Appendix A
Simulator Options

A.1 Introduction

This appendix describes simulator options that affect accuracy or convergence.

A.2 SPICE Options

A.2.1 Global Options

SPICE expects the global options to be given on the `.options` statement.

A.2.1.1 Abstol

The absolute tolerance for currents. Defines the smallest interesting current anywhere in the circuit. Currents smaller than `abstol` are ignored when checking for convergence, and when choosing the time step. `abstol` is a supplement to `reltol` that plays a role when the simulator is checking the accuracies of very small currents. `abstol` prevents the simulator from attempting to converge femtoampere signals to attoampere levels.

As a rule, an absolute tolerance such as `abstol` should be set 10^6–10^8 times smaller than the largest signal of the same type present in the circuit. This ratio could be even greater on sensitive circuits. Typically, the largest current present in analog integrated circuits is in the $1\mu A$–$100\mu A$range, which is why `abstol` defaults to 1 pA. It should be set higher for power electronic circuits. Setting `abstol` too loose results in degraded accuracy. Setting `abstol` too tight prevents the simulator from converging.

A.2.1.2 Gmin

A very small conductance added across nonlinear devices to prevent nodes from floating if a device is turned completely off. By default, $gmin=10^{-12}\mho$. It must be positive, though Spectre allows it to be zero. The manner in which SPICE and Spectre add the `gmin` conductors to the various nonlinear devices is different and is shown in Figures 2.13 and 2.14 on page 41.

If `gmin` is too large it adversely affects accuracy. If it is too small it may adversely affect convergence. Be advised that if `gmin` is large enough to positively affect convergence, it is also large enough to negatively affect accuracy.

A.2.1.3 Limpts

Maximum number of plot points that can be plotted or printed during an AC, DC, or transient analysis. The default value is 201. This nuisance parameter was helpful when hoards of undergraduates were all competing for the same line printer in the basement of Evan's Hall back in the 70's, but it has little value now. Feel free to set it to a value that is as large as you need. In SPICE2, setting this value too high wastes a precious piece of the fixed amount of memory available.

A.2.1.4 Pivrel

The relative threshold used for selecting pivots when factoring the Jacobian matrix. Big pivots are good because they reduce the likelyhood of error building up while factoring. However, insisting on having the largest possible pivots can result in the sparse Jacobian matrix filling-in during the factorization, which would result in the simulation running much more slowly and requiring more memory. `pivrel` specifies how large an entry in the Jacobian has to be in order to be a pivot candidate. It is defined as the minimum acceptable ratio between the absolute value of a pivot and the largest remaining element in the same column.

In rare cases, increasing the value of `pivrel` solves a convergence problem, but more often it just causes the simulator to run more slowly and gives no benefit. Pinning your hopes on `pivrel` is clearly a desperation move, `pivrel` must be specified between 0 and 1, with the default value being 10^{-3}. Reasonable values range between 10^{-12} to 0.5.

A.2.1.5 Pivtol

The minimum absolute value allowed for a Jacobian entry to be considered as a pivot. Default value is 10^{-13}.

Most likely this parameter is settable simply because the original developers were not sure what value to use and wanted to be able to adjust it if necessary. Fortunately, they chose wisely. It should never need adjusting.

A.2.1.6 Reltol

The universal accuracy control. Give value between 0 and 1, values closer to zero imply greater accuracy. `reltol` directly affects the Newton convergence criteria and the time-step control algorithm. It specifies the upper limit on errors relative to the size of the signals present. The default value is 0.1% and typical values range from 10^{-6} to 10^{-2}.

Reducing `reltol` decreases the error in the results computed by the simulator, however no level of accuracy is guaranteed. Nor is any particular level of accuracy implied from a given value for `reltol`. In particular, setting `reltol` to 0.1% in no way implies that the accuracy attained by the simulator is 0.1%.

A.2.1.7 Vntol

The absolute tolerance for voltage. Defines the smallest interesting voltage anywhere in the circuit. Voltages smaller than `vntol` are ignored when checking for convergence, and when choosing the time step. `vntol` is a supplement to `reltol` that plays a role when the simulator is checking the accuracies of very small voltages. `vntol` prevents the simulator from attempting to converge nanovolt signals to picovolt levels.

Absolute tolerances, such as `vntol`, should be set 10^6–10^8 times smaller than the largest signal of the same type present in the circuit. This ratio could be even greater on sensitive circuits. Typically, the largest voltage present in analog integrated circuits is in the order of 10 V, which is why `vntol`, defaults to $1\ \mu V$. It should be set higher for high voltage circuits. Setting `vntol` too loose results in degraded accuracy. Setting `vntol` too tight can prevent the simulator from converging, though this problem is not as severe as it is with `abstol`.

A.2.2 DC Analysis Options

SPICE expects the DC analysis options to be given on the `.options` statement.

A.2.2.1 Itl1

Maximum number of iterations for a DC operating point analysis. The default value is 100. Occasionally you can get SPICE to converge by increasing this value. The likelyhood of convergence stops increasing above 1000, with larger values simply resulting in SPICE taking longer to fail.

A.2.2.2 Itl2

Maximum number of iterations per step in a DC sweep. The default value is 50. Set this to a larger value if SPICE has convergence difficulties in the middle of a DC sweep. Alternatively, you can try taking smaller steps. Again, do not bother with values over 1000, which just delay the 'no convergence' message.

A.2.2.3 Itl6

Number of steps to take in when source stepping. The default value is 0, which disables source stepping. Larger values imply higher likelyhood of convergence, but longer waits.

