SILVACO

SmartSpice User's Manual

Silvaco, Inc.
4701 Patrick Henry Drive, Bldg. 2
Santa Clara, CA 95054
Phone: (408) 567-1000
Web: www.silvaco.com

August 28, 2013

Notice

The information contained in this document is subject to change without notice.

Silvaco, Inc. MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE.

Silvaco, Inc. shall not be held liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

AccuCell, AccuCore, Athena, Athena 1D, Atlas, Blaze, C-Interpreter, Catalyst AD, Catalyst DA, Clarity RLC, Clever, Clever Interconnect, Custom IC CAD, DeckBuild, DevEdit, DevEdit 3D, Device 3D, Discovery, DRC Assist, Elite, Exact, Expert, Expert C++, Expert 200, ExpertViews, Ferro, Gateway, Gateway 200, Giga, Giga 3D, Guardian, Guardian DRC, Guardian LVS, Guardian NET, Harmony, Hipex, Hipex C, Hipex Net, Hipex RC, HyperFault, Interconnect Modeling, IWorkBench, Laser, LED, LED 3D, Lisa, Luminous, Luminous 3D, Magnetic, Magnetic 3D, MaskViews, MC Etch & Depo, MC Device, MC Implant, Mercury, MixedMode, MixedMode 3D, MixedMode XL, MultiCore, Noise, OLED, Optolith, Organic Display, Organic Solar, OTFT, Quantum, Quantum 3D, QUEST, RealTime DRC, REM 2D, REM 3D, SEdit, SMovie, S-Pisces, SSuprem 3, SSuprem 4, SDDL, SFLM, SIPC, SiC, Silvaco, Silvaco Management Console, SMAN, Silvaco Relational Database, Silos, Simulation Standard, SmartLib, SmartSpice, SmartSpice 200, SmartSpice API, SmartSpice Debugger, SmartSpice Embedded, SmartSpice Interpreter, SmartSpice Optimizer, SmartSpice RadHard, SmartSpice Reliability, SmartSpice Rubberband, SmartSpice RF, SmartView, SolverLib, Spayn, SpiceServer, Spider, Stellar, TCAD Driven CAD, TCAD Omni, TCAD Omni Utility, TCAD & EDA Omni Utility, TFT, TFT 3D, Thermal 3D, TonyPlot, TonyPlot 3D, TurboLint, Universal Token, Universal Utility Token, Utmost III, Utmost III Bipolar, Utmost III Diode, Utmost III GaAs, Utmost III HBT, Utmost III JFET, Utmost III MOS, Utmost III MultiCore, Utmost III SOI, Utmost III TFT, Utmost III VBIC, Utmost IV, Utmost IV Acquisition Module, Utmost IV Model Check Module, Utmost IV Optimization Module, Utmost IV Script Module, VCSEL, Verilog-A, Victory, Victory Cell, Victory Device, Victory Device Single Event Effects, Victory Process, Victory Process Advanced Diffusion & Oxidation, Victory Process Monte Carlo Implant, Victory Process Physical Etch & Deposit, Victory Stress, Virtual Wafer Fab, VWF, VWF Automation Tools, VWF Interactive Tools, and Vyper are trademarks of Silvaco, Inc.

Style Conventions		
Font Style/Convention	**Description**	**Example**
•	This represents a list of items or terms.	• Bullet A • Bullet B • Bullet C
1. 2. 3.	This represents a set of directions to perform an action.	To open a door: 1. Unlock the door by inserting the key into keyhole. 2. Turn key counter-clockwise. 3. Pull out the key from the keyhole. 4. Grab the doorknob and turn clockwise and pull.
→	This represents a sequence of menu options and GUI buttons to perform an action.	**File→Open**
`Courier`	This represents the commands, parameters, and variables syntax.	`HAPPY BIRTHDAY`
Times Roman Bold	This represents the menu options and buttons in the GUI.	**File**
New Century Schoolbook Italics	This represents the variables of equations.	$x + y = 1$
Note:	This represents the additional important information.	**Note:** Make sure you save often when working on a manual.

