line option to control the number of all warnings issued.

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

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

27 `error=yes` Give error messages.
Possible values are `no` or `yes`.

28 `digits=5` Number of digits used when printing numbers.

29 `measdgt=0` Number of decimal digits in floating point numbers in measurement output in mt0 fortmat.

30 `ingold=sci`

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

32 `annotate=no` Degree of annotation.
Possible values are `no` or `title`.

Matrix parameters

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

34 `pivrel=0.001` Relative pivot threshold.

35 `pivabs=0` Absolute pivot threshold.

36 `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 are: designs with very large number of small resistors or large number of behavioral instances containing voltage based equations..
Possible values are `partial` or `full`.

37 `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`.

38 `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.

Parameter Index

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

Shell Command (shell)

Description

The shell analysis passes a command to the operating system command interpreter given in the SHELL environment variable. The command behaves as if it were typed into the command interpreter, except that any %X codes in the command are expanded first.

The default action of the shell analysis is to terminate the simulation.

Definition

```
Name shell parameter=value ...
```

Parameters

1	`cmd="kill %P"`	Shell command.
2	`iferror=quit`	What to do if command returns nonzero error status. Possible values are `continue` or `quit`.
3	`annotate`	Degree of annotation. Possible values are `no`, `title` or `yes`.

S-Parameter Analysis (sp)

Description

The S-parameter analysis linearizes the circuit about the DC operating point and computes S-parameters of the circuit taken as an N-port. The port statements define the ports of the circuit. Each active port is turned on sequentially, and a linear small-signal analysis is performed. Spectre converts the response of the circuit at each active port into S-parameters and outputs these parameters. There must be at least one active port statement in the circuit.

If a filename is specified using the `file` parameter, the S-parameter analysis generates an ASCII file containing the S-parameters of the circuit that can later be read-in by the `nport` component. The generated file can be in either Spectres native format or in Touchstone format.

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

Definition

```
Name sp parameter=value ...
```

Parameters

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

Sweep interval parameters

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

Sweep variable parameters

11	`dev`	Device instance whose parameter value is to be swept.
12	`mod`	Model whose parameter value is to be swept.

13 `param`	Name of parameter to sweep.
14 `freq (Hz)`	Frequency when parameter other than frequency is being swept.

Port parameters

15 `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.
21 `skipdc=no`	Skip the DC analysis. Possible values are `no` or `yes`.

Output parameters

22 `file`	S-parameters output file name.
23 `mode="ss"`	S-parameters mode selector. Can be mm for mixed-mode.
24 `datafmt=spectre`	Data format of the S-parameter output file. Possible values are `spectre` or `touchstone`.
25 `datatype=realimag`	Data type of the S-parameter output file. Possible values are `realimag`, `magphase` or `dbphase`.

26 `noisedata=no` — Should noise data be saved to the S-parameter output file, and if so, in what format.
Possible values are `no`, `twoport` or `cy`.

27 `oppoint=no` — Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if operating point is computed in the previous analysis and is unchanged.
Possible values are `no`, `screen`, `logfile`, or `rawfile`.

Noise parameters

28 `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`.

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

30 `iprobe` — Input probe. Refer the output noise to this component.

Convergence parameters

31 `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

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

33 `stats=no` — Stats parameter is not supported. Please use annotate.
Possible values are `no` or `yes`.

34 `title` — Analysis title.

If the list of active ports is specified with the `ports` parameter, then the ports are numbered sequentially from one in the order given. Otherwise, all ports present in the circuit are active, and the port numbers used are those that were assigned on the port statements. If `donoise=yes` is specified, then the noise correlation matrix is computed. If in addition, the output is specified using `oprobe`, the amount that each noise source contributes to the

output is computed. Finally, if an input is also specified (using `iprobe`), the two-port noise parameters are computed (F, Fmin, NF, NFmin, Gopt, Bopt, and Rn).

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 refering to the pairing of the ports, with the port numbers ordered in pair as (1,2) (3,4), etc. in the ports list. With mm, spectre calculates differential-to-differential, differential-to-common, common-to-differential, and comnon-to-common S-parameters. A combination of mixed-mode and standard S-parameters is calculated if the mode parameter is set to, say, for example, 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 of active ports to be exactly 5 given in the port list.

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

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

Parameter Index

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

datatype 25	log 9	readforce 20	title 34
dec 8	mod 12	readns 16	values 10
dev 11	mode 23	restart 31	write 17
donoise 28	noisedata 26	skipdc 21	writefinal 18
file 22	oppoint 27	span 5	
force 19	oprobe 29	start 2	

Stability Analysis (stb)

Description

The STB analysis linearizes the circuit about the DC operating point and computes the loop gain, gain and phase margins (if the sweep variable is frequency), for a feedback loop or a gain device.

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

Definition

```
Name stb parameter=value ...
```

Parameters

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

Sweep interval parameters

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

Sweep variable parameters

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

Probe parameters

15 `probe`	Probe instance around which the loop gain is calculated.
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).

Output parameters

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

Convergence parameters

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

Annotation parameters

22 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
23 `stats=no`	Stats parameter is not supported. Please use annotate. Possible values are `no` or `yes`.
24 `title`	Analysis title.

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

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

Understanding Loop based and Device Based Algorithms

==========

Two algorithms--the loop based and the device based, are available for small-signal stability analysis. Both algorithms are based on the calculation of 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 it points to a current probe or voltage source instance, the loop based algorithm will be invoked; when it points to a supported active device instance, the device based algorithm will be invoked.

Loop Based Algorithm

The loop based algorithm calculates the true loop gain that consists of normal loop gain and reverse loop gain. The loop based algorithm requires the `probe` being placed on the feedback loop to identify and characterize the particular loop of interest. The introduction of the probe component should not change any of the circuit characteristics.

The loop based algorithm provides accurate stability information for single loop circuits, and multiloop circuits in which a `probe` component can be placed on a critical wire to break all loops. For a general multiloop circuit, such a critical wire may not be available. The loop based algorithm can only be performed on individual feedback loops to ensure they are stable. Although the stability of all feedback loops is only a necessary condition for the whole circuit to be stable, the multiloop circuit tends to be stable if all individual loops are associated with reasonable stability margins.

Device Based Algorithm

The device based algorithm calculates the loop gain around a particular active device. This algorithm is often applied to assess the stability of circuit design in which local feedback loops cannot be neglected; the loop based algorithm cannot be performed for these applications since the local feedback loops are inside the devices, they are not accessible from the schematic level or netlist level to insert the `probe` component.

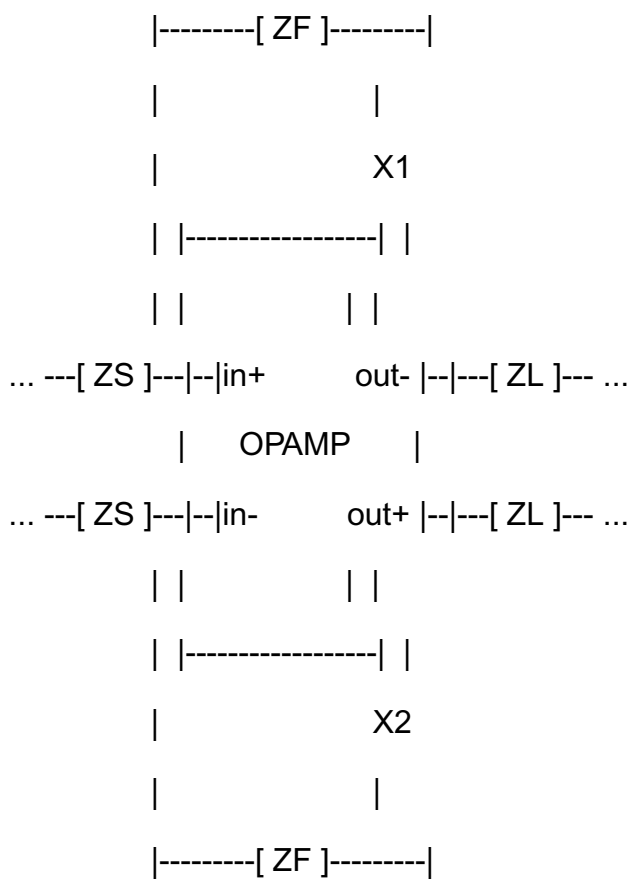
With the `probe` parameter points to a particular active device, the dominant controlled source in the device will be nulled during the analysis. The dominant controlled source is defined as

by nulling this source renders the active device to be passive. The device based algorithm produces accurate stability information for a circuit in which a critical active device can be identified such that nulling the dominant gain source of this device renders the whole network to be passive.

Stability Analysis of Differential Feedback Circuits

==========

A balanced fully differential feedback circuit is illustrated below:



The feedback loops are broken at X1 and X2, with x1in and x2in being the input side nodes, x1out and x2out being the output side nodes. The following subcircuit connects these four nodes 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
DMloopgain stb probe=diffprobe_inst.vinj
```

and the following analysis measures the common-mode loop gain:

```
CMalterv alter dev=diffprobe_inst.evinj param=gain value=1
CMalteri alter dev=diffprobe_inst.fiinj param=gain value=1
CMloopgain stb probe=diffprobe_inst.vinj
```

Parameter Index

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

Sweep Analysis (sweep)

Description

The `sweep` analysis sweeps a parameter executing the list of analyses (or multiple analyses) for each value of the parameter. The swept parameter can be circuit temperature, a device instance parameter, a device model parameter, a netlist parameter, or a subcircuit parameter for a particular subcircuit instance.

A set of parameters can be swept simultaneously, using the `paramset` parameter. The other sweep interval or variable parameters cannot be specified with the `paramset` parameter. Do `spectre -h paramset` for information on defining a paramset.

Within a sweep statement, you can specify analyses statements. These statements should be bound within braces. The opening brace is required at the end of the line defining the sweep. Sweep statements can be nested.

You can sweep the circuit temperature by giving the parameter name as `param=temp` with no `dev`, `mod`, or `sub` parameter. You can sweep a top-level netlist parameter by giving the parameter name with no `dev`, `mod`, or `sub` parameter. You can sweep a subcircuit parameter for a particular subcircuit instance by specifying the subcircuit instance name with the `sub`

parameter and the subcircuit parameter name with the `param` parameter. The same can be done using `dev` for the device instance name or `mod` for the device model name.

After the analysis has completed, the modified parameter returns to its original value.

Definition

```
Name sweep parameter=value ...
```

Parameters

Sweep interval parameters

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

Sweep variable parameters

10 `dev`	Device instance whose parameter value is to be swept.
11 `sub`	Subcircuit instance whose parameter value is to be swept.
12 `mod`	Model whose parameter value is to be swept.
13 `param`	Name of parameter to sweep.
14 `paramset`	Name of parameter set to sweep.

Annotation parameters

15 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title` or `sweep`.
16 `title`	Analysis title.

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

Example:

```
swp sweep param=temp values=[-50 0 50 100 125] {
    oppoint dc oppoint=logfile
}
```

Parameter Index

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

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	`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`.
10	`annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
11	`title`	Analysis title.
12	`oppoint=no`	Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if operating point is computed in the previous analysis

and is unchanged.
Possible values are `no`, `screen`, `logfile`, or `rawfile`.

`13 prevoppoint=yes` Use 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 following each parameter name indicates where to find the description of that parameter.

annotate 10	prevoppoint 13	smoothing 4	vel 5
oppoint 12	readns 8	start 3	
points 6	restart 9	stop 1	
ports 7	settling 2	title 11	

Transient Analysis (tran)

Description

This analysis computes the transient response of a circuit over the interval from `start` to `stop`. The initial condition is taken to be the DC steady-state solution if not otherwise given.

Definition