A.2.3 Transient Analysis Options

SPICE expects the transient analysis options to be given on the `.options` statement (except as noted).

A.2.3.1 Chgtol

`chgtol` is the absolute tolerance for the state variable in the LTE criterion. Thus, `chgtol` is the absolute tolerance for charge through capacitors and flux across inductors. `chgtol` limits how closely LTE is controlled when the time step is small and when the charge through the capacitor or flux across the inductor is small. Unlike `abstol`, `chgtol` is multiplied by `reltol` in the LTE criterion. Thus, capacitor charges must be less than `reltol` × `chgtol` before they are dominated by `chgtol`.

A.2.3.2 Itl3

Sets the lower transient analysis iteration limit. The default value is 4. It is difficult to conceive of a situation that calls for adjusting the value of this parameter.

A.2.3.3 Itl4

Sets the upper per step iteration limit for transient analysis. The default value is 10. Increasing this value aids convergence on some circuits, particularly on circuits that exhibit strongly discontinuous behavior, like some macromodels. On such circuits, increasing the value could speed up the analysis considerably. On other circuits, it could cause the analysis to run slower.

The recommended upper limit is 50 unless SPICE is failing to converge in transient, in which case 500 might be better.

A.2.3.4 Itl5

Upper limit on the total number of iterations during a transient analysis. The default value is 5000. Setting the value to 0 disables the limit. This is another parameter than made sense when there was large number of users, all simulating opamps, and all competing for a single mainframe. It is a good habit to always set this limit to 0. Who wants to wait for a long transient simulation, only to have it unexpectedly terminate prematurely due to this limit.

A.2.3.5 Lvltim

This flag specifies the time-step control algorithm. If 1, SPICE uses the iteration-count time-step control, and if 2 it uses truncation-error time-step control. Briefly put, controlling truncation error is very important. If the truncation error is not properly controlled, the time-constants computed by the simulator could be meaningless. Iteration-count time-step control ignores completely the truncation error and so is capable of computing incorrect results. Always use `lvltim=2` unless you do not mind if the simulator produces results in which the time-constants are way off. See Section 4.3.1.2 on page 179 for more information.

A.2.3.6 Maxord

The maximum integration method order for the backward difference method (Gear's method). Default and minimum value is 2. Maximum value is 6. Higher order methods should be more efficient when very high accuracy is needed and the signals are very smooth. However, they can also be unstable on lightly damped circuits. I recommend leaving this parameter at its default value of 2 (the higher-order methods have never been heavily used and there are rumored to be some problems in SPICE's implementation).

A.2.3.7 Method

Integration method, choose between `gear` and `trapezoidal`. The default value is `trapezoidal`. Select `gear` if trapezoidal ringing is a problem. Select `trapezoidal` is simulating highly underdamped circuits. Otherwise, you are generally free to choose either. Trapezoidal rule is usually a bit more efficient. See Section 4.2.1 on page 131 for more information.

A.2.3.8 Tmax

T_{max} is the optional fourth argument on the transient analysis statement. It places an upper bound on the time step. If T_{max} is not specified, SPICE sets it to

$$T_{\mathrm{max}} = \frac{T_{\mathrm{stop}} - T_{\mathrm{start}}}{50}. \tag{A.1}$$

In addition, SPICE reduces T_{max} to T_{step} (the second argument on the transient analysis statement) if there are no energy-storage elements such as capacitor or inductor present in the circuit. Finally, SPICE bounds T_{max} to be no larger than half the transmission delay of the shortest transmission line. Very short transmission lines significantly slow a SPICE transient analyses.

A.2.3.9 Tstep

T_{step} is the printing and plotting increment. It is also the suggested time step. T_{step} also has an effect on the number of points used when performing Fourier analysis. See Section 5.3.2 on page 278 for more information.

A.2.3.10 Trtol

`trtol` is only used in the LTE criterion where it multiplies `reltol`. Its primary function is to allow control of the LTE criterion independent of the Newton convergence criteria. It is not a good idea to tighten the LTE criterion without also tightening the Newton convergence criteria because it can cause the simulator to take very small time steps trying to follow the numerical noise generated by errors allowed by the Newton criteria. Thus, one should never reduce `trtol` much below its default value of 7. There is benefit to loosening the LTE criterion relative to the Newton criteria. For example, if you were less concerned about accurately depicting time constants than equilibrium points, you might want to loosen the LTE criterion relative to the Newton criteria by increasing `trtol`. Similarly, if charge conservation is important, you might want to tighten the Newton criteria without affecting the LTE criterion. This allows you to improve charge conservation without slowing the simulation down by much. In this case, tighten `reltol` and loosen `trtol` by the same amount.

A.2.3.11 Uic

When `uic` is specified on the transient analysis statement, it directs SPICE to skip any attempt to perform an initial DC or IC solve for a transient analysis. With `uic`, all nodes for which initial conditions are not specified start at 0 V. One important use for this option is to allow people to use transient analysis even though the initial condition calculations do not converge. One can even continue transient analysis until steady-state is reached and save the final results for use as a node-set. For more information, see Section 4.3.5.1 on page 196.

A.3 Spectre Options

A.3.1 Global Options

Spectre expects global analysis options to be given on the `options` or `set` statements. Type ‘`spectre -help options`’ for a full list of Spectre’s options.

A.3.1.1 Approx

When set, this flag allows Spectre to make small approximations in the component model equations that allow the simulation to run faster. The errors made when using this option should be very small. If you are a nervous type, disable this to reassure yourself that the approximations are reasonable, then turn it back on.