Chapter 1
Introduction

1.1 Overview

SmartSpice is a high quality, commercial grade, general purpose circuit simulation program for nonlinear DC, nonlinear transient, and linear AC analyses. SmartSpice simulates a circuit by calculating the behavior of all circuit components simultaneously. Through its many reliable models, SmartSpice bases its simulation on physical properties, as well as electrical parameters, to simulate the behavior of complex circuits. It simulates circuits and subcircuits consisting of resistors, capacitors, inductors, mutual inductors, independent voltage, current sources, dependent sources, transmission lines, switches, and the five most common semiconductor devices: DIODEs, JFETs, BJTs, MOSFETs, and MESFETs.

SmartSpice is the result of intensive research and development; and is superior to all commercially available circuit simulators with competitive speed, superior convergence, high accuracy, and powerful pre- and post-processing. SmartSpice is available on PC windows-based platforms, UNIX-based platforms and Linux PC platforms. The following supported platforms are:

- Windows XP (32 bit), Vista (32 bit or 64 bit), 7 (32 bit or 64 bit)
  (Our Windows applications are 32 bit for compatibility with XP. When Windows Vista is used the Program will be limited to 3GB of memory.)
- Red Hat Enterprise Linux 4, 5, 6 (64 bit)

The size of the circuit that SmartSpice simulates is limited only by the computer hardware and operating system.

1.2 Summary of SmartSpice Analysis Techniques

- DC Analysis
- AC Small-Signal Analysis
- Transient Analysis
- Transfer Function
- Network Analysis
- Sensitivity Analysis
- Noise Analysis
- Distortion Analysis
- Fourier Analysis
- Forward and Reverse FFT
- Monte Carlo/Worst-Case Analysis
- Pole-Zero Analysis

1.3 SmartSpice Special Functions

.ST	Allows repeated calculation while stepping through the model parameter values.
.MEASURE	Measures user-specified circuit activity, thus reducing the volume of output data and minimizing calculation time.
.MODIF	Allows you to investigate the behavior of a circuit for the same sets of parameters as they are modified over a user specified range of values.

1.4 SmartSpice Command Line Syntax

The syntax for the SmartSpice command line is:

```
% smartspice [-help]
[-n] [-u] [-v] [-e <file>] [-pp] [-t <type>] [-gui] [-top <name>]
[-case 0|1] [-alog <file>] [-w <dir>] [-fast] [-flat]
[-startupfile startupfile] [-foreground]
[-P <num> [-nodist]] [-PS <num> [-nodist]] [-PVLG <num> [-nodist]]
[-forcesolver <solver_name>] [-solverprecision <N>]
[-delnet] [-noredir] [-rtilim] [-spectre_dump]
[-eacflow] [-fasteac] [-turbo]
[-pjo] [-pjoc]
[-i|-c|-pp|-b [-sb] infile [-o outfile] [-o <path>/filename]
[-r rawfile]] [-eldo|-hspice|-pspice|-spectre]
[-rf] [-psf_dir <dir>]
[-delsubcktmap] [-dhestat]
[-rcauto -rcratio <num>] [-rclevel <num>] [-rc_optimize] [-rcdump]
[+checkpoint <filename> [+multisave] ]
[-checkpoint +recover:<timepoint>]
[-mps] [-mp] [-mpr <file>] [-mprg <file>] [-a a_file]
```

-help	Command line parameters usage (this message).
-n	**(don't read startup file)** The `SmartSpice.ini` is read when SmartSpice starts up before reading any input deck, and its contents are treated like the contents of a `.CONTROL` block. When this flag is specified, the file is not read. For further details about `SmartSpice.ini`, see "Initialization Scheme" on page 1-27.
-u	**(show usage**) SmartSpice prints a list of the allowed command line flags when this option is used.
-v	**(print version number)** This option causes SmartSpice to print the version number as well as the hostname of the machine on which SmartSpice is running.
-e <file>	Specify error file to output. Windows only.