```
Name tran parameter=value ...
```

Parameters

Simulation interval parameters

1 `stop (s)` Stop time.

2 `start=0 s` Start time.

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

4 `autostop=no` If yes, the analysis is terminated when all event-type meaurement expressions have been evaluated. Event-type expressions use thresholding, event or delay type functions. Possible values are `no` or `yes`.

Time-step parameters

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

6 `step=0.001 (stop-start) s`
Minimum time step used by the simulator solely to maintain the aesthetics of the computed waveforms.

Initial-condition parameters

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

8 `skipdc=no` If yes, there will be no dc analysis for transient.
Possible values are `no`, `yes`, `waveless`, `rampup`, `autodc`, or `sigrampup`.

9 `readic` File that contains initial condition.

Convergence parameters

10 `readns` File that contains estimate of initial transient solution.

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

State-file parameters

12 `write`	File to which initial transient solution is to be written.
13 `writefinal`	File to which final transient solution is to be written.
14 `ckptperiod`	Checkpoint the analysis periodically using the specified period.
15 `saveperiod`	Save the tran analysis periodically on the simulation time.
16 `saveclock=1800 s`	Save the tran analysis periodically on the wall clock time.
17 `savetime=[...]`	Save the analysis states into files on the specified time points.
18 `savefile`	Save the analysis states into the specified file.
19 `recover`	Specify the file to be restored.

Integration method parameters

20 `method`	Integration method. Default derived from `errpreset`. Possible values are `euler`, `trap`, `traponly`, `gear2`, `gear2only`, or `trapgear2`.

Accuracy parameters

21 `errpreset`	Selects a reasonable collection of parameter settings. Possible values are `liberal`, `moderate` or `conservative`.
22 `relref`	Reference used for the relative convergence criteria. Default derived from `errpreset`. Possible values are `pointlocal`, `alllocal`, `sigglobal`, or `allglobal`.
23 `lteratio`	Ratio used to compute LTE tolerances from Newton tolerance. Default derived from `errpreset`.
24 `fastbreak=no`	If yes, VHDLAMS Break statement is handled using faster Verilog method. Possible values are `no` or `yes`.

25 `d2aminstep=0` Minimum stepsize that can be taken when there is a D2A event. If this is zero then the simulators min step size will be chosen.

26 `fastcross=discrete`
Using limited threshold reject method for fast cross detection. Possible values are `no`, `yes`, `discrete`, or `cm`.

27 `transres=1e-9 stop s`
Transition resolution. The transient analysis attempts to stop at corners of input waveforms. (ex. corners of rising/falling edge of a pulse). If such events occur within a time less than transres, the analysis will combine the events into one and force only one time point. The rest of the steps will be determined by error control. This may lead to loss of detail.

28 `lteminstep=0.0 s` Local truncation error will be ignored if the step size is less than lteminstep..

Annotation parameters

29 `stats=no` Stats parameter is not supported. Please use annotate. Possible values are `no` or `yes`.

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

31 `title` Analysis title.

Output parameters

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

33 `nestlvl` Levels of subcircuits to output.

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

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

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

37 `skipcount` Save only one of every skipcount points.

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

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

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

41 `flushpoints` Flush outputs after number of calculated points.

42 `flushtime (s)` Flush outputs after real time has elapsed.

43 `flushofftime (s)` Time to stop flushing outputs.

44 `infoname` Name of info analysis to be performed at each time point in the `infotimes` array.

45 `infotimes=[...] s` Times when the analysis specified by `infoname` is performed.

46 `acnames=[...]` Names of ac, noise, sp, or xf analyses to be performed at each time point in the `actimes` array.

47 `actimes=[...] s` Times when analyses specified in `acname` array are performed.

Newton parameters

48 `maxiters=5` Maximum number of iterations per time step.

49 `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

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

Transient noise parameters

51 `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 1/noisefmax.

52 `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 .

53 `noiseseed` Seed for the random number generator. Should be positive integer. Specifying the same seed allows you to reproduce a previous experiment.

54 `noisefmin (Hz)` If specified, the power spectral density of the noise sources will depend on 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.

55 `noisetmin (s)` Time interval between noise source updates. Default is 1/ noisefmax. Smaller values will produce smoother noise signals, but will reduce time integration step.

56 `noiseupdate=fmax` Forces evaluation of bias-dependent device noise sources at each time step, even if it is smaller than noisetmin value. . Possible values are `fmax` or `step`.

Dynamic parameters

57 `param` Name of parameter to be updated to different value with time during tran. Users can use param=isnoisy with param_vec=[...] to turn On and Off 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 noise is ON (param value is 1) from 100ns to 500ns and OFF from 500ns to stop time.
58 `paramset`	Name of dynamic parameter set.
59 `param_vec=[...]`	The time_value points for the param=name.
60 `param_file`	The file that contains the time_value points for the param=name.
61 `sub`	Subcircuit instance for the sunckt instance parameter given in param=name.
62 `mod`	Device model for the model parameter given in param=name.
63 `dev`	Device instance for the instance parameter given in param=name.
64 `param_step`	Defines how often to update the dynamic parameter values. If param_step=0, it updates the parameter value on given time point.

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

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

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

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

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

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

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

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

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

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

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

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

`skipdc=waveless`: Same initial solution as `skipdc=yes`, but the waveform production in the time-varying independent sources is disabled during the transient analysis. Independent source values are fixed to their initial values (not their DC values).

`skipdc=rampup`: Independent source values start at 0 and ramp up to their initial values in the first 10% of the analysis interval. After that their values remain constant. Zero initial solution is used.

`skipdc=autodc`: Same as `skipdc=waveless` if a nonzero initial condition is specified. Otherwise, same as `skipdc=ramup`.

`skipdc=sigrampup`: Independent source values start at 0 and ramp up to their initial values in the first phase of the simulation. Unlike `skipdc=rampup`, the waveform production in the time-varying independent source is enabled after the rampup phase. The rampup

simulation is from If the `start` parameter is not specified, the default `start` time is set to -0.1*`stop`.

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

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

When you simulate the same circuit many times, we suggest that you use both the `write` and `readns` parameters and give the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.

Nodesets and initial conditions have similar implementation but produce different effects. Initial conditions actually define the solution, whereas nodesets only influence it. When you simulate a circuit with a transient analysis, Spectre forms and solves a set of differential equations. However, differential equations have an infinite number of solutions, and a complete set of initial conditions must be specified in order to identify the desired solution. Any initial conditions you do not specify are computed by the simulator to be consistent. The transient waveforms then start from initial conditions. Nodesets are usually used as a convergence aid and do not affect the final results. However, in a circuit with more than one solution, such as a latch, nodesets bias the simulator towards finding the solution closest to the nodeset values.

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

`method=euler`: Backward-Euler is used exclusively.

`method=traponly`: Trapezoidal rule is used almost exclusively.

`method=trap`: Backward-Euler and the trapezoidal rule are used.

`method=gear2only`: Gears second-order backward-difference method is used almost exclusively.

`method=gear2`: Backward-Euler and second-order Gear are used.

`method=trapgear2`: Allows all three integration methods to be used.

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

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

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

`relref=pointlocal`: Compares the relative errors in quantities at each node to that node alone.

`relref=alllocal`: Compares the relative errors at each node to the largest values found for that node alone for all past time.

`relref=sigglobal`: Compares relative errors in each of the circuit signals to the maximum for all signals at any previous point in time.

`relref=allglobal`: Same as `relref=sigglobal` except that it also compares the residues (KCL error) for each node to the maximum of that nodes past history.

The `errpreset` parameter lets you adjust the simulator parameters to fit your needs quickly. You can set `errpreset` to `conservative` if the circuit is very sensitive, or you can set it to `liberal` for a fast. but possibly inaccurate, simulation. The setting `errpreset=moderate` suits most needs.

The effect of `errpreset` on other parameters is shown in the following table. In this table, `T= stop - start`.

errpreset	reltol	relref	method	maxstep	lteratio
liberal	* 10	sigglobal	gear2	Interval/10	3.5

moderate sigglobal traponly Interval/50 3.5

conservative * 0.1 alllocal gear2only Interval/100 10.0

The default value for `errpreset` is moderate.

The value of `reltol` is increased or decreased from its value in the options statement, but it is not allowed to be larger than 0.01. Spectre sets the value of `maxstep` so that it is no larger than the value given in the table. Except for `reltol` and `maxstep`, `errpreset` does not change the value of any parameters you have explicitly set. The actual values used for the transient analysis are given in the log file.

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

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

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

The skipping method is controlled by `skipcount`. If this is set to N, then only every Nth point is saved.

The parameters `skipstart` and `skipstop` apply to both data reduction methods. Before `skipstart` and after `skipstop`, Spectre saves all computed data.

If you do not want any data saved before a given time, use `outputstart`. If you do not want any data saved after a given time, change the `stop` time.

Dynamic Parameters during Transient Analysis

The parameters defined in the `Dynamic parameters` section allow users to change temperature,design parameters or some option parameters (reltol, residualtol, vabstol, iabstol, 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.

Users can also define time value pairs in a file and give the file name though 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 your comment line starts with ; at the beginning of a line. tscale is keyword and tscale_value is a value such as 1.0e-6, 1.0e-9 etc. and is applied to each time point under the time column. time and value are two key words to identify the time and value columns.The values under 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 paramater.

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 and noise is ON from 100ns to 500ns,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, since every signal has the same x value. If `compression=yes`, Spectre writes data to the output file only if the signal value changes by at least 2*the convergence criteria. In order to save data for each signal independently, x axis information corresponding to each signal must be saved. If the signals stay at constant values for large periods of the simulation time, setting `compression=yes` results in a smaller output data file. If the signals in your circuit move around a lot, setting `compression=yes` results in a larger output data file.

Parameter Index

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

Transfer Function Analysis (xf)

Description

The transfer function analysis linearizes the circuit about the DC operating point and performs a small-signal analysis that calculates the transfer function from every independent source in the circuit to a designated output. The variable of interest at the output can be voltage or current.

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

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

The `stimuli` parameter specifies what is used for the inputs for the transfer functions. There are two choices. `stimuli=sources` indicates that the sources present in the circuit should be used. The `xfmag` parameters provided by the sources may be used to adjust the computed gain to compensate for gains or losses in a test fixture. One can limit the number of sources in hierarchical netlists by using the `save` and `nestlvl` parameters.

The transfer functions computed versus output and source types are:

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 series with the terminal. By default, the transfer functions from a small set of terminals are computed. If transfer functions from specific terminals are desired, specify the terminals in the save statement. You must use the `:probe` modifier (ex. Rout:1:probe) or specify `useprobes=yes` on the options statement. If transfer functions from all terminals are desired, specify `currents=all` and `useprobes=yes` on the options statement.

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

Definition

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

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

Parameters

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

Sweep interval parameters

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

Sweep variable parameters

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

Probe parameters

15	`probe`	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`.

Output parameters

22 `stimuli=sources`	Stimuli used for xf analysis. Possible values are `sources` or `nodes_and_terminals`.
23 `save`	Signals to output. Possible values are `all`, `lvl`, `allpub`, `lvlpub`, `selected`, `none`, or `nooutput`.
24 `nestlvl`	Levels of subcircuits to output.
25 `oppoint=no`	Should operating point information be computed, and if so, where should it be sent. Operating point information would not be output if operating point is computed in the previous analysis and is unchanged. Possible values are `no`, `screen`, `logfile`, or `rawfile`.

Convergence parameters

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

Annotation parameters

27 `annotate=sweep`	Degree of annotation. Possible values are `no`, `title`, `sweep`, `status`, or `steps`.
28 `stats=no`	Stats parameter is not supported. Please use annotate. Possible values are `no` or `yes`.
29 `title`	Analysis title.

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

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

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

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

When you simulate the same circuit many times, we suggest that you use both the `write` and `readns` parameters and give the same file name to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.

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:

`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.

Once 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 following each parameter name indicates where to find the description of that parameter.