A.3.1.2 Diagnose

This is a flag that when set causes additional diagnostics to be printed. It is not enabled by default because some the diagnostics may be costly to compile or may be a nuisance in normal circumstances. When enabled, Spectre warns about unusual growth in maximum signal levels, which might indicate the presence of unstable time constants. It also produces counts of the number of times each node prevents convergence. This information helps localize convergence problems to a particular portion of the circuit.

A.3.1.3 Gmin

A very small conductance added across nonlinear devices to prevent nodes from floating if a device is turned completely off. By default, `gmin`$=10^{-12}\mho$. It must be positive, though Spectre allows it to be zero. The manner in which SPICE and Spectre add the `gmin` conductors to the various nonlinear devices is different and is shown in Figures 2.13 and 2.14 on page 41.

If `gmin` is too large it adversely affects accuracy. If it is too small it may adversely affect convergence. Be advised that if `gmin` is large enough to positively affect convergence, it is also large enough to negatively affect accuracy.

A.3.1.4 Homotopy

Specifies the homotopy (continuation method) used when computing DC operating points and initial conditions. The available choices are `none`, `gmin` (`gmin` stepping), `source` (source stepping), `ptran` (pseudo-transient), `dptran` (damped pseudo-transient), and `all`, the default value. Specifying a different value allows you to skip methods that always fail for a particular circuit. Only change the value from `all` when experience with a particular circuit shows that there is at least one reliable method.

A.3.1.5 Iabstol

The absolute tolerance for currents. Spectre's `iabstol` is the same as SPICE'S `abstol`. See the description for `abstol` in Section A.2.1.1 on page 335 for more information.

A.3.1.6 Limit

Specifies the limiting algorithm to be used to help Newton's method converge. Possible values are `dev`, which specifies that limiting is performed at the device, `delta`, specifies a node based limiting, and `log`, which specifies a node-based logarithmic limiting. The default is `dev.` `delta` has been found to work well on MOS circuits, however both node-based approaches are very slow on circuits that exhibit high voltages. Occasionally, one can get Spectre to converge much faster by playing with the value of this parameter, but in generally it is rarely worth the time.

A.3.1.7 Macromodels

Over the years Spectre has been tuned to run quickly on real circuits. These optimizations have the side effect of causing Spectre to run more slowly on circuits that deviate greatly from the norms of physical circuits. In particular, Spectre performs poorly with some macromodels. Setting this flag makes Spectre more tolerant of the aberrant behavior exhibited by these macromodels. With `macromodels` set, Spectre becomes more determined to efficiently pass through the abrupt discontinuities exhibited by some macromodels and uses local error criteria to allow huge voltages and currents at a subset of the nodes.

By default `macromodels=no` because otherwise Spectre runs more slowly on most circuits, but enabling it may cause Spectre to run much quicker on circuits that exhibit non-physical behavior.

A.3.1.8 Opptcheck

This flag specifies whether Spectre should check the operating point parameters against their soft limits after the each analysis. Soft limits are ranges that can be specified for any component parameter that when violated causes Spectre to issue a warning. A file containing a set of soft limits tailored for IC design is provided with Spectre, but you are free to use your own files or customize the ones provided. By default, `opptcheck=yes`. However, it is not useful unless you provide Spectre with a set of soft limits (use `+param` argument on the Spectre command line or in the `SPECTRE_DEFAULTS` environment variable).

It is highly recommended that you use this feature because it can alert you to subtle problems in your circuit. With a little effort developing your own soft limit files, Spectre will automatically warn you of the following situations and more:

1. Transistors that are the wrong type (NPNs versus PNPs, N-type versus P-type).
2. Excessive substrate currents caused by incorrectly wiring the circuit or unexpected operating conditions.
3. Excessive device currents, voltages or powers.
4. Unexpected operating conditions that result in unacceptably poor g_m or f_T.
5. Incorrectly specified parameter values, such a width or length mistakenly given in microns rather than meters.

A.3.1.9 Pivabs

The minimum absolute value allowed for a Jacobian entry to be considered as a pivot. Default value is 0.

A.3.1.10 Pivotdc

This flag turns on numerical pivoting throughout the DC analysis. Normally numerical pivoting is used only once at the beginning of DC analysis and the matrix is not reordered unless it is clearly needed. Occasionally convergence is improved by allowing Spectre to reorder the matrix every time it is factored during a DC analysis. It is rare for this option to help convergence, but when it does it is usually when simulating a circuit with extraordinary gain, such as a string of inverters or amplifiers. However, enabling this option can result in the DC analysis running much more slowly, especially on large circuits. By default, this option is disabled.

A.3.1.11 Pivrel

The relative threshold used for selecting pivots when factoring the Jacobian matrix. See the description of `pivrel` in Section A.2.1.4 on page 336 for more information.

A.3.1.12 Reltol

The universal accuracy control. See the description of `reltol` in Section A.2.1.6 on page 337 for more information.

A.3.1.13 Rforce

`rforce` is the value used by Spectre when imposing nodesets, forces, and initial conditions. Spectre imposes nodesets, forces, and initial conditions on nodes by attaching the series combination of a voltage source and a resistor whose value is `rforce` times a local multiplier. The multiplier is unity for free nodes, but may be different for branches and nodes internal to components. For example, you are allowed to also specify a local `rforce` multiplier on inductors. The actual value used is the product of the local and global values.