-pp	**(post-processor mode)** This option allows only the post-processor functionality in SmartSpice and no simulations are possible. For this functionality to work a full SmartSpice licende must exist but is not used.
-t <type>	Specify terminal type.
-gui	Run in interactive mode (default).
-top <name>	Set top subcurcuit.
-case 0|1	0: (default) case sensitivity disabled 1: case sensitivity enabled This command line switch can be used to turn ON/OFF case sensitivity for the following items in a netlist: • Parameter names • Node names • Device names • Model names • Subcircuit names • Data (`.DATA`) names • Library (`.LIB`) entry names • Measure names
-alog <file>	Specify a log file for simulation run-time info.
-w <dir>	Use another directory as temporary.
-fast	Turns on hierarhical parser.
-flat	Turns off hierarhical parser.

Note: The following options affect rawfile generation: POST, PROBE, RAWPTS, NUMDGT and SAVEMEASURES.

-hspice	**(HSPICE compatibility mode)** This option turns on the HSPICE compatibility mode.
-pspice	**(PSPICE compatibility mode)** This option turns on the PSPICE compatibility mode, and causes the same SmartSpice reaction as the command `set simulator=pspice` in the startup file.
-startupfile	**(specify alternative startup file)** This option allows you to specify the `startup` file instead of the default `SmartSpice.ini`.

Example

```
smartspice -startupfile MyInitialization.ini
```

If SmartSpice is invoked with no options, it starts in the Windows mode.

Note: The following option **-foreground** only applies to the Windows operating system.

-foreground	This option applies only to the batchmode operation (-b), and is used as the first switch after SmartSpice to change the operation from background to foreground (i.e. the prompt only returns after the simulation is complete). This can be used to start jobs in a sequence run for maximum tool license use. (See the Note above)
-P n_cpus	(**run in parallel on several CPUs)** On multi-threaded platforms, for certain models, SmartSpice can distribute over the requested number of processors (n_cpu).
-PS <N>	Allow solver use N processors.
-PVLG <N>	Specify number of threads for Verilog-A.
-forcesolver	**(solvername)** The specified solver (solvername) will be used during simulation. This command line argument will override the solver, specified in the netlist.
-delnet	Remove netlist description before simulation.
-noredir	Skip output redirect for remote agent.
-rtilim <num>	Set limit on run-time item id for remote agent.
-spectre_dump	Dump a converted Spectre netlist.
-eacflow	Turns on EAC automatic flow.
-fasteac	Turns on hierarhical parser and EAC automatic flow.
-turbo	Turns on a faster mode of SmartSpice to get a quicker simulation (see 27.1 Performance Control Using Command Line Mode '-turbo' for further details).
-pjo	Remote mode: Enable per-job output.
-pjoc	Remote mode: Enable per-job output, combined.
-i	**(window mode)** This option forces SmartSpice to run with a graphical user interface, and is Qt-based on UNIX platforms.
-c	**(command mode)** A CSH-like command line user interface is used to run SmartSpice interactively. For a summary of the available SmartSpice commands see Chapter 5 Commands.

Example

```
smartspice -foreground  -b  circuit.in  -r  Rawfile.raw
```

-b	**(batchmode)** SmartSpice parses the input file `infile` and performs the required analyses.

For displaying text information in batchmode, by default, SmartSpice uses the standard output/error streams. By using UNIX shell commands, it's possible to redirect these output streams to file. For an example of C shell, see the following the command line:

```
smartspice -b exl.in > & result.out
```

Output results and error messages will be saved in the file `result.out`.

-sb	**(silent batchmode**) SmartSpice parses the input file `infile` and performs the required analysis. The differences between batchmode and silent batchmode are as follows: • Silent batchmode works without GUI features (windows, message boxes, plots and so forth). • Silent batchmode redirects all the output information to a file that is specified using the command line option [-o outfile], or automatically to the file `<input_deck_name>SB.out` if [-o outfile] commands are absent. • Silent batchmode doesn't draw in SMARTVIEW.