```
log  9              readns  16          stimuli  22
```

4

Syntax

This chapter discusses the following topics:

Using analogmodel for Model Passing (analogmodel)

Description

`analogmodel` is a reserved word in Spectre that allows you to bind an instance to different masters based on the value of a special instance parameter called modelname. An instance of analogmodel must have a parameter named `modelname` whose string value will be the name of the master this instance will be bound to. The value of `modelname` can be passed into subcircuits.

The `analogmodel` keyword is used by the Cadence Analog Design Environment to enable model name passing through the schematic hierarchy.

Sample Instance Statement:

name [(]node1 ... nodeN[)] analogmodel modelname=mastername [[param1=value1] ...[paramN=valueN]]

name

Name of the statement or instance label.

[(]node1...nodeN[)]

Names of the nodes that connect to the component.

analogmodel

Special device name to indicate that this instance will have its master

name specified by the value of the modelname parameter on the instance.

modelname

Parameter to specify the master of this instance indicated by mastername.

The mastername must either be a valid string identifier or a netlist

parameter that must resolve to a valid master name - a primitive, a model

a subckt, or an AHDL module.

param1

Parameter values for the component. Depending on the master type, these

can either be device parameters or netlist parameters. This is an optional field.

Example:

```
//example spectre netlist to illustrate modelname parameter
simulator lang=spectre
parameters 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
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
```

Behavioural 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.

In this release bsource simulation performance has been improved by compiling the bsource devices. This is explained in more

detail in the bsource compilation section below.

The syntax for bsource is:

name (node1 node2) bsource behav_param param_list

where behav_param can be

c=simple_expr , Capacitance between the nodes

g=simple_expr, Conductance between the nodes

i=generic_expr, Current through bsource

l=simple_expr, Inductance between the nodes

phi=simple_expr, Flux in the bsource device

q=simple_expr, Charge in bsource device

r=simple_expr, Resistance between the nodes

v=generic_expr, Voltage across the nodes

simple_expr is defined as follows:-

A spectre expression containing,

1. netlist parameters
2. current simulation time, $time
3. node voltages, v(a,b), where a and b are nodes in the spectre netlist
   or v(a), which is voltage between node a and ground
4. branch currents, i("inst_id:index"), where inst_id is an instance name
   given in the netlist and index is the port index. The default value for
   index is 0.

generic_expr is defined as a simple_expr or ddt() or idt() of simple_expr

param_list is param_name=value

param_name can be

Multiplicity factor

m The value of m will be default to 1.

Temperature Parameters

tc1 Linear temperature co-efficient. Valid for all behavioural elements.

Default value is 0 1/C.

tc2 Quadratic temperature co-efficient. Valid for all behavioural elements.

Default value is 0 C^-2

tnom Parameters measurement temperature. Valid for all behavioural elements.

Default value is 0.0.

trise Temperature rise for ambient. Valid for all behavioural elements.

Default value is 0.0.

Clipping Parameters

max_val Maximum value of bsource expression. Valid for all behavioural elements, but generally used with i and v elements for clipping the current or voltage between the specified values.

min_val Minimum value of bsource expression. Valid for all behavioural elements, but generally used with i and v elements for clipping the current or voltage between the specified values.

Noise Parameters

af Flicker noise exponent, Valid for r and g elements

Default value is 2.

fexp Flicker noise frequency exponent. Valid for r, g, v, and i elements.

Default value is 1.

isnoisy Specifies whether to generate noise. Valid for r, g, i, and v elements

Valid values are yes and no. Default value is yes.

kf Flicker noise co-efficient. Valid for r and g elements.

white_noise White noise expression. Valid for v and i elements.

flicker_noise Flicker noise expression. Valid for v and i elements.

DC Mismatch Parameters

mr DC-Mismatch parameter. Valid for r only. For algorithm in detail,

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 behavioural expressions;the other parameters must be assigned constant or parametric expressions.

Instance Parameters Supported

bsource supports the following instance parameters for the Spectre primitives.

Resistor isnoisy, m, r, tc1, tc2, trise, kf, af, fexp, ldexp, wdexp, l, w, mr.

Capacitor c, m, tc1, tc2, trise, ic.

Inductor l, m, tc1, tc2, trise.

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

1. Capacitor
   1) cap (F) Capacitance at operating point.
2. Conductor
   1) g (S) Conductance at operating point.
   2) v (V) Voltage at operating point.
   3) i (A) Current through the conductor.
   4) pwr (W) Power dissipation.
3. Current source
   1) v (V) Voltage across the source.
   2) i (A) Current through the source.
   3) pwr (W) Power dissipation.
4. Inductor
   1) ind (H) Inductance at operating point.
   2) i (A) Current at operating point.
5. Charge
   1) cap (F) Capacitance at operating point.
   2) ddt_v (V/s) Voltage gradient at operating point.
6. Resistor
   1) v (V) Voltage at operating point.
   2) i (A) Current through the resistor.
   3) res (Ohm) Resistance at operating point.
   4) pwr (W) Power dissipation.
7. Voltage source

1) v (V) Voltage across the source.

2) i (A) Current through the source.

3) pwr (W) Power dissipation.

Temperature effects on bsource:-

The equation for computing temperature factor is given as follows,

tempFactor = [1 + tc1*(temp-trise+tnom)+tc2*(temp-trise+tnom)^2]

Examples of bsource usage:-

Non-linear resistor/capcacitor/inductor modelling

```
res  (n1 n2)  bsource r=100*(1+(1/2)*v(n1,n2))
res  (n1 n2)  resistor r=100*(1+(1/2)*v(n1,n2))
cap  (n1 n2)  bsource c=1.0e-6*(1+(1/2)*v(n1,n2))
cap  (n1 n2)  capacitor  c=1.0e-6*(1+(1/2)*v(n1,n2))
ind  (n1 n2)  bsource l=0.1*(1+(1/2)*v(n1,n2))
ind  (n1 n2)  inductor l=0.1*(1+(1/2)*v(n1,n2))
```

Charge model for capacitor

```
cap  (n1 n2) bsource q=1.0e-6*v(n1,n2)
```

Voltage and Current (Sinewave) Source

```
vsrc (n1 n2) bsource v=10.0*sin(2*pi*freq*$time)
isrc (n1 n2) bsource i=1.0e-3*sin(2*pi*freq*$time)
```

Current controlled current source

```
vsrc (n1 n2) vsource v=10
cccs1 (n3 n4) bsource i=gain*i("vsrc:0")
```

Current controlled voltage source

```
vsrc (n1 n2) vsource v=10
```

```
    ccvs1 (n3 n4) bsource v=100*i("vsrc:0")
Voltage controlled voltage source
    vsrc (n1 n2) resistor r=100k
    vcvs1 (n3 n4) bsource v=gain*v(n1,n2)
Voltage controlled current source
    vsrc (n1 n2) resistor r=100k
    vccs1 (n3 n4) bsource i=v(n1,n2)/2000.0
Giving voltage clipping limit
    res (n1 n2) bsource r=100*(1+(1/2)*v(n1,n2)) max_val=105 min_val=95
Giving Temperature Co-efficient for Resistor
    res (n1 n2) bsource r=100  tc1=0.01 tc2=0.003 trise=10 tnom=30
Giving DC mismatch parameter for Resistor
    res (n1 n2) bsource r=100  mr=0.3
Doing Altergroup with bsource
    vsrc1 (n1 n2) bsource v=10*sin(2*pi*freq1*$time)
      vsrc2 (n3 n4) bsource v=10*cos(2*pi*freq2*$time)
      cccs1 (n5 n6) bsource i=gain*i("vsrc1:0")
      res   (n5 n6) bsource r=100*(1+(1/2)*v(n5,n6))
      tran1 tran stop=1u
      altAnal altergroup {
      cccs1 (n5 n6) bsource i=gain*i("vsrc2:0")
      res   (n5 n6) bsource r=100*(1+(1/3)*pow(v(n5,n6),2))
      }
      tran2 tran stop=1u
```

Note: For resistor/capacitor/inductor, spectre netlist could directly be given with non-linear equation. However, they will internally be treated as behavioural source.

bsource compilation

==========

In this release the performance of bsource devices has been improved by performing a one time

compilation step. The performance improvement obtained is proportional to the complexity of the

bsource expression. Following the initial compilation, recompilation will only be performed if

the bsource expression is changed.

Bsource compilation is enabled by default. If you are making frequent changes to bsource expressions

used in your design, the overhead of the compilation step may become an issue. To turn off compilation

set the CDS_AHDLCMI_ENABLE shell environment variable to NO e.g:

```
setenv CDS_AHDLCMI_ENABLE NO
```

To re-enable bsource compilation set the CDS_AHDLCMI_ENABLE to YES e.g:

```
setenv CDS_AHDLCMI_ENABLE YES
```

or undefine the CDS_AHDLCMI_ENABLE environment variable e.g:

```
unsetenv CDS_AHDLCMI_ENABLE
```

Checkpoint - Restart (checkpoint)

Description

Spectre has the ability to save checkpoint files while the analyses are running, and to restart an analysis from its checkpoint file. Checkpoint files can be generated in several ways:

1) Periodically based on real time (wall clock time).

2) Asynchronous UNIX signals.

3) By other methods unique to the analyses.

To generate checkpoint files periodically based on real time, set the Spectre option `ckptclock` to the time interval in seconds that you want checkpoints. This option is turned on by default with a value of 1800 seconds (30 minutes). Spectre will delete the checkpoint file if the simulation completes normally. If the simulation terminates abnormally, the checkpoint file will not be deleted.

If Spectre receives the UNIX signal USR2, then Spectre will immediately write a checkpoint file. If Spectre receives interrupt signals like QUIT, TERM, INT, or HUP, Spectre will attempt to write a checkpoint file and then exit. After other fatal signals, it may not be possible for Spectre to write a checkpoint file.

The name of the checkpoint file is a combination of the circuit name, the analysis name, and the extension `.ckpt`. For example, if the circuit is named `test1` and the transient analysis is named `timeSweep`, then the checkpoint file will be named `test1.timeSweep.tran.ckpt`.

Spectre keeps only the latest checkpoint file. When a new checkpoint is created, it creates the file under a temporary name. After the file has been successfully written, it deletes the previous checkpoint file and renames the new file.

Currently only the transient analyses supports checkpoint and restart.

Checkpoint

The transient analysis can generate checkpoint files by using the above methods, or by generating a checkpoint file periodically based on the transient simulation time. This is accessed by a transient analysis parameter called `ckptperiod`, which is turned off by default. To enable the checkpoint feature, the argument +checkpoint must be added to the spectre command line.

Restart

To restart an analysis from a checkpoint file, use the +recover option on the Spectre command line. Spectre will look though the requested analyses to see if a checkpoint file exists for any of them. If a checkpoint file for a given analysis does exist, Spectre will skip over any analyses previous to that one, and start the analysis using the information from the file.

Configuring CMI Shared Objects (cmiconfig)

Description

Spectre supports the ability to install devices dynamically from shared objects at run time. CMI Configuration files are used to determine and locate the set of shared objects to be installed. Spectre first reads the default CMI configuration file which specifies the default shared objects provided by Cadence. The configuration file specified by the value of the CMI_CONFIG environment variable is then read. The third configuration file that Spectre reads is ~/.cmiconfig. Finally, the configuration file specified in Spectres -cmiconfig command line argument is read. Each CMI configuration file modifies the existing configuration established by the configuration files read before.

The following commands can be used in a CMI configuration file.

setpath Specifies and resets the search path.

setpath <path> or setpath (<path1> <path2> ... <pathN>)

prepend Adds a path before the current search path.

prepend <path> or prepend (<path1> <path2> ... <pathN>)

append Adds a path after the current search path.

append <path> or append (<path1> <path2> ... <pathN>)

load Add a shared object to the list of shared objects to load.

loads [path/]<shared_object_name>

unload Removes a shared object to the list of shared objects to load.

unload <shared_object_name>

For example, given the following CMI configuration file