`rforce` is used adjust the forcing resistor's value to overcome small resistors in the circuit. The default value is 1, but you can make it smaller if small resistors in your circuit are causing your nodesets, forces, and initial conditions to be inaccurate.

A.3.1.14 Vabstol

The absolute tolerance for voltage. Spectre's `vabstol` is the same as SPICE'S `vntol`. See the description of `vntol` in Section A.2.1.7 on page 338 for further information.

A.3.2 DC Analysis Options

Spectre expects the DC analysis options to be given on the DC analysis statement (except as noted). Type ' `spectre -help dc`' for a full list of Spectre's DC analysis parameters.

A.3.2.1 Check

Specifies that Spectre should check the operating point parameters against the soft limits for the final value of the DC analysis. See the description for `opptcheck` in Section A.3.1.8 on page 345 for more information about soft limits.

A.3.2.2 Force

A 'force' is a new concept in Spectre. It provides DC analysis something analogous to transient analysis's 'initial condition'. It is used to force node voltages or branch currents to some prespecified value. Unlike with nodesets, the resulting solution is not an equilibrium point. It is occasionally useful to force node voltages or branch currents in order to give the circuit an isolated solution.

Use the `force` option to specify the source for the forced values. If `force=none`, which is the default, then no values are forced at the nodes and branches. When `force=node`, then values specified on initial condition statements are forced. When `force=dev`, the values specified as initial condition parameters on various components are used. Finally, if `force=all`, then the initial conditions provided on both initial condition statements and as initial condition parameters are used. If you specify a ‘force’ file with the `readforce` parameter, force values read from the file are used in lieu of any ‘ic’ statements.

A.3.2.3 Homotopy

Specifies the homotopy (continuation method) used when computing DC operating points and initial conditions. See the description of `homotopy` parameter in Section A.3.1.4 on page 343 for further information. Specifying `homotopy` on the DC analysis overrides `homotopy` specified as a global option.

A.3.2.4 Maxiters

Maximum number of iterations for a DC operating point analysis. It also indirectly specifies the maximum number of iterations taken on each step of a homotopy. Default value is 150, and rarely needs to be changed.

A.3.2.5 Maxsteps

Maximum number of steps used in homotopy method. The default value is 10,000. There is little to be gained by increasing this value because if you get to this value there is very little hope of ever converging. However, you might want to set this value lower so that Spectre gives up quicker.

A.3.2.6 Readforce

Allows you to specify the name of a ‘state’ file and use its contents as node forces. The state file is an ASCII file that contains name/value pairs, one pair per line. The name is the either the name of a node or a branch. The state file can be manually generated, it can be generated by Spectre during a previous analysis, or it could be generated by Spectre on a previous run.

A.3.2.7 Readns

Allows you to specify the name of a 'state' file and use its contents as nodesets. The state file is an ASCII file that contains name/value pairs, one pair per line. The name is the either the name of a node or a branch. The state file can be manually generated, it can be generated by Spectre during a previous analysis, or it could be generated by Spectre on a previous run.

Names corresponding to signals that no longer exist are ignored except for a warning. A typical usage is to specify the same file for both `readns` and `write`. The first time the analysis is run the file is not found, which generates a warning that can be ignored. On subsequent runs, the nodesets are automatically updated on every run.

A.3.2.8 Restart

This flag causes Spectre to discard the operating point computed as the last point of the previous analysis. This is done because the last point of a temperature sweep, DC sweep, or transient analysis may be far from the operating point needed for the next analysis. With `restart=yes`, which is its default setting, Spectre discards the operating point and starts fresh if anything has changed that causes the previous operating point to be out-of-date.

It is useful to set `restart=no` if you are running a sequence of analyses and you script the analyses to assure the operating point changes little or not at all between analyses. For example, if you want to perform an AC analysis at 0 C and 50 C, and you also need a DC temperature sweep from 0 C to 50 C, then arrange to perform the AC analysis at 0 C first. It computes the operating point at 0 C. Then perform the temperature sweep with `restart=no`. It finishes by computing the operating point at 50 C. Finally, perform the AC analysis at 50 C with `restart=no`. Using `restart=no` for the second and third analyses reduces the time needed by the analyses by giving them good initial guesses for the operating point.

Setting `restart=no` does not cause Spectre to skip computing an operating point. It only affects what is used as the starting point for the operating point computation. The only time Spectre skips computing the operating point is if it is explicitly told to (see the `prevoppoint` parameter for the AC analyses, or the `skipdc` parameter for transient analysis) or if it is sure that the operating point is up-do-date. In either case, `restart` does not play a role in deciding whether the operating point should be computed.

A.3.2.9 Write

Specifies the name of the file to which Spectre should write the 'state' of the circuit after the initial point of a DC analysis. This file can be later used as a nodeset file, a node force file, or an initial condition file. The file contains one name/value pair for every signal in the circuit.

A.3.2.10 Writefinal

Specifies the name of the file to which Spectre should write the 'state' of the circuit after the final point of a DC analysis. This file can be later used as a nodeset file, a node force file, or an initial condition file. The file contains one name/value pair for every signal in the circuit.

A.3.3 Transient Analysis Options

Spectre expects the transient analysis options to be given on the transient analysis statement. Type '`spectre -help tran`' for a full list of Spectre's transient analysis parameters.

A.3.3.1 Cmin

Specifies the amount of capacitance that should be connected from each node to ground. Normally this should be zero. However, if you are having trouble with convergence in the intrinsic transient analysis (not the initial condition analysis that precedes the actual transient analysis) then setting cmin to a small value (say 1 fF) usually allows the transient analysis to gracefully work its way beyond discontinuities that would otherwise cause convergence failure.