Example

```
smartspice -sb circuit.in -o alloutput.out
smartspice -sb circuit.in
```

SmartSpice parses the input file `circuit.in` and performs the required analysis. In the first example, SmartSpice will redirect all output information in an `alloutput.out` file. The second example will redirect all output information in `<input_deck_name>SB.out` file in the input deck's directory automatically.

-o	**(output file)** During batchmode simulation, messages are printed on the screen by default. This option redirects messages to the file `.out`.
-o <path>/ filename	Specifies the name of the output file as `filename`. The optional file path `path` specifies an existing directory. If a `path` is not specified, SmartSpice saves the output file in the same directory with the input file. When the `-b` and `-hspice` options are specified, SmartSpice saves all output files (i.e., output log, raw (`.tr*`, `.sw*`), measure (`.ma*`, `.mt*`, `.ms*`), hierarchy (`.pa*`), epic (`.epc`)) in the existing directory `path`. These files will have the same name `filename` but different corresponding extension. The input and output file names can be different.

-r	**(rawfile)** After batchmode simulation, output variables saved during simulation are stored in a rawfile. The option `-r` is used to specify the name of the file where the results are stored. SmartSpice loads rawfiles from prior simulations, compares results, generates plots, and so forth.
-rf	Run SmartSpice RF simulator.
-psf_dir <dir>	Specify PSF directory.
-delsubcktmap	Delete subcircuit map after sorcing in -fast mode.
-dhestat	Debugging information for -fast mode.
-rcauto	Turns on RLC-reduction tool when number of passive elements > `rcratio`.
-rcratio <num>	Set ratio for `-rcauto`. Default is 5. `ratio` = number of passive/number of active.
-rcdump	Save expanded netlist into the file after RLC-reduction.
-rclevel <N>	Use RCL-reduction tool control, where N selects tool reduction mode: 0: subcircuits (default) 1: subcircuits with post-filtering 2: subcircuits and top-level 3: subcircuits and top-level with post-filtering 4: expanded netlist 5: expanded netlist with post-filtering 6: DSPF annotated netlist 7: DSPF annotated netlist with post-filtering
-rc_optimize	Run optimized RLC-reduction. SmartSpice will automatically adjust `RMIN_RC` option to get the most reduced resistors network.
+checkpoint	This allows checkpoint files to be created under transient analysis simulations.
-checkpoint	This allows checkpoint files to be recovered from a previous transient analysis simulation.

SmartSpice is capable of recovering from an unexpected program termination during a transient analysis simulation. This functionality is performed through so called checkpoint files and is disabled by default. It can be activated through the command line options `+checkpoint/-checkpoint`.

There is a restriction for this feature, the simulation (saved in batch mode) cannot be recovered in GUI mode, and vice versa. A warning will be issued if this recovery method is tried (see Section 6.9 Saving and Recovering of a Transient Analysis Simulation).

+checkpoint [+multi-save]	Enable SAVE (according .TRAN specifications).
+checkpoint +recover:<timepoint>	Enable RECOVER at 'timepoint' instant.
-mps <N>	Allows you to run parallel simulation on multiple CPUs. The number of CPUs can be limited by specifying the number n. If the number n is not specified, SmartSpice will use all available CPUs during the simulation.
-mp <N>	If the deck contains analysis with sweep, SmartSpice will generate new circuit for every sweep step, and write out each circuit in the same directory. Then SmartSpice will create extra child processes. Each child process will handle a separate circuit. The parent process waits for the children to finish. If -mp 4 is specified, then 4 extra processes will be created. The total amount of SmartSpice processes in memory is 5, including 4 child processes and 1 parent. After finishing the simulation, the child will remove its own circuit simulation file. The .out and .raw files will have the suffix -n, which says what sweep step was simulated. This functionality is similar to the parallel alters functionality.
-nodist	**(do not distribute for .ALTER)** See the description of the .ALTER statement for details of this flag.
-solverprecision <N>	(N=64|80|128|160|256|320) The XMS and SPEEDS solvers will use N-bit extended precision during simulation. N could be 64, 80, 128, 160, 256, 320 on Linux 64. On Windows = 64. By default, N=64.