```
append /hm/spectre_dev/tools.sun4v/spectrecmi/lib/cmi/1.0
load libbjtx+tfet.so
load libmosx.so
```

The shared objects `libbjtx+tfet.so` and `libmosx.so` are loaded

from `/hm/spectre_dev/tools.sun4v/spectrecmi/lib/cmi/1.0` in addition to the default shared objects provided by Cadence.

Built-in Mathematical and Physical Constants (constants)

Description

Spectre supports the following list of built-in mathematical and physical constants:

M_ is a mathematical constant

M_E 2.7182818284590452354 exp(1) = e

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 57.29577951308232087720 number of degrees per radian

P_ is a physical constant

P_Q	1.6021918e-19	charge of electron in coulombs
P_C	2.997924562e8	speed of light in vacuum in meters/sec
P_K	1.3806226e-23	Boltzmans constant in joules/kelvin
P_H	6.6260755e-34	Plancks constant in joules*sec
P_EPS0	8.8541879239442001396 8e-12	permittivity of vacuum in farads/meter
P_U0	(4.0e-7 * M_PI)	permeability of vacuum in henrys/meter
P_CELSIUS0	273.15	zero celsius in kelvin

These constants can be used in expressions, or anywhere that a numeric value of expression is expected.

Convergence Difficulties (convergence)

Description

If you are having convergence difficulties, try the following suggestions:

1. 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 gives.

3. Perform sanity check on the parameter values using the parameter range checker (use `+param param-limits-file` as a command line argument) and heed any warnings. Print the minimum and maximum parameter value using the `info` analysis. 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 might 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 in order to avoid regions in the model that might contribute to convergence problems.

DC Convergence Suggestions:

Once you have a solution, write it to a nodeset file using the `write` parameter and read it back in on subsequent simulations using the `readns` parameter.

1. If you have an estimate of what the solution should be, use nodeset statements or a nodeset file and set as many nodes as possible.

2. If convergence difficulties occur when using nodesets or initial conditions, try increasing `rforce` (on options statement).

3. If this is not the first analysis, perhaps the solution from the previous analysis is far from the solution for this analysis. If so, set restart=yes.

4. If simulating a bipolar analog circuit, ensure the region parameter on all transistors and diodes is set correctly.

5. If 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 somewhere in the range of 0.1 to 0.5 using `pivrel` (on options statement).

7. Divide the circuit into smaller pieces and simulate them individually, but ensure that the results will be 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 the transient analysis well beyond the time when all the sources have reached their final value (remember that transient analysis is very cheap when all of the signals in the circuit are not changing) and write the final point to a nodeset file. To make the transient analysis more efficient, set the integration method to backward Euler (`method=euler`) and loosen the local truncation error criteria by increasing `lteratio`, say to 50. Occasionally, this approach will fail or be very slow because the circuit contains an oscillator. Often times the oscillation can be eliminated for the sake of finding the dc solution by setting the minimum capacitance from each node to ground (`cmin`) to a large value.

Transient Convergence Suggestions:

1. 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 jumps in the solution.

encryption (encryption)

Description

1. Define Encryption blocks in the netlist

Keywords (.protect, .unprotect) will be the pair used for defining an encryption block.(protect, unprotect) will be accepted syntax while in Native Spectre mode.The dot keywords will be used in the context of the spice mode.(.protect, .unprotect)/(protect, unprotect) can be abbreviated to(.prot, .unprot)/(prot. unprot), respectively.

If a whole file needs to be encrypted, one way is putting protect atthe beginning of the file and unprotect at the end of the file.The other way is using -all option in spectre encryptor.

Examples of netlist with protected blocks.

Ex1: Protecion of subckts

.protect

.subckt sub1 (...)

....

.ends sub1

.subckt sub2 (...)

.....

.ends sub2

.unprotect

2. Encrypte the netlist

spectre_encrypt is a standalone encryptor. Its usage:

spectre_encrypt [-i input_file] [-o output_file] [-all]where

[-i Input_file] : the netlist to be encrypted.

[-o output _file]: the output file of the encrypted netlist.

[-all]: the whole file will be encrypted in the input netlist.

NOTE:

The include files or library in the netlist need to be encrypted seperartely by the users.

encryptor will not do the encryption on the included files automatomatically.

3. Simulate the encrypted netlist

There is no difference on how to run spectre on an encrypted or unencrypted netlist.Spectre will automatically do the decryption if the netlist is encrypted.

For encrypted netlists, spectre will have protection on the devices, models,signals, and parameters in the encrypted blocks. Any error/warning messages andoutputs on those protected information are suppressed or filtered out.

The following list all the detailed protection.

1) Circuit inventory does not include encrypted parts.

2) info command suppresses all info on encryped parts.

3) Errors on encrypted parts are reported generally as:

Error has occurred within the encrypted block (no details are given).

4) If a portion of the model is encrypted, the entire model is encrypted

5) Any command referencing protected elements results in error

reporting those elements dont exist.

6) Protected device and model parameters can not be altered directly

through alter. However, if they depend on other alterable parameters

then protected parameters are recalculated. Altergroup is allowed

to replace protected devices and models.

7) Protected nodes are outputted in encrypted format when ic file

is requested (similarly for nodes in checking point and restart).

8) Customer can NOT use encryption feature if there are models using CMI version (2.0).

4. Output Operating points on protected devices

By default, all informations on the protected devices are suppressed and not visible to users.On customers request, we provide a way for IP providers to control if theywant to expose the operating points of protected devices to the end users for back annotation.

Keywords (visible invisible) or (.visible, .invisible) in spice netlistcontent are defined for the purpose.

The oppoint of the protected devices between visible and invisible will not be suppressed.

For example:

```
prot
 x1 n1 n2 n3 n4  nmos
 visible
  x2  n5 n6 n7 n8  nmos
  x3  n9 n10 n10 n8 pmos
 invisible
 X4   n11 n12 n13 n13 pmos
 unprot
```

The export feature is not supported. It is designated for internal use only. (export)

Expressions (expressions)

Description

An expression is a construct that combines operands with operators to produce a result that is a function of the values of the operands and the semantic meaning of the operators. Any legal operand is also an expression in itself. Legal operands include numeric constants and references to top-level netlist parameters or subcircuit parameters. Calls to algebraic and trigonometric functions are also supported. The complete lists of operators, algebraic, and trigonometric functions are given after some examples.

Examples:

```
simulator lang=spectre

parameters p1=1 p2=2            // declare some top-level parameters

r1 (1 0) resistor r=p1          // the simplest type of expression

r2 (1 0) resistor r=p1+p2       // a binary (+) expression

r3 (1 0) resistor r=5+6/2       // expression of constants, = 8

x1 s1 p4=8      // instantiate a subcircuit, defined in the following lines

subckt s1

parameters p1=4 p3=5 p4=6       // subcircuit parameters

    r1 (1 0) resistor r=p1              // another simple expression

    r2 (1 0) resistor r=p2*p2           // a binary multiply expression

    r3 (1 0) resistor r=(p1+p2)/p3      // a more complex expression

    r4 (1 0) resistor r=sqrt(p1+p2)     // an algebraic function call

    r5 (1 0) resistor r=3+atan(p1/p2)   // a trigonometric function call
```

```
   r6 (1 0) RESMOD r=(p1 ? p4+1 : p3)  // the ternary operator

ends

//  a model card, containing expressions

model RESMOD resistor tc1=p1+p2 tc2=sqrt(p1*p2)

//  some expressions used with analysis parameters

time_sweep tran start=0 stop=(p1+p2)*50e-6 // use 5*50e-6 = 150 us

//  a vector of expressions (see notes on vectors below)

dc_sweep dc param=p1 values=[0.5 1 +p2 (sqrt(p2*p2)) ]  // sweep p1
```

Where Expressions Can Be Used:

The Spectre native netlist language allows expressions to be used where numeric values are expected on the right-hand side of an "=" sign, or within a vector, where the vector itself is on the right-hand side of an "=" sign. Expressions can be used when specifying device or analysis instance parameter values (for example specifying the resistance of a resistor or the stop time of a transient analysis, as outlined in the preceding example), when specifying model parameter values in model cards (for example specifying "bf=p1*0.8" for a bipolar model parameter, bf), or when specifying initial conditions and nodesets for individual circuit nodes.

Operators:

The following operators are supported, listed in order of decreasing precedence. Parentheses can be used to change the order of evaluation. For a binary expression like "a+b", "a" is the first operand and "b" is the second operand. All operators are left associative, with the exceptions of the "to the power of" operator (**) and the ternary operator (? :), which are right associative. For logical operands, any nonzero value is considered true. The relational and equality operators return a value of 1 to indicate true or 0 to indicate false. There is no short circuiting of logical expressions involving && and ||.

Operator	Symbol	Value
Unary +, Unary -	+, -	Value of operand, negative of operand.
To the power of	**	First operand to raised to power of second operand.
Multiply, Divide	*, /	Product, Quotient of operands.

Operator	Symbol	Value
Binary Plus/Minus	+, -	Sum, Difference of operands.
Shift	<<, >>	First operand shifted left (, right) by number of bits specified by second operand.
Relational	<, <=, >, >=	Less than, less than or equal, greater than, greater than or equal.
Equality	==, !=	True if operands are equal, not equal.
Bitwise AND	&	Bitwise AND (of integer operands).
Bitwise Exclusive NOR	~^ (or ^~)	Bitwise Exclusive NOR (of integer operands).
Bitwise OR	|	Bitwise OR (of integer operands).
Logical AND	&&	True only if both operands true.
Logical OR	||	True if either operand is true.
Ternary Operator	(cond) ? x : y	Returns x if ‘cond’ is true, y if not where x and y are expressions.

Algebraic and Trigonometric Functions:

The trigonometric and hyperbolic functions expect their operands to be specified in radians. The atan2() and hypot() functions are useful for converting from Cartesian to polar form.

Function	Description	Domain
log(x)	Natural logarithm	x > 0
log10(x)	Decimal logarithm	x > 0
exp(x)	Exponential	x < 80
sqrt(x)	Square Root	x > 0
min(x,y)	Minimum value	All x, all y
max(x,y)	Maximum value	All x, all y
abs(x)	Absolute value	All x
pow(x,y)	x to the power of y	All x, all y
int(x)	integer value of x	All x

floor(x) largest integer <= x All x

ceil(x) smallest integer >= x All x

fmod(x,y) floating point modulus All x, all y, except y=0

sin(x) Sine All x

cos(x) Cosine All x

tan(x) Tangent All x, except

x = n*(pi/2), where n odd

asin(x) Arc-sine -1 <= x <= 1

acos(x) Arc-cosine -1 <= x <= 1

atan(x) Arc-tangent All x

atan2(x,y) Arc-tangent of x/y All x, all y

hypot(x,y) sqrt(x*x + y*y) All x, all y

sinh(x) Hyperbolic sine All x

cosh(x) Hyperbolic cosine All x

tanh(x) Hyperbolic tangent All x

asinh(x) Arc-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 the list of these constants

Using Expressions in Vectors:

Expressions can be used as vector elements, as in the following example:

dc_sweep dc param=p1 values=[0.5 1 +p2 (sqrt(p2*p2))] // sweep p1

Note that when expressions are used within vectors, anything other than constants, parameters, or unary expressions (unary +, unary -) must be surrounded by parentheses. Vector elements should be space separated for clarity, though this is not mandatory. The preceding "dc_sweep" example shows a vector of four elements, namely 0.5, 1, +p2, and sqrt(p2*p2). Note that the square root expression is surrounded by parentheses.

User Defined Functions (functions)

Description

Spectres user-defined function capability allows you to build upon the provided set of built-in mathematical and trigonometric functions. You can write your own functions, and call these functions from within any expression. The syntax for calling a user-defined function is the same as the syntax for calling a built-in algebraic or trigonometric function. Note that user-defined functions must be defined before they are referenced (called). Arguments to user-defined functions will be taken as real values, and the functions will return real values. A user-defined function may contain only a single statement in braces and this statement must return an expression (which will typically be an expression involving the function arguments). The return expression may reference the built in parameters `temp` and `tnom`. User-defined functions must be declared at the top level only, and must not be declared within subcircuits. User-defined functions may be called from anywhere that an expressions can be currently used in Spectre. User-defined functions may call other functions (both user-defined and built-in), however any user-defined function will need to be declared before it can be called. User-defined functions can override built-in mathematical and trigonometric functions.

NOTE: Only real values for arguments and return values are supported in this release.

See `spectre -h expressions` for a list of built-in algebraic and trigonometric functions.

Definition