A.3.3.2 Errpreset

Selects a reasonable collection of parameter settings. Possible values are conservative, moderate or liberal. Specifying conservative makes Spectre more cautious and biases it towards accuracy. Conversely, liberal makes Spectre more reckless (without taking undue risks) and biases it towards speed. See Section 4.3.2.2 on page 187 for more information.

A.3.3.3 Ic

Use the ic parameter to specify the source for the initial conditions. If ic=none, then no initial conditions are used. When ic=node, then values specified on initial condition statements are used. When ic=dev, the values specified as initial condition parameters on various components are used. Finally, if ic=all, which is the default, then the initial conditions provided on both initial condition statements and as initial condition parameters are used. If you specify an 'ic' file with the readic parameter, the initial conditions are read from the file are used in lieu of any 'ic' statements.

A.3.3.4 Lteratio

Ratio used to compute truncation error tolerances from Newton tolerance. The default value is derived from errpreset. This parameter is similar to SPICE's trtol, except it is twice as small (the default value of Iteratio=3.5 corresponds to the default value of trtol=7). For more information, see the description of trtol in Section A.2.3.10 on page 342.

A.3.3.5 Maxiters

Sets the upper per step iteration limit for transient analysis. The default value is derived from `macromodels`. Increasing this value aids convergence on some circuits, particularly on circuits that exhibit strongly discontinuous behavior, like some macromodels. On such circuits, increasing the value could speed up the analysis considerably. On other circuits, it could cause the analysis to run slower. The recommended upper limit is 50.

A.3.3.6 Maxstep

Specifies the maximum time step used during a transient analysis. On the whole, people tend to over use `maxstep`. Normally, one should tighten `reltol` if greater accuracy is needed. However, there are certain situations where tightening `maxstep` is the best way to get the best accuracy. Examples include simulating the onset of oscillation, where the oscillation is small compared to the DC level and so tightening `reltol` is largely ineffective, and when performing Fourier analysis where a well controlled and nearly evenly spaced time step is a virtue. The default value is derived from `errpreset`.

Unlike SPICE, the maximum time step in Spectre is not affected by the electrical length of the transmission lines.

A.3.3.7 Method

Specifies the integration method or combination of integration methods used during the transient analysis. The possible choices include

`euler`
: Backward-Euler is used exclusively

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

`traponly`
: Trapezoidal rule is used almost exclusively (backward-Euler is used at break-points).

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

`gear2only`
: Gear's second-order backward-difference method is used almost exclusively (backward-Euler is used at break-points).

`trapgear2`
: Allows all three integration methods to be used. Trapezoidal rule and second-order Gear are used on alternating time steps, while backward-Euler is used at break-points.

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

See Section 4.2.2 on page 133 for a more detailed description of the advantages and disadvantages of each method. The default is derived from `errpreset`.

A.3.3.8 Readic

Allows you to specify the name of a 'state' file and use its contents as initial conditions. The state file is an ASCII file that contains name/value pairs, one pair per line. The name is the either the name of a node or a branch. The state file can be manually generated, it can be generated by Spectre during a previous analysis, or it could be generated by Spectre on a previous run.

A.3.3.9 Readns

Allows you to specify the name of a 'state' file and use its contents as nodesets. The state file is an ASCII file that contains name/value pairs, one pair per line. The name is the either the name of a node or a branch. The state file can be manually generated, it can be generated by Spectre during a previous analysis, or it could be generated by Spectre on a previous run.

Names corresponding to signals that no longer exist are ignored except for a warning. A typical usage is to specify the same file for both `readns` and `write`. The first time the analysis is run the file is not found, which generates a warning that can be ignored. On subsequent runs, the nodesets are automatically be updated on every run.

A.3.3.10 Relref

This parameter determines how the relative error used in convergence criteria and the truncation error criterion is computed. The possible choices include

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

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

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

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

Default derived is from `errpreset`. See Section 4.3.2.1 on page 183 for more information.

A.3.3.11 Restart

This flag causes Spectre to discard the operating point computed as the last point of the previous analysis. For more information, see the description for `restart` in Section A.3.2.8 on page 349.

A.3.3.12 Skipdc

Flag that indicates the transient analysis should not employ the DC analysis algorithms in an attempt to compute any values in the initial state that were not specified as initial conditions (this is similar to the `uic` parameter of SPICE). This parameter is generally used when the simulator refuses to converge otherwise. See Section 4.4.1 on page 207 for more information about the various choices available with `skipdc`.

A.3.3.13 Step

Specifies the minimum step size Spectre takes to maintain the aesthetics of the results. Spectre monitors the voltage across capacitors and the current through inductors, and chooses the time step to maintain the accuracy of the solution. It does not need to do this for the node voltages and branch currents not associated with capacitors or inductors, because these quantities are accurate regardless of the time step. However, when plotting these waveforms it is important to have more points where the curvature is high in order to convey the true shape of the waveform. For this reason, Spectre chooses the time step to faithfully depict the shape of waveforms that do not directly affect the accuracy of the solution. However, occasionally these waveform have discontinuous jumps that would require infinitely small time steps to resolve. `step` specifies the minimum step size that will be taken to resolve the details of these ‘algebraic’ signals. The default value is derived from `errpreset`.