Example

```
smartspice -b -hspice   ex1.in  -o /home/user/result
```

In this example SmartSpice parses the input file ex1.in and saves all output in the directory /home/user. The output files will have the result prefix name (for example result.tr0 or result.mt0).

-e	**(error file)** On Windows, during batchmode simulation, error messages are printed to a file.

Note: The following options affect the data output by SmartSpice: BRIEF, NOMOD, FORMAT, ACCT and NUMDGT.

-psf_dir name_of_the_new_directory	Specifies a folder where PSF files will be saved. By default, PSF files are placed into a `../psf` directory.

Example

```
smartspice -psf_dir name_of_the_new_directory
```

Note: The following option is used to work with a remote computer with installed Unix/Linux operating systems without X server.

Syntax

```
smartspice -top <top_subckt_name>
```

top_subckt_name	A subcircuit name which will be used as a top-level subcircuit (See Section 1.10 HSIM Compatibility).

-n	Do not read startup file.
-hsim	Use HSim ompatibility mode.
-mpr <file>	Distributive processing in batch mode.
-mprg [file]	Distributive processing in GUI mode.
-200	Run SmartSpice200 which is limited to 5 Verilog-A modules or subcircuit calls and/or 10 transistors and/or 50 elements.
-delsubcktmap	Cleanup internal tables and release memory before simulation starts.
-slver	Prints out modellib interface version.
-debug	Invoke Verilog-A debugger in batch mode.
-a a_file	Saves whole listing of the current circuit into one `a_file`.

Initialization Scheme

The initialization file (or `startup` file) is read when SmartSpice starts, before reading any input deck, and its contents are treated like the contents of a `.CONTROL` block. The initialization scheme defines the name of the initialization file and its location. In the old initialization scheme, the name of the initialization file was different on Unix and Windows, and SmartSpice could read this file (or files) from many locations. Now SmartSpice supports

a new initialization scheme. The default name of the initialization file is `SmartSpice.ini`, and is the same on Unix and Windows. The algorithm of loading the initialization file is:

1. When SmartSpice starts with the option `-startupfile filename` on the command line, SmartSpice will source `filename`. If sourcing is successful, the initialization phase is finished, otherwise the step 2 will be done.
2. SmartSpice sources default `SmartSpice.ini` from the program (executable) directory first. If sourcing is successful, the initialization phase is finished, otherwise SmartSpice sources the default `SmartSpice.ini` from the home directory. Note, for Unix the home directory is usually set to `/home/username`, where `username` is the login name. For Windows, the home directory is usually set to `USERPROFILE`, where `USERPROFILE` is the Windows environment variable for the current user. For example, user `kon` `USERPROFILE` = "`WinPartition(C:)\Documents and Settings\kon`".

1.4.1 Solver Multithreading Control

A command line option has been introduced to control solver multithreading. Specifying the option `-PS [x]` sets the desired number of threads to be used by the solver. SmartSpice uses the following algorithm:

- If the option `-PS [x]` is supplied, the SmartSpice solver will use `x` CPUs.
- If only `-PS` is supplied, the maximum number of available CPUs will be used).

Command Line Options	Action
`-P -PS`	Core and solver will run on maximum available CPUs
`-P 4 -PS 2`	Core will run on 4 CPUs and solver will run on 2
`-PS 2`	Core will run on maximum available CPUs and solver will use 2 threads

1.4.2 Output Thread Availability and Use Reported

```
******  SmartSpice Threads Information  ******

Command Line Model Threads Count            :       1
Command Line Solver Threads Count           :       1
Available CPU Count                         :       8
Actual Model Evaluation(Load) Threads Count :       1
Actual Solver Threads Count                 :       1
```

1.5 SmartSpice Modes

SmartSpice has 2 mode levels: compatibility modes and sub modes.