```
real myfunc( [real arg1, ...real argn] ) {
```

Examples:

```
real myfunc( real a, real b ) {
   return a+b*2+sqrt(a*sin(b));
}
```

An example of a function calling a previously defined function follows

```
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)
```

Global Nodes (global)

Description

The global statement allows a set of nodes to be designated as common to the main circuit and all subcircuits. Thus, components inside subcircuits can be attached to global nodes even though the subcircuits terminals are not attached to these nodes.

Any number of global nodes may be specified using the global statement. To do this, follow the keyword global with a list of the node names that you wish to declare as global. The first node name that appears in this list is taken to be the name of the ground node. Ground is also known as the datum or reference node. If a global statement is not used, 0 is taken to be the name of the ground node.

At most one global statement is allowed and, if present, it must be the first statement in the file (however, you can have `simulator lang=spectre` statement, before the global statement so that you can use mixed case names for the node names). Ground is always treated as global even if a global statement is not used.

Definition

```
global <ground
```

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 to define a model for the

IC component package, and a buffer model for each pin. IBIS standard also allows to describe a board level components 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 Electrical Board Description file with `.ebd` extension. Component pins, buffers, and package descriptions are in a separate IBIS file with extension `.ibs`. Additional Package file with extension `.pkg` can be used to describe advanced package models.

IBIS files can be referenced in Spectre netlist using ibis_include statement:

ibis_include "DRAM.ibs" [options]

IBIS file "DRAM.ibs" will be translated into Spectre netlist format using `ibis2subckt` utility. The output file, "DRAM.scs", containing subcircuit definitions for each IBIS component, board, and package found in the input IBIS file, will be includes in the netlist. The list of options may consist of: corner={typ|min|max}, swsel=<int>, mdsel=<int>. These options are transferrerd to `ibis2subckt`. They are used to select IBIS buffer model corner, change position of series-switch models, and chose 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 same as component name appended with `_ibis` suffix. Subcircuit terminals are component pins, in the same order they are listed in 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, then 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:

x_ic (p1 p3 s3_in p4 p5 s5_in s5_out s5_en p2) IC_ibis

Ibis2subckt can be also used as a stand-alone utility with following command line arguments:

ibis2subckt -in <IBIS files> -out <subckt file> -corner {typ|min|max} -swsel <int> -mdsel <int>

Default corner is typ; model selector - 1; switch selector - 1

Initial Conditions (ic)

Description

The `ic` statement is used to provide initial conditions for nodes in the transient analysis. It can occur multiple times in the input and the information provided in all the occurrences is collected. Initial conditions will only be accepted for inductor currents and node voltages where the nodes have a path of capacitors to ground. For more information, read the description of transient analysis. It should be noted that specifing `cmin` for a transient analysis, will not satisify the condition that a node has a capacitive path to ground.

Definition

```
ic <node=value
```

This 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 either 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 parameters control pattern matching. `depth` controls the depth of the pattern matching and by default matches signals at all hierarchical levels. `sigtype` with 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` will not be saved. When `subckt` is given, the wildcard patters in the save statement and the depth of the pattern matching must be relative to the subcircuit master.

The concept of nodes for the save 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 a inductor or the voltage of the internal node in a diode. Topological nodes can be either at the top-level or in a subcircuit.

For example,

```
ic 7=0 out=1 OpAmp1.comp=5 L1:1=1.0u
```

where 7=0 implies that node 7 should start at 0V, node out should start at 1V, node comp in subcircuit OpAmp1 should start at 5V, and the current through the first terminal of L1 should start at 1uA.

The Structural if-statement (if)

Description

The structural if-statement can be used to conditionally instantiate other instance statements.

Definition

```
if <condition
```

The condition is a boolean expression based on the comparisons of various arithmetic expressions which are evaluated during circuit hierarchy flattening. The statement1 and statement2 fields can be ordinary instance statements, if-statements, or a list of these within braces ({}). (Note that ordinary instance statements need a newline to terminate them.) The else part is optional. When if-statements are nested without braces, an else matches the closest previous unmatched if at the same level.

It is possible to have duplicate instance names within the if statement under strict topological conditions. These are:

* references to instance with duplicate names is only possible within a structural if statement which has both an "if" part and an "else" part.
* both the "if" part and the "else" part must be either a simple one-statement block, or another structural if statement to which these same rules apply.
* The duplicate instances must have the same number of terminals and be bound to the same list of nodes.
* The duplicate instances must refer to the same primitive or model.
* Where duplicate instances refer to a model, the underlying primitive must be the same.

This feature allows automatic model selection based on any netlist or subcircuit parameter. As an example, consider using Spectres inline subcircuits and structural if statement to implement automatic model selection based on bipolar device area. Here, the duplicate instances are the inline components.

```
model npn_default bjt is=3.2e-16 va=59.8

model npn10x10 bjt is=3.5e-16 va=61.5

model npn20x20 bjt is=3.77e-16 va=60.5

// npn_mod choses scaled models binned on area!

// if ( area < 100e-12 ) use model npn10x10

// else if ( area <  400e-12 ) use model npn20x20

// else use model npn_default

inline subckt npn_mod (c b e s)

  parameters area=5e-12

  if ( area < 100e-12 ) {

      npn_mod (c b e s) npn10x10  // 10u * 10u, inline device

  } else if ( area < 400e-12 ) {

      npn_mod (c b e s) npn20x20  // 20u * 20u, inline device

  } else {

      npn_mod (c b e s) npn_default       // 5u * 5u, inline device

  }

ends npn_mod

q1 (1 2 0 0) npn_mod area=350e-12         // gets 20x20 model

q2 (1 3 0 0) npn_mod area=25e-12          // gets 10x10 model

q3 (1 3 0 0) npn_mod area=1000e-12        // gets default model
```

Include File (include)

Description

File inclusion allows the circuit description to be spread over several files. The include statement itself is replaced by the contents of the file named. An included file may also contain include statements. If the name given is not an absolute path specification, then it is taken relative to the directory of the file currently being read.

In order to read existing SPICE library and model files, Spectre automatically switches to SPICE input mode when it opens an include file. Thus, all files that use the Spectre native language must begin with a `simulator lang=spectre` statement. The one exception is files that end with a ".scs" file extension which are treated specially and are read in Spectre input mode. This language mode treatment applies to files included by both Spectres include statement, and CPPs #include statement.

After reading the include file, Spectre restores the language processing mode to what it was before the file was included, and continues reading the original file starting at the line after the include statement. Lines cannot be continued across file boundaries.

The CPP #include statement differs from Spectres include statement in that CPP macro processing will not be performed on files included by Spectre, but will be performed on files included by CPP. If your netlist contains a #include statement, you must run CPP to perform this inclusion, otherwise an error will occur.

If the file to be included cannot be found in the same directory as the including file, Both Spectres include and CPPs #include will search for the file to be included along the search path specified by the -I command line arguments.

Spectres include statement allows you to embed special characters in the name of the file to be included. Spectres include statement will automatically expand the "~" character to the users home directory, and will expand environment variables and % codes, such as

include "~/models/${SIMULATOR}_pd/npn.scs"

which will look in the directory given by the environment variable SIMULATOR followed by "_pd", which is under the "models" directory, in the users home directory. Note: These special character features are not available using CPPs #include statement.

Definition

```
include "filename"
```

Spectre Netlist Keywords (keywords)

Description

The following lists 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 obey the following rules when using them, or you may result in an error.

1. Netlists keywords cannot be used as instance names, subckt names, model names, or function names.

2. 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
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
altergroup	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
local	Netlist Keyword

march	Netlist Keyword
model	Netlist Keyword
nodeset	Netlist Keyword
parameters	Netlist Keyword
paramset	Netlist Keyword
plot	Netlist Keyword
print	Netlist Keyword
prot	Netlist Keyword
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
unprot	Netlist Keyword
vary	Netlist Keyword
visible	Netlist Keyword

Library - Sectional Include (library)

Description

Library inclusion allows the circuit description to be spread over several files. The library statement itself is replaced by the contents of the section of the library file. A library section may also contain library reference statements. If the file name given is not an absolute path specification, then it is taken relative to the directory of the file currently being read.

There are two kinds of library statements. One that references a library section, and one that defines a library section. The definition of a library section is prohibited in the netlist.

In order to read existing SPICE library and model files, Spectre automatically switches to SPICE input mode when it opens a library file. Thus, all files that use the Spectre native language must contain a `simulator lang=spectre` statement within each section of the library or the file can have a `scs` filename extension. After reading the library section, Spectre restores the language processing mode and continues reading the original file starting at the line after the library statement. Lines cannot be continued across file boundaries.

Spectre allows only one library per file, but a library may contain multiple sections (typically one section per process corner for example.)

Definition

```
Inside netlist (reference library section)
```

Sample Library:

```
library corner_lib

section tt

  model nch bsim3v3 type=n mobmod=1 capmod=2 version=3.1

  + xj=1.7e-7 vsat=7.99e4 at=3.6e4 a0=0.799 ags=0.4

  + a1=0 a2=1 keta=-0.05 nch=2.8e17 ngate=1.31e20 k1=0.74

  model pch bsim3v3 type=p mobmod=1 capmod=2 version=3.1

  + xj=1.7e-7 vsat=1.38e5 at=1e5 a0=1.3 ags=0.3
```

```
  + a1=1.1e-4 a2=1 keta=0 nch=4.1e17 ngate=7.6e19 k1=0.88
  model knpn bjt is=10e-13 bf=170 va=58.7 ik=5.63e-3 rb=rbn rbm=86
  + re=3.2 cje=0.25e-12 pe=0.76 me=0.34 tf=249e-12 cjc=0.34e-12 pc=0.55
  + mc=0.35 ccs=2.4e-12 ms=0.35 ps=0.53 rc=169
  model 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
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
```

Node Sets (nodeset)

Description

The `nodeset` statement is used to provide an initial guess for nodes in any DC analysis or the initial condition calculation for the transient analysis. It can occur multiple times in the input, the information provided in all the occurrences is collected. For more information, read the description of DC analysis.

Definition

```
nodeset <node=value
```

This statement takes a list of signals followed by some optional parameters as an argument. `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 characters `*` for any string and `?` for any character. The parameters control pattern matching.

depth controls the depth of the pattern matching and by default matches signals at all hierarchical levels. sigtype with 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 will not be saved. When subckt is given, the wildcard patters in the save statement and the depth of the pattern matching must be relative to the subcircuit master.

The concept of nodes for the save 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 a inductor or the voltage of the internal node in a diode. Topological nodes can be either at the top-level or in a subcircuit.

For example,

```
nodeset 7=0 out=1 OpAmp1.comp=5 L1:1=1.0u
```

where 7=0 implies that node 7 should be about 0V, node out should be about 1V, node comp in subcircuit OpAmp1 should be about 5V, and the current through the first terminal of L1 should be about 1uA.

Parameter Soft Limits (param_limits)

Description

The parameter values passed to Spectre components and analysis are subject to both hard and soft limits. If you set a parameter to a value that violates a hard limit, such as giving z0=0 to a transmission line, Spectre issues an error message and quits. If the given parameter value violates a soft limit, only a warning is issued, but Spectre uses the value of the component as given. Hard limits are used to prevent you from using values that would cause Spectre to fail or put a model in an invalid region. Soft limits are used to call attention to unusual parameter values that might have been given mistakenly. If a parameter value violates a soft limit, a message similar to one of the following sample messages is printed:

```
Parameter rb has the unusually small value of 1uOhms.
```

or

```
Parameter 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 provide limits on parameters that do not have built-in limits. To do so,

create a parameter range limits file, and invoke Spectre giving the name of the file after the +param command line option. For example,

```
spectre +param limits-file input-file
```

Limits are given using the following syntax:

```
[PrimitiveName] [model] [LowerLimit <[=]] [|]Param[|] [<[=] UpperLimit]
```

The limits can be given as strict (using <=) or nonstrict (using <). If the limits are strict, there can be no space between < and =. The limits for one parameter are given on one line. There is no way of continuing the specification of the limits for a parameter over more than one line. If a parameter is given more than once, the limits given the last time override earlier limits. The primitive name must be a Spectre primitive name, not a name used for SPICE compatibility. So, for example, mos3 must be used rather than mos. Parameter limits can be written using Spectre native mode metric scale factors. Thus a limit of f <= 1.0e6 could also written as f <= 1M.

Examples

```
mos3      0.5u <= l <= 100u
          0.5u <= w
          0 < as <= 1e-8
          0 < ad <= 1e-8
      model |vto| <= 3
```

Notice that it is not necessary to give the primitive name each time. If not given, it is assumed to be the same as the previous parameter. Upper and lower limits may be given, and if not given there will be no limit on the parameter value. Thus, in the example, if w is less than 0.5um, a warning will be issued, but there is no limit on how large w can be. If a parameter is mentioned, but no limits given, then all limits are disabled for that parameter. Limits are placed on model parameters by giving the model keyword. If the model keyword is not given, the limits are applied to instance parameters. Notice that you can also place upper or lower limits on the absolute value of a parameter. For example,

```
resistor 0.1 < |r| < 1M
```

indicates that the absolute value of r should be greater than 0.1 Ohm and less than 1 MOhm. There can be no spaces between the absolute value symbols and the parameter name.

Examples

```
1 <= 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 and 0.5 itself. The limits are applied separately, thus 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 either a real number, an integer, or an enumeration. To specify the limits of a parameter that takes enumerations use the indices associated with the enumerations. For example, consider the region parameter of the bjt. There are four possible regions: off, fwd, rev, or sat (see `spectre -help bjt`). Each enumeration is assigned a number starting at 0 and counting up. Thus, off=0, fwd=1, rev=2, and sat=3. The specification

```
bjt       3 <= 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"
```

will temporarily suspend the reading of the current file until the contents of `filename` have been read. Include statements may be nested arbitrarily deep with the condition that the operating system may limit the number of files that Spectre may have open at once. Paths in file names are taken to be relative to the directory that contains the current file, not from the directory in which Spectre was invoked.

Spectre can be instructed to always read a parameter limits file by using the SPECTRE_DEFAULTS environment variable. For example, if you put the following in you shell initialization file (.profile for sh, .cshrc for csh)

setenv SPECTRE_DEFAULTS "+param /cds/etc/spectre/param.lmts"

Spectre would always read the specified limits file.

Netlist Parameters (parameters)

Description

The Spectre native netlist language allows parameters to be specified and referenced in the netlist, both at the top-level scope and within subcircuit declarations (run `spectre -h subckt` for more details on parameters within subcircuits).

Definition

```
parameters <param=value
```

Examples:

simulator lang=spectre

parameters p1=1 p2=2 // declare some parameters

r1 (1 0) resistor r=p1 // use a parameter, value=1

r2 (1 0) resistor r=p1+p2 // use parameters in an expression, value=3

x1 s1 p4=8 // "s1" is defined below, pass in value 8 for "p4"

subckt s1

parameters p1=4 p3=5 p4=6 // note: no "p2" here, p1 "redefined"

r1 (1 0) resistor r=p1 // local definition used: value=4

r2 (1 0) resistor r=p2 // inherit from parent(top-level) value=2

r3 (1 0) resistor r=p3 // use local definition, value=5

r4 (1 0) resistor r=p4 // use passed-in value, value=8

r5 (1 0) resistor r=p1+p2/p3 // use local+inherited/local = (4+2/5) = 4.4

ends

time_sweep tran start=0 stop=(p1+p2)*50e-6 // use 5*50e-6 = 150 us

dc_sweep dc param=p1 values=[0.5 1 +p2 (sqrt(p2*p2))] // sweep p1

Parameter Declaration:

Parameters can be declared anywhere in the top-level circuit description or on the first line of a subcircuit definition. Parameters must be declared before they are used (referenced). Multiple parameters can be declared on a single line. When parameters are declared in the top-level, their values must be specified. When parameters are declared within subcircuits, their default values are optionally specified.

Parameter Inheritance:

Subcircuit definitions inherit parameters from their parent (enclosing subcircuit definition, or top-level definition). This inheritance continues across all levels of nesting of subcircuit definitions, that is, if a subcircuit s1 is defined, which itself contains a nested subcircuit definition s2, then any parameters accessible within the scope of s1 are also accessible from within s2. Also, any parameters declared within the top-level circuit description are also accessible within both s1 and s2. However, any subcircuit definition can redefine a parameter that it has inherited. In this case, if no value is specified for the redefined parameter when the subcircuit is instantiated, then the redefined parameter uses the locally defined default value, rather than inheriting the actual parameter value from the parent.

Parameter Namespace:

Parameter names must not conflict with device or analysis instance names, that is, it is not possible to reference a parameter called `r1` if there is an instance of a resistor (or other device or analysis) called `r1`. Parameter names must also not be used where a node name is expected.

Parameter Referencing:

Spectre netlist parameters can be referenced anywhere that a numeric value is normally specified on the right-hand side of an "=" sign or within a vector, where the vector itself is on the right-hand side of an "=" sign. This includes referencing of parameters in expressions (run `spectre -h expressions` for more details on netlist expression handling), as indicated in the preceding examples. You can use expressions containing parameter references when specifying device or analysis instance parameter values (for example specifying the resistance of a resistor or the stop time of a transient analysis, as outlined in the preceding example), when specifying model parameter values in model cards (for example specifying "bf=p1*0.8" for a bipolar model parameter, bf), or when specifying initial conditions and nodesets for individual circuit nodes.

Altering/Sweeping Parameters:

Just as certain Spectre analyses (for example `sweep`, `alter`, `ac`, `dc`, `noise`, `sp`, `xf`) can sweep device instance or model parameters, they can also sweep netlist parameters. Run `spectre -h <analysis>` to see the particular details for any of these analyses, where <analysis> is the analysis of interest.

Temperature as a parameter:

You can use the reserved parameters `temp` and `tnom` anywhere that an expression can be used, including within expressions and user-defined functions. The `temp` parameter always represents the simulator (circuit) temperature, and `tnom` always represents the measurement temperature. All expressions involving `temp` or `tnom` are re-evaluated any time the circuit temperature or measurement temperature changes.

You can also alter or sweep the `temp` and `tnom` parameters using any of the techniques available for altering or sweeping netlist or subcircuit parameters (with the exception of altergroups).

This capability allows you to write temperature dependent models for example, by using `temp` in an equation for a model or instance parameter. For example,

r1 1 0 res r=(temp-tnom)*15+10k // temp is temperature

o1 options temp=55 // causes a change in above resistor r1

Reserved Parameters

The following parameters are reserved, and may not be declared as either top-level parameters or subcircuit parameters: `temp`, `tnom`, `scale`, `scalem`, `freq`, `time`.

Parameter Set - Block of Data (paramset)

Description

A parameter set is a block of data, which can be referenced by a sweep analysis. Within a paramset the first row contains an array of top-level netlist parameters. All other rows contain numbers which are used to alter the value of the parameters during the sweep. Each row represents an iteration of the sweep. This data should be bound within braces. The opening brace is required at the end of the line defining the paramset. The paramset cannot be defined within subcircuits or cannot be nested.

Definition

```
<Name
```

Example:

```
data paramset {
  p1  p2  p3
  1.1 2.2 3.3
  4.4 5.5 6.6
}
```

Tips for Reducing Memory Usage with SpectreRF (rfmemory)

Description

Problem: How can you reduce memory useage when running SpectreRF simulations? What are some things you need to be aware of?

Solution:

The amount of swap/memory that Spectre/SpectreRF requires will depend on several things...

- The analysis type you are running (PSS, QPSS, transient, dc, etc).

  (PSS and QPSS take up a lot of memory by their very nature).

- Your simulator options. Reltol, Vabstol, maxstep, relref are the

  primary options that affect swap/memory.

- The complexity of models being used

- The size of your circuit (number of active devices for example)

- 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 the PSS analysis. Harmonics increases the number of timepoints
  taken by the simulator when the number of harmonics get larger than 10.
  However, this is not a large effect as compared to a 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.
  They control the number of integration intervals QPSS takes, hence
  dominate the memory usage of QPSS.

Here are several things you may wish to try:

---

1. Check to see how much swap and memory you have on your workstation.
   At a minimum, we strongly recommend 2+GB RAM and 5+GB swap...and
   more is better.
   In MMSIM6.0, we now have a Spectre executable with 64 bit addressing. This
   means no 4GB process size limit, but if the circuit is large, the RAM
   requirements could get really high. With an RCX parasitic resimulation,
   64 GB (Yes, GigaBytes) of RAM might be required to simulate a parasitic
   resimulation.
2. Reduce the amount of data points stored by reducing the number of

harmonics saved.

Specify only the individual harmonics that you wish to view. You can do this by selecting, "Array of Coefficients" or "Array of Indices" from output harmonics section of the Choosing Analyses form.

For harmonics, once you exceed 10, the number of timepoints goes up by forcing a smaller maxstep. Since all of the solution matrices at each timepoint need to be saved and at the end of each PSS iteration need to be accessed, the memory requirements can get quite large.

maxacfreq also affects maxstep. Higher maxacfreq (above 40 * PSS beat freq) causes more timepoints, thus more memory. If you use maxacfreq, set it to the highest frequency in your circuit.

If running a Pnoise analysis, choose all the sidebands you can, focussing on the ones that will give you most noise.

3. Use the swapfile option to PSS/QPSS.

The swapfile parameter of the 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 is actually saving 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 fast as possible. If you have access to a system with *lots* of RAM, the 64-bit spectre may be a better way to go.

- If running SpectreRF in Artist, the swapfile option is located in

the PSS/QPSS Choosing Analysis Options form. Go to the swapfile option and enter "<path_to>/some_file_name".

- If running SpectreRF standalone, on the pss analysis line simply add swapfile="<path_to>/some_file_name".

  In either case, make sure there is enough disk space under the <path_to> directory.

  Note: The swapfile is only used during the PSS/QPSS iterations and 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 only requesting the PSS harmonic information that you really need (in item 2 above).

  Your Spectre process may run at 99% until physical memory has run out. Then it uses Unix swap and the process runs at 5% of CPU because of the slow swapping to disk. Using a swapfile can mean the process runs at 50% because larger sections of disk are being swapped, i.e typically 3x (and up to 10x) faster.

4. Reduce the number of nodes/nets you are saving (dont use save all voltages/currents!). However, be aware of this little caveat...If you are running a PSS analysis and then trying to plot the frequency domain current (or power) spectrum, you must set this in the netlist: currents=yes

   Caution: 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 will require too much disk space and too much time to execute. Remember that the PSS analysis saves all of the matrices from each time point, where data is saved at each iteration of the shooting interval, starting from the second. All the data is kept when the initial state equals the final state (within the tolerance parameters). The more time points that you require, the more disk space you use. Here is a rule of thumb: (And remember, your mileage may vary)

   > 30 periods of the largest amplitude tone:

      Use QPSS (followed by QPxx small signal analyses)

   10-30 periods of the largest amplitude tone:

      Use PSS or QPSS. It will depend on your circuit,

      the number of harmonics you choose, number of moderate tones, etc.

   < 10 periods of the largest amplitude tone:

      Use PSS (followed by Pxx small signal analyses).

7. To improve QPSS convergence, a guideline is to try the following things in the order presented (i.e. if A doesnt work, then 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 on stabcycles, see solution 11159516 .

8. The new Spectre front end (+csfe in IC5141 and the default in MMSIM6.0) decreases memory usage and will help solve SpectreRF memory usage issues.
9. Break the circuit into smaller chunks, simulate the chunks individually. Or model more complex parts of the circuit with Verilog-A modules.
10. Run Spectre when no other users are using the same machine if "too many people are running Spectre simulations on the same machine at the same time".

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). The 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]]
```

This 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 either 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 parameters control pattern matching. `depth` controls the depth of the pattern matching and by default matches signals at all hierarchical levels. `sigtype` with 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` will not be saved. When `subckt` is given, the wildcard patters in the save statement and the depth of the pattern matching must be relative to the subcircuit master.

The concept of nodes for the save 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 a inductor or the voltage of the internal node in a diode. Topological nodes can be either at the top-level or in a subcircuit.

For example,

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`, eg 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`, eg 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`, eg 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.

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 was interrupted. Since 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.

1. Users have the option to save circuit information at set intervals, or at multiple points during the transient analysis. If the simulation is unexpectedly halted, the user can restart the transient from any saved timepoint.
2. Users can experiment with different accuracy settings over different transient time periods in order to get the optimal speed/accuracy trade-off.

Requirements for Savestate-Recover

When the simulation is restarted, the following pints must be kept in mind:

- Netlist topology must not be changed. Topology changes or the removal/addition of nodes in the restore file will cause a fatal error.
- You can edit any netlist parameter as long as the circuit topology remains the same.
- The transient analysis stop time must be larger than the time point corresponding to the savestate file.
- Saved state file is binary and platform dependent.
- Savestate-Recover only works for tran analysis and the tran must not be within a Sweep or Monte Carlo analysis

Use Model - Savestate

Savestate is enabled/disabled through Spectre command line option.

```
spectre [+savestate ] [-savestate ] ...
```

where:

`+savestate --` enables Savestate. You may use `+ss` as an abbreviation of `+savestate`.

`-savestate --` disables Savestate.You may use `-ss` as an abbreviation of `-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:

`saveperod`, `saveclock`, and `savetime` define the time points to save the states.

`savefile` defines where the saved states are written to.

If saveperiod given, Spectre generates a saved state file periodically based on the transient simulation time. Only the last saved state file is kept.

If saveclock given, Spectre generates a saved state file periodically based on real time (wall clock time). Its Default is 1800 seconds (30 minutes). If savetime given, Spectre generates a saved state file on each specified time point.

If savefile is not defined, the default file name will be `%C.%A.srf`

where %C is the input `ckt` file name, %A is analysis name.

If multiple save time points are given, i.e `analysisName tran stop=stoptime savetime=[time1 time2 ...] savefile=filename`, the saved state file will be `filename@time1`, `filename@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 terminates immediately.

The `saveclock`, `saveperiod`, and `savetime` parameters should not be specified at the same time. In case more than one parameter is specified, Spectre reads them in the following order: two based on the following priority:

1. saveperiod
2. savetime
3. saveclock

Recover

There are two ways to recover the simulation from the saved state file. One way is to define the recover file in the Spectre command line. The second way 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 the recover. You may use `+rec` as an abbreviation of `+recover`.

`-recover --` disbales the recover. You may use `-rec` as an abbreviation of `-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. The assumption is that when defining `recover=state_file` in a tran statement, users would 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 (start from 0) is added to the raw, such as *.raw#, where # is 0, 1, 2,...

For example, in the first run, spectre `input.scs` Spectre saves the simulation state into a file on a time point.

By default `input.raw` directory is created, When Spectre is run in recover mode, spectre `+recover=saved_state_file` input.scs. A new raw directory named `input.raw0` is created.

The index to the raw directory is increased at each successive recover run. Another use model is that users define multiple tran runs in a netlist. In the first tran, Spectre saves the simulation state on a time point. In the next tran run, simulation is continued from a 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@5.000000e-08"
```

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 ...

`design_parameters_list` = dpar1 dpar2 ...

`analyses_list` = anal1 anal2 ...

The list of the design parameters may include valid instance and model parameters. You can also specify device instances or device models without a modifier. In this case Spectre will attempt to compute sensitivities with respect to all corresponding instance or model parameters. Caution should be exercised in using this option as warnings or errors may be generated since many instance and model parameters cannot be modified. If no design parameters are specified then all the instance and model parameters are added. The list of the output variables for both AC and DC analyses may include node voltages and branch currents. For DC analyses, it also may include device instance operating point parameters.

Examples:

```
sens (q1:betadc 2 Out) to (vcc:dc nbjt1:rb) for (analDC)
```

For this statement DC sensitivities of betadc operating point parameter of transistor `q1` and of nodes `2` and `Out` will be computed with respect to the `dc` voltage level of voltage source `vcc` and the model parameter `rb` for the DC analysis `analDC`. The results will be stored in the output file analDC.sens.dc.

```
sens (1 n2 7) to (q1:area nbjt1:rb) for (analAC)
```

For this statement AC sensitivities of nodes 1, n2, 7 will be computed with respect to the area parameter of transistor `q1` and the model parameter `rb` for each frequency of the AC analysis analAC. The results will be stored in the output file analAC.sens.ac.

```
sens (1 n2 7) for (analAC)
```

For this statement AC sensitivities of nodes 1, n2, 7 will be computed with respect to all instance and model parameters of all devices in the design for each frequency of the AC analysis analAC. The results will be stored in the output file analAC.sens.ac.

```
sens (vbb:p q1:int_c q1:gm 7) to (q1:area nbjt1:rb) for (analDC1)
```

For this statement DC sensitivities of the branch current `vbb:p`, the operating point parameter `gm` of the transistor `q1`, the internal collector voltage `q1:int_c` and the node `7` voltage will be computed with respect to the instance parameter `area` for instance `q1` and the model parameter `rb` for model `nbjt1`.

SpectreRF Summary (spectrerf)

Description

SpectreRF is a optional collection of analyses that are useful for circuits that are driven with a large periodic signal. Examples include mixers, oscillators, switched-capacitor filters, sample-and-holds, chopper stabilized amplifiers, frequency multipliers, frequency dividers, and samplers. They efficiently and directly compute the periodic and quasiperiodic steady-state solution of such circuits. They are capable of computing the large-signal and small-signal behavior, including noise behavior. Thus, SpectreRF is capable of computing the noise figure or intermodulation distortion of a mixer, the phase noise and harmonic distortion of an oscillator, and the frequency-response and noise behavior of a switched-capacitor filter. For more information on the SpectreRF analyses, run `spectre -help analysisName` where analysisName is `pss`, `pac`, `pxf`, `pnoise`, `psp`, `qpss`, `qpac`, `qpxf`, `qpnoise`, `qpsp`, or `envlp`.

Subcircuit Definitions (subckt)

Description

Hierarchical circuit description:

The `subckt` statement is used to define a subcircuit. Subcircuit definitions are simply circuit macros that can be expanded anywhere in the circuit any number of times. When an instance in your input file refers to a subcircuit definition, the instances specified within the subcircuit are inserted into the circuit. Subcircuits may be nested. Thus a subcircuit definition may contain instances of other subcircuits. Subcircuits may also contain component, analysis or model statements. Subcircuit definitions may also be nested, in which case the innermost subcircuit definition can only be referenced from within the subcircuit in which it is defined, and cannot be referenced from elsewhere.

Instances that instantiate a subcircuit definition are referred to as subcircuit calls. The node names (or numbers) specified in the subcircuit call are substituted, in order, for the node names given in the subcircuit definition. All instances that refer to a subcircuit definition must have the same number of nodes as are specified in the subcircuit definition and in the same order. Node names inside the subcircuit definition are strictly local unless declared otherwise in the input file with a global statement.

Subcircuit Parameters:

Parameter specification in subcircuit definitions is optional. In the case of nested subcircuit definitions, any parameters which have been declared for the outer subcircuit definition are also available within the inner subcircuit definition. Any parameters that are specified are referred to by name optionally followed by an equals sign and a default value. If, when making a subcircuit call, you do not specify a particular parameter, this default value is used in the macro expansion. Subcircuit parameters can be used in expressions within the subcircuit consisting of subcircuit parameters,

constants, and various mathematical operators. Run `spectre -h expressions` for more details on Spectre expression handling capability. Run `spectre -h parameters` for more details on how Spectre handles netlist parameters, including subcircuit parameters, and how they inherit within nested subcircuit definitions.

Subcircuits always have an implicitly defined parameter `m`. This parameter is passed to all components in the subcircuit and each component is expected to multiply it by its own multiplicity factor. In this way, it is possible to efficiently model several copies of the subcircuit in parallel. It is an error to attempt to explicitly define `m` on a `parameters` line. Also, because `m` is only implicitly defined, it is not available for use in expressions in the subcircuit.

Inline Subcircuits:

An inline subckt is a special case of a subckt where one of the devices or models instantiated within this subckt does not get its full hierarchical name, but rather inherits the subckt call name itself. An inline subckt is syntactically denoted by the presence of the keyword "inline" before the "subckt". It is called in the same manner as a regular subcircuit. The body of the

inline subcircuit can typically contain one of the following, corresponding to different use models:

* multiple device instances (one of which is the "inline" component)

* multiple device instances, (one of which is "inline")

  and one or more parameterized models

* a single "inline" device instance and a parameterized

  model to which the device instance refers

The "inline" component is denoted by giving it the same name as the inline subcircuit itself. When the subcircuit is flattened, the "inline" component does not take on a hierarchical name such as X1.M1, but rather takes on the name of the subckt call itself, such as X1. Any non-inline components in the subckt take on the regular hierarchical name, just as if the subcircuit were a regular subckt (i.e. not an inline subckt).

Probing the inline device

Spectre allows the following list of items to be saved or probed for primitive devices. These would also apply to devices modeled as the inline components of inline subcircuits:

1. all terminal currents e.g. "save q1:currents"

2. specific (index) terminal current e.g. "save q1:1" (#1=collector)

3. specific (named) terminal current e.g. "save q1:b" ("b"=base)

4. save all operating point info e.g. "save q1:oppoint"

5. save specific operating point info e.g. "save q1:vbe"

6. save all currents and oppoint info e.g. "save q1"

Parameterized Models and Inline Subckts:

Inline subckts can be used in the same way as regular subcircuits to implement parameterized models, however inline subckts provide some powerful new options. When an inline subcircuit contains both a parameterized model and an inline device which references that model, then the user can create instances of the device, and each device will automatically get an appropriately scaled model assigned to it. For example, the instance parameters to an inline subckt could represent something like emitter width and length of a BJT device and within the subckt a model card could be created which is parameterized for emitter width and length and scales accordingly. When the designer instantiates the macro,

he/she supplies the values for the emitter width and length, and a device is instantiated with an appropriate geometrically scaled model. Again, the inline device does not get a hierarchical name, and can be probed in the same manner as the inline device in the previous section on modeling parasitics, that is, it can be probed just as if it was a simple device, and not actually embedded in a subckt

Automatic Model Selection using Inline Subckts:

See `spectre -h if` for a description on how to combine 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 & M3)