A.3.3.14 Write

Specifies the name of the file to which Spectre should write the ‘state’ of the circuit after the initial point of a transient analysis. This file can be later used as a nodeset file, a node force file, or an initial condition file. The file contains one name/value pair for every signal in the circuit.

A.3.3.15 Writefinal

Specifies the name of the file to which Spectre should write the ‘state’ of the circuit after the final point of a transient analysis. This file can be later used as a nodeset file, a node force file, or an initial condition file. The file contains one name/value pair for every signal in the circuit.

Appendix B
Spectre Netlist Language

B.1 Introduction

In addition to accepting SPICE netlists, Spectre also supports a simple, powerful, and extensible language for describing netlists. This appendix describes the basics of Spectre's netlist language only to the level of detail needed to allow you to understand the netlists given in this book.

B.2 The Language

When reading a netlist, SPICE determines the type of a component by the first letter in its name. For example, R1 must be a resistor and Vin must be a voltage source. This approach follows engineering convention and fits well with SPICE'S original goal of being a simulator for integrated circuits. However, this convention has two unfortunate limitations. First, it limits the simulator to only supporting only 25 component types (X is reserved for subcircuits). Second, it does not allow the representation type of the components to change. For example, Q1 must refer to a built-in model for a bipolar transistor. It might also be convenient use a macromodel for the transistor. Indeed, integrated transistors often have other junctions near-by that can form parasitic transistors. The built-in model does not include parasitic transistors, but it is easy to construct a macromodel using a subcircuit that contains the original transistor along with the parasitic transistor. However, because of the constraints on the first letter, one cannot change the type of a transistor from built-in model to macromodel without changing the transistors name. This might not seem like a serious problem until you consider a circuit with thousands of transistors. If you would like to change just the NPN transistors from the built-in model to the subcircuit macromodel that includes the parasitic transistor, you have to make perhaps thousands of hand edits on the netlist.

Spectre's native netlist language is modeled after the SPICE language, but is more uniform. It provides several important features that are unavailable in SPICE due to limitations of its language, including the ability to support an arbitrary number of primitives and the ability to switch representation type.

There are three basic types of primary statements, along with some secondary control statements. The primary statements include component instance, component model, and analysis statements. The control statements include those for specifying initial conditions, nodesets, outputs, etc, and are not discussed further here. In addition, Spectre provides statments used to define parameterized subcircuits.

Basic Language Attributes The Spectre parser has two modes. By default, Spectre is configured to expect SPICE netlists unless told otherwise. When it reads

```
    simulator lang=spectre
```

it switches off SPICE compatibility. When doing so, it makes the following changes:

1. Spectre no longer accepts SPICE statements and constructs.
2. The language becomes case-sensitive, with all keywords being lower case.
3. Standard SI scale-factors are used as a convenient way of specifying either very large or very small numbers. The SI scale

$P = 10^{15}$	$T = 10^{12}$	$G = 10^9$	$M = 10^6$	$K = 10^3$
$k = 10^3$	$\_ = 1$	$\% = 10^{-2}$	$c = 10^{-2}$	$m = 10^{-3}$
$u = 10^{-6}$	$n = 10^{-9}$	$p = 10^{-12}$	$f = 10^{-15}$	$a = 10^{-18}$

Table B.1: SI scale factors.

$t = 10^{12}$	$g = 10^9$	$meg = 10^6$	$k = 10^3$	$m = 10^{-3}$
$mil = 25.4 \times 10^{-6}$	$u = 10^{-6}$	$n = 10^{-9}$	$p = 10^{-12}$	$f = 10^{-15}$

Table B.2: SPICE scale factors.

factors are different than those used by SPICE. For example, with SI, $m = 10^{-3}$ and $M = 10^6$, whereas with SPICE both m and $M = 10^{-3}$, while $MEG = 10^6$. The SI scale factors are detailed in Table B.1 while the SPICE scale-factors are shown in Table B.2.

To switch back to accepting SPICE netlists, use

```
simulator lang=spice
```

When Spectre includes one file from another using the `include` statement, it automatically switches to SPICE mode at the beginning of the new file, and returns to the previously active mode when it finishes reading it. For this reason, all Spectre netlists must begin with a `lang=spectre` statement.

Model Statements Often, certain characteristics are common to a large number of instances of components of the same type. For example, the saturation current of a diode is a function of the process used to construct the diode and also of the area of the diode. Rather than describing the process on each diode instantiation, that description is done once in a model statement and many diode instances refer to it. The area, which can be different for each component, is included on each instance statement. Though it is possible to have several model statements for a particular type of component, each instance can only reference at most one model. Not all types of components support model statements.

Model statements have the form

```
model dnp diode is=3.1e-10 n=1.12 cjo=3.1e-8 vj=.914
```

In this example, *dnp* is the model name, and *diode* is the primitive name. The primitive name is either the name of a built-in primitive or a user designed behavioral model.

Instance Statements The instance statement consists of an instance name, the nodes to which the terminals of the instance are connected, the name of the master, and the parameters. It takes the form,

```
M1 (4 pin 1 1) nmos w=402.4u l=7.6u
```

where *M1* is the instance name, '*(4 pin 1 1)*' is the node list (parentheses are optional), *nmos* is the master name, in this case a model name, and '*w=402.4u l=7.6u*' are the parameters. The master field contains either the name of a built-in component type (for example, capacitor), the name of a user-defined behavioral model, a model name, or a subcircuit definition name. The nodes must appear in the order defined by the master. Unlike SPICE, the first character in the instance names is not fixed to any particular value by the type field.