	Mode name	Mode command key

Compatibility modes:	• HSPICE compatibility mode
	• HSIM compatibility mode
	• PSPICE compatibility mode
	• SPECTRE compatibility mode
	• ELDO compatibility mode
		`-hspice`
		`-hsim`[a]
		`-pspice`
		`-spectre`
		`-eldo`

Sub modes:	• Fast sub mode or hierarchical parser sub mode(default sub mode)
	• Flat sub mode. Turns off all fast sub mode features. For more details, see Section 1.10.4 Command line option flag -fast (smartspice -fast).
	• EAC automatic flow sub mode
		`-fast`
		`-flat`
		`-eacflow`

a.Compatible with HSIM syntax and outputs, but retains SPICE accuracy.

Compatibility modes cannot be combined with each other, but can be combined with any sub modes.

Examples

```
smartspice -flat
smartspice -spectre -eacflow
```

In the first example, SmartSpice runs in normal flat submode (by default, hierarchical mode is used).

The second example will run SmartSpice in SPECTRE compatibility mode and turn on EAC automatic flow sub mode.

1.6 Hspice Compatibility Mode

The command line option `-hspice` can be used to turn on the HSPICE compatibility mode. In general, SmartSpice maintains a high degree of compatibility with other SPICE simulators. However, there are occasions when a standard SmartSpice behavior has been in use for some time, and conflicts with other SPICEs. This flag is useful in distinguishing between the two behaviors.

If the input deck is specified in the `-hspice` batchmode as a link, rawfiles will then be created in the link directory.

SmartSpice supports redirection of rawfiles, which were created in Hspice compatibility mode, using the option `-r`. You can use this feature to collect all rawfiles in one location, which has more free space and other advantages.

Example

```
smartspice -hspice -b circuit.in -r RawfilesLocation/
Results.raw
smartspice -hspice -b circuit.in -r RawfilesDirectory
```

In the first example the rawfile `Results.raw` will be created in the directory `RawfileLocation`. The second example will create a rawfile `circuit.raw` in the directory `RawfilesDirectory`.

Now, if the variable simulator is set to `-hspice`, all trig functions use degrees instead of radian for arguments, and outputs units and the default value of the parameters:

TNOM	The normal temperature.
TEMP	The Circuit operating temperature.

Will be set to 25 degrees C.

1.6.1 Measurement Results

When the option `-hspice` is specified on the command line, and a measurement fails due to an `out of interval` error, a warning is issued and a zero result is returned for that variable. This contrasts with the standard SmartSpice behavior in which an error is issued and no result is returned for that variable.

1.6.2 .ALTER and *.mt# Files

While running in batchmode with the option `-hspice`, SmartSpice correctly outputs the header information in each `.mt#` file using `alter#` as a header for the right most column when the deck contains a `.ALTER` statement, and `index#` otherwise. The data in the right most column is the alter index or vector index, respectively.

1.7 PSpice to SmartSpice Conversion

To run PSpice decks in SmartSpice, use the command:

```
smartspice -pspice .....
```

. The following is a sample of PSpice deck format:

```
.NODESET         V(N_0001)=1
.NODESET         V(N_0002)=1
.IC          V(Vxxx_1 )=0
.IC          V(Vxxx_2 )=0
.IC          V(Vxxx_3 )=0
.IC          V(Vxxx_4 )=0

** Analysis setup **
.tran 1.520ms 1.520ms 0 2us
.OPTIONS RELTOL=0.001
.TEMP 27

* From [PSPICE NETLIST] section of pspice91.ini:
.lib "library1.lib"
.lib "library2.lib"
.lib "library3.lib"
.lib "library4.lib"
.inc "include1.inc"
.inc "include2.inc"
.inc "circuit1.cir"

.INC "test_case.net"
.INC "test_case.als"
.INC "test_case__probe.inc"
.OPTIONS POST
.OPTIONS PROBE
.OPTIONS RAWPTS=100

.probe I(VIN)
.probe I(L1)
.probe V(SW)
.probe V(vout)

.END
```

To read the PSpice model libraries, use `set use_syntax0_libs=true` before running a SmartSpice simulation.