```
M1 (2 1 0 0) mos_mod

M2 (5 6) s1 p1=6u p2=7u

M3 (6 7) s1
```

This expands/flattens to:

```
M1 (2 1 0 0) mos_mod

M2 (5 6 0 0) mos_mod l=6u w=7u  // the "inline" component, inherits call name

M2.cap1 (5 0) capacitor c=1n    // a regular hierarchical name

M2.cap2 (6 0) capacitor c=1n

M3 (6 7 0 0) mos_mod l=1u w=2u  // the "inline" component, inherits call name

M3.cap1 (6 0) capacitor c=1n

M3.cap2 (7 0) capacitor c=1n
```

Here the final flattened names of the three mosfets (one for each instance) are M1, M2 and M3, rather than M1, M2.s1 and M3.s1 as they would be if s1 was a regular subcircuit. The parasitic capacitors (which the user is not really interested in, or perhaps even aware of, if the inline subckt definition was written by a separate modeling engineer) have full hierarchical names however.

Example 3: inline subckt - scaled models

Consider the following example, in which a parameterized model is declared within an inline subcircuit for a bipolar transistor. The model parameters are the emitter width, length, and area, and also the temperature delta (trise) of the device above nominal. Ninety nine instances of a 4*4 transistor are then placed, and one instance of a transistor with area=50 is placed. Each transistor gets an appropriately scaled model.

```
* declare a subckt, which instantiates a transistor with

* a parameterized model. The parameters are emitter width
```

```
* and length.
inline subckt bjtmod (c b e s)
parameters le=1u we=2u area=le*we trise=0
model mod1 bjt type=npn bf=100+(le+we)/2*(area/1e-12)
+       is=1e-12*(le/we)*(area/1e-12)
bjtmod (c b e s) mod1 trise=trise       // "inline" component
ends bjtmod
* some instances of this subckt
q1 (2 3 1 0) bjtmod le=4u we=4u         // trise defaults to zero
q2 (2 3 2 0) bjtmod le=4u we=4u trise=2
q3 (2 3 3 0) bjtmod le=4u we=4u
.
.
q99 (2 3 99 0) bjtmod le=4u we=4u
q100 (2 3 100 0) bjtmod le=1u area=50e-12
```

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 can be double or single in Spice Netlist.

In Spectre netlist,

vec_include "vector_filename" [HLCheck = 0|1]

vector_filename -- the filename of the digital vector file.

HLCheck = 0 | 1 -- 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, users do not need to use the HLCheck flag.

Each command card only specify 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,

.vec "file1.vec"

.vec "file2.vec"

.vec "file3.vec"

===== VCD/EVCD =====

VCD and EVCD formats are widely used in digital circuit design and contain different kinds of information for transistor level simulation. Users need to provide signal information, such as timing characteristics, voltage threshold, and driving ability of input signals for each VCD or EVCD file.

Since VCD and EVCD formats are compatible, the same signal information file can be shared

between VCD and EVCD.

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 only specifies 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,

.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 raw directory,

one is check error report file

%A.vecerr

and the other is check summary report file

%A.veclog

Where %A is analysis name.

To find VEC, VCD and EVCD file, signal_info file format description and examples, see Chapter 13, Verilog Value Change Dump Stimuli, in the *Virtuoso UltraSim Simulator User Guide*.

Verilog-A Usage and Language Summary (veriloga)

Description

Verilog-A is an analog hardware description language standard from Open Verilog International. It allows analog circuit behavior to be described at a high level of abstraction. Behavioral descriptions of modules/components may be instantiated in a Spectre netlist along with regular Spectre primitives.

Verilog-A descriptions are written in file(s) separate from the Spectre netlist file. These descriptions are written in modules (see the module `alpha` below). To include a module in the Spectre netlist, first add the line

ahdl_include "VerilogAfile.va"

to the Spectre netlist file (where VerilogAfile.va is the name of the file in which the required module is defined). The module is instantiated in the Spectre netlist in the same manner as Spectre primitives for example,

name (node1 node2) alpha arg1=4.0 arg2=2

This instantiates an element `alpha`, having two nodes and two parameters.

Verilog-A modules can be debugged using `hdldebug`. `hdldebug` has a GUI and a command line mode. Please refer to the Verilog-A Debugging Tool User Guide for more information.

In this release Verilog-A simulation performance has been improved by compiling the Verilog-A modules. This is explained in more detail in the Verilog-A compilation section below.

==========

Module Template

==========

The following is a Verilog-A module template

```
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
        // 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

==========

In this release the simulation performance of Verilog-A has been improved by performing a one time compilation step. The performance improvement obtained is proportional to the complexity and amount of Verilog-A in your design. Following the initial compilation, recompilation will only be performed if the Verilog-A source is changed.

Verilog-A compilation is enabled by default. If you are making frequent changes to the Verilog-A used in your design, the overhead of the compilation step may become an issue. To turn off compilation set the CDS_AHDLCMI_ENABLE shell environment variable to NO. For example,

```
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
```

or undefine the CDS_AHDLCMI_ENABLE environment variable. For example,

```
unsetenv CDS_AHDLCMI_ENABLE
```

The compiled C code flow stores the compiled shared objects in a database on disk for the simulation to use. The shared objects are stored in a directory termed the 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 the ahdlSimDB by setting the CDS_AHDLCMI_SIMDB_DIR environment variable to the path of a directory. For example,

```
setenv CDS_AHDLCMI_SIMDB_DIR /projects/ahdlcmiSimDirs
```

If the path is writable, the 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, termed ahdlShipDBs. To create such databases, set the CDS_AHDLCMI_SHIPDB_COPY to YES. For example,

```
setenv CDS_AHDLCMI_SHIPDB_COPY YES
```

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 the Verilog-A Reference Manual.

Analog Operators/Waveform Filters

---

ddt(x <,abstol>) Differentiate x wrt time.

idt(x, ic <, assert <, abstol> >)

Integrate x wrt time. Output = ic during dc analysis and when assert is 1.

idtmod(x, <ic <, modulus <, offset> > >)

Circular Integration of x wrt time. Output = ic during DC analysis. Integration is performed with given offset and modulus if specified.

transition(x <, delay <, trise <, tfall>>>)

Specify details of signal transitions. For efficient simulation, it is recommended that x not be a continuous signal, i.e. a function of a probe. See the Verilog-A manual for further explanation of this issue.

slew(x <, SRpos <, SRneg>>) Model slew rate behavior.

delay(x, time_delay, max_delay)

Response(t) = x(t - time_delay).

zi_nd(x, numer, denom, period, < ttransition <,sample offset time >)

z-domain filter function, numerator-denominator form.

zi_zd(x, zeros, denom, period, < ttransition <,sample offset time >)

z-domain filter function, zero-denominator form.

zi_np(x, numer, poles, period, < ttransition <,sample offset time >)

z-domain filter function, numerator-pole form.

zi_zp(x, zeros, poles, period, < ttransition <,sample offset time >)

z-domain filter function, zero-pole form.

laplace_nd(x, numer, denom, <, abstol >)

s-domain filter function, numerator-denominator form.

laplace_zd(x, zeros, denom, <, abstol >)

s-domain filter function, zero-denominator form.

laplace_np(x, numer, poles, <, abstol >)

s-domain filter function, numerator-pole form.

laplace_zp(x, zeros, poles, <, abstol >)

s-domain filter function, zero-pole form.

Built-In Mathematical Functions

---

abs(x) Absolute value

exp(x) Exponential if x < 80

ln(x) Natural logarithm

log(x) Log base 10

sqrt(x) Square root

min(x,y) Minimum

max(x,y) Maximum

pow(x,y) x to the power of y

Noise Functions

white_noise(power <, tag >)

Generates white noise with given power. Noise contributions with the same tag are combined for a module.

flicker_noise(power, exp <, tag >)

Generates pink noise with given power at 1 Hz that varies in proportion to 1/f^exp. Noise contributions with the same tag are combined for a module.

noise_table(vector <, tag >)

Generates noise where power is determined by linear interpolation from the given vector of frequency-power pairs. Noise contributions with the same tag are combined for a module.

AC Analysis Stimuli

ac_stim(<analysis_name <, mag > >)

Small signal source of specified magnitude, active for given analysis.

Analog Events

Analog events must be contained in an analog event detection statement; @(analog_event) statement.

cross(x, direction <, timetol <, abstol >>)

Generates an event when x crosses zero.

above(x, <, timetol <, abstol >>)

Generates an event when x becomes greater than or equal to zero. An above event can be generated and detected during initialization. By contrast, a cross event can be generated and detected only after at least one transient time step is complete.

timer(start_time <, period>)

Set (optionally periodic) breakpoint event at time = start_time.

initial_step< (arg1 <, arg2 <, etc... > >)

Generate an event at the initial step of an analysis. arg1, arg2, etc. 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" ... etc ...

final_step< (arg1 <, arg2 <, etc... > >)

Generate an event at the final step of an analysis. arg1, arg2, etc. 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" ... etc ...

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.

$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. 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" ... etc ...

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.

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 called inst_name.

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.

[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.

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X