The list of Spectre's available built-in components is shown in Table B.3 on the facing page and Table B.4 on page 362. Further information on these components and their parameters is found by using Spectre's `-help` command-line option.

bjt	Bipolar junction transistor
bsim1	BSIM1 MOSFET
bsim2	BSIM2 MOSFET
bsim3	BSIM3 MOSFET
diode	Diode model
gaas	GaAs MESFET
jfet	Junction FET
mos0	MOSFET level-0
mos1	MOSFET level-1
mos2	MOSFET level-2
mos3	MOSFET level-3
mos8	GE-Intersil MOSFET
phy_res	Physical resistor

Table B.3: Spectre's semiconductors. In addition, Spectre supports a number of proprietary foundry models.

Subcircuit Definitions Subcircuit definitions are circuit macros that can be expanded anywhere in the circuit any number of times. They take the form

```
subckt diffamp (pin nin pout nout)
  parameters ad=1 ac=0 rd=0 rc=0
  Acmp  (i1  0   pin  0)      vcvs gain=ac/2
  Acmn  (cm  i2  nin  0)      vcvs gain=ac/2
  Admp  (poutx  cm pin nin)  vcvs  gain=ad/2
  Admn  (noutx  cm pin nin)  vcvs  gain=-ad/2
  Rdp   (pout  poutx)        resistor r=rd/2
  Rdn   (nout  noutx)        resistor r=rd/2
  Rc    (i1   i2)            resistor r=rc-rd/4
ends diffamp
```

subckt, parameters, and ends are keywords. This subcircuit has four terminals *(pin, nin, pout,* and *nout),* and four parameters *(ad, ac, rd,* and *rc).* The parameters are used in constant expressions that specify the parameter values for the components in the subcircuit.

`capacitor`	Two terminal capacitor
`cccs`	Linear current-controlled current source
`ccvs`	Linear current-controlled voltage source
`core`	Magnetic core with hysteresis
`delay`	Ideal delay line
`fourier`	Fourier analyzer
`inductor`	Two Terminal inductor
`iprobe`	Current probe
`isource`	Independent current source
`msline`	Microstrip transmission line
`mutual_inductor`	Mutual inductor coupling
`nport`	N-port (specified using file of S-parameters)
`pcccs`	Polynomial current-controlled current source
`pccvs`	Polynomial current-controlled voltage source
`port`	Independent resistive source
`pvccs`	Polynomial voltage-controlled current source
`pvcvs`	Polynomial current-controlled voltage source
`relay`	Four terminal relay
`resistor`	Two terminal resistor
`scccs`	s-domain current-controlled current source
`sccvs`	s-domain current-controlled voltage source
`svccs`	s-domain voltage-controlled current source
`svcvs`	s-domain voltage-controlled voltage source
`switch`	Ideal switch
`tline`	Transmission line (with skin effect)
`transformer`	Linear two-winding ideal transformer
`vccs`	Linear voltage-controlled current source
`vcvs`	Linear voltage-controlled voltage source
`vsource`	Independent voltage source
`winding`	Winding for magnetic core
`zcccs`	z-domain current-controlled current source
`zccvs`	z-domain voltage-controlled voltage source
`zvccs`	z-domain current-controlled current source
`zvcvs`	z-domain voltage-controlled voltage source

Table B.4: Spectre's components.

When an instance in your input file refers to a subcircuit definition, the instances specified within the subcircuit are inserted into the circuit. Local model definitions are allowed within a subcircuit definition. Also instances of other subcircuits as well as local subcircuit definitions are allowed within a subcircuit definition. Names of subcircuits and models defined within a subcircuit are strictly local to that subcircuit.

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 specified in the subcircuit definition and must be in the same order. Node names inside the subcircuit definition are strictly local unless declared otherwise in the input file with a global statement. Model names for models defined inside a subcircuit definition are strictly local and are not accessible for specifying component instances outside the current subcircuit definition.

Parameter specification in subcircuit definitions is optional. Any parameters that are specified are referred to by name followed by an equals sign and then a default value. If, when making a subcircuit call in your input file, you do not specify a particular parameter, then this default value is used in the macro expansion. Subcircuit parameters are used in expressions within the subcircuit as demonstrated in the example above `(r=rc-rd/4)`. The expressions employ subcircuit parameters and constants, and are constructed using the four standard arithmetic operators `(+, -, *, /)` and parentheses for grouping.

Analysis Statements Analysis statements have the same form as component instance statements, the main difference is that they specify analyses rather than components, and the order in which they are given determines the order in which the analyses are run. Analysis statements take the form,

```
dmStepResponse tran stop=2us errpreset=conservative
```

`ac`	AC small-signal analysis
`alter`	Alter a circuit or component parameter
`check`	Check component parameter values
`dc`	DC operating-point analysis
`info`	Circuit and component information
`noise`	Noise analysis
`options`	Set circuit options (immediate)
`set`	Set circuit options (deferred)
`shell`	Run a UNIX shell command
`sp`	S-parameter analysis
`tdr`	Time-domain reflectometer analysis
`tran`	Transient analysis
`xf`	Small-signal transfer-function analysis

Table B.5: Spectre's analyses.

It is also possible to provide a list of nodes with an analysis, as in

```
dmNoise (dout 0) noise start=1 stop=1GHz iprobe=Vid
```

In contrast to SPICE, Spectre provides a rich set of parameters that are used to specify the behavior of each analysis. Any number of any analysis commands can be given in any order. Information on individual analyses are available by using the `-help` command line option to Spectre.