The following are procedures needed to be taken before sourcing the deck:

Step 1

1. Use the command `smartspice -pspice .....`
2. Type `set use_syntax0_libs=true` in the SmartSpice main command window.

Step 2

1. Put `.LIB` entry name and `.ENDL` inside each library file.
2. Remove all `$` in the spice deck because `$` is used for comment statements.
3. Remove the spaces of `+` inside the include file.
4. Globally change `{` to `(`, and `}` to `)` inside the include file.
5. Comment out the alias file in the spice deck.
6. Change the `LEVEL` to be 49 in the library file.
7. Change to `.tran 1.520us 1.520ms 0 2us UIC` in the `.CIR` file.
8. Change to `.probe I(V_VIN)` in the `.CIR` file.
9. Change to `.probe I(L_L1)` in the `.CIR` file.

If you run the simulation in GUI mode, you can also open **Set window under System** in the main SmartSpice window. You can type `simulator=pspice` and then press **Set**. Also, type `use_syntax0_libs=true` and then press **Set**. Once you run the simulation, you can immediately go into PSpice mode.

If you run the simulation in batchmode in UNIX or LINUX, you can save these two by setting `set simulator=pspice` and `set use_syntax0_libs=true` in `SmartSpice.ini`. Once you type a SmartSpice command, you can immediately go into PSpice mode.

If you run the simulation in batchmode in Windows, you can save these two by setting `set simulator=pspice` and `set use_syntax0_libs=true` in `smspice.set`. Once you type a SmartSpice command, you can immediately go into PSpice mode.

Inability to use an include file. This problem is caused by a difference in syntax between PSpice and SmartSpice for the `.probe` command. To resolve this issue the `.probe N(x)` needs to be changed to `.probe V(x)`. This change will allow the `probe.inc` file to be imported into the simulation.

After taking the conversion procedures, the input deck would be something like in the following:

```
.NODESET         V(N_0001)=1
.NODESET         V(N_0002)=1
.IC          V(Vxxx_1 )=0
.IC          V(Vxxx_2 )=0
.IC          V(Vxxx_3 )=0
.IC          V(Vxxx_4 )=0

** Analysis setup **
.tran 1.520us 1.520ms 0 2us UIC
.OPTIONS RELTOL=0.001
.TEMP 27

* From [PSPICE NETLIST] section of pspice91.ini:
.lib "library1.lib" TT
.lib "library2.lib" TT
.lib "library3.lib" TT
.lib "library4.lib" TT
.inc "include1.inc"
.inc "include2.inc"
.inc "circuit1.cir"

.INC "test_case.net"
*.INC "test_case.als"
.INC "test_case__probe.inc"
.OPTIONS POST
.OPTIONS PROBE
.OPTIONS RAWPTS=100

.probe I(V_VIN)
.probe I(L_L1)
.probe V(SW)
.probe V(vout)

.END
```

Remember to use `set simulator=pspice` and `set use_syntax0_libs=true`. These two commands will make PSpice to SmartSpice conversion easier, and produce a more accurate and faster run time than PSpice.

1.8 Spectre Compatibility

1.8.1 Initialization of Spectre Compatibility Mode

To activate compatibility mode, SmartSpice should be launched with the command line key `-spectre`.

1.8.2 Common Suggestions and Known Limitations

The mode supports any combination of SmartSpice and Spectre netlists, connected to each other with `include` or `lib` statements. Spectre device instance names will be converted to SmartSpice compatible names.

Example

```
device1 1 2 resistor r=100    - Spectre line
r_device1 1 2 r=100           - SmartSpice converted line
```

To support compatibility, each converted device name will have the artificial prefix `?_` in accordance with the model referenced, where `?` is the appropriate first letter, recognized by SmartSpice (`m` - for mosfet, `r` - for resistor, etc.).

Name resolver sets the following rules on file combinations:

- SmartSpice device instances may reference SmartSpice or Spectre models;
- Spectre device instances should reference Spectre models only, these models should be defined in the same file or be included from external files.

Any Spectre file should satisfy the following requirements:

- The first line is ignored.
- At the very beginning