Currently, Spectre supports the analyses shown in Table B.5.

SPICE Extensions In addition to providing support for SPICE netlists, Spectre also provides access to many of the benefits of the Spectre netlist language from SPICE netlists. For example, parameterized subcircuits and multiple analyses are available. Furthermore, any Spectre specific parameters can be appended to the SPICE form of a component instance or analysis statement. You are also free to use either SPICE or Spectre forms for instance and analysis statemerits in SPICE netlists. By switching to Spectre forms, you get representation switching, named parameters, constant expressions for parameter values, and support for new component types without constraining the first letter of the component instance name.

Bibliography

[antognetti93] Paolo Antognetti and Giuseppe Massobrio (editors). *Semiconductor Device Modeling with* SPICE. McGraw-Hill, 1993.

[boser88] Bernhard E. Boser and Bruce A. Wooley. "The design of sigma-delta modulation analog-to-digital converters." *IEEE Journal of Solid-State Circuits,* vol. 26, no. 6, December 1988.

[chua87] Leon O. Chua, Charles A. Desoer and Ernest S. Kuh. *Linear and Nonlinear Circuits.* McGraw-Hill, 1987.

[connelly92] J. Alvin Connelly and Pyung Choi *Macromodeling with SPICE.* Prentice-Hall, 1992.

[duff86] I. S. Duff, A. M. Erisman and J. K. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, 1986.

[gilbert82] Barrie Gilbert. "A monolithic microsystem for analog synthesis of trigonometric functions and their inverses." *IEEE Journal of Solid-State Circuits,* vol. SC-17, No. 6, December 1982.

[gray84] Paul R. Gray and Robert G. Meyer. *Analysis and Design of Analog Integrated Circuits.* John Wiley and Sons, 1984.

[harris78] Fredric J. Harris. "On the use of windows for harmonic analysis with the discrete Fourier transform." *Proceedings of the IEEE,* vol. 66, no. 1, January 1978.

[jeng] Min-Chie Jeng. To be published by Kluwer Academic Publishers, Boston.

[kinget94] Peter Kinget, Jan Crols, Mark Ingels and Enzo Peluso. *Circuits and Devices Magazine.* "Are circuit simulators becoming too stable?", vol. 10, no. 3, pp. 50, May 1994.

[kundert90] Kenneth S. Kundert, Jacob K. White and Alberto Sangiovanni-Vincentelli. *Steady-State Methods for Simulating Analog And Microwave Circuits.* Kluwer Academic Publishers, Boston 1990.

[maas88] Stephen A. Maas. *Nonlinear Microwave Circuits.* Artech, 1988.

[mccalla87] W. J. McCalla. *Fundamentals of Computer-Aided Circuit Simulation.* Kluwer Academic Publishers, Boston 1987.

[meyer71] J. E. Meyer. "MOS models and circuit simulation." *RCA Review,* vol. 32, 1971.

[meyer89] Robert G. Meyer and William D. Mack. "A wide-band class AB monolithic power amplifier." *IEEE Journal of Solid-State Circuits,* vol. 24, no. 1, February 1989.

[middlebrook75] R. D. Middlebrook. "Measurement of loop gain in feedback systems." *International Journal of Electronics,* vol. 38, no. 4, April 1975.

[motchenbacher73] C. D. Motchenbacher and F. C. Fitchen. *Low Noise Electronic Design.* John Wiley and Sons, 1973.

[pederson84] Donald O. Pederson. "A historical review of circuit simulation." *IEEE Transactions on Circuits and Systems,* vol. CAS-31, no. 1, January 1984.

[rohrer71] Ronald Rohrer, Laurence Nagel, Robert Meyer and Lynn Weber. “Computationally efficient electronic-circuit noise calculations.” *IEEE Journal of Solid-State Circuits,* vol. SC-6, no. 4, August 1971.

[sangiovanni81] Alberto L. Sangiovanni-Vincentelli. “Circuit Simulation.” In *Computer Design Aids for VLSI Circuits,* P. Antognetti, D. O. Pederson and H. De Man (editors). Martinus Nijhoff Publishers, 1986, pp. 19–112.

[spice2] *SPICE Version 2G,* Available through the Software Distribution Office, Cory Hall, University of California, Berkeley, CA 94720.

[spice3] *SPICE Version 3F,* Available through the Software Distribution Office, Cory Hall, University of California, Berkeley, CA 94720.

[vlach83] Jiri Vlach and Kishore Singhal. *Computer Methods for Circuit Analysis and Design.* Van Nostrand Reinhold, 1983.

[vladimirescu94] Andrdi Vladimirescu. *The SPICE Book.* Wiley, 1994.

[ward78] Donald. E. Ward and Robert. W. Dutton. ‘A charge-oriented model for MOS transistor capacitances.” *IEEE Journal of Solid-State Circuits,* vol. SC-13, no. 5, pp. 703–708, October 1978.

[white86] J. K. White and A. Sangiovanni-Vincentelli. *Relaxation Techniques for the Simulation of VLSI Circuits.* Kluwer Academic Publishers, Boston 1986.

[yang82] Ping Yang and Pallab K. Chatterjee. “SPICE modeling for small geometry MOSFET circuits.” *IEEE Transactions on the Computer-Aided Design of Integrated Circuits and Systems,* vol. CAD-1, no. 4, pp. 169–182, October 1982.

[zverev67] Anatol I. Zverev. *Handbook of Filter Synthesis.* Wiley, 1967.

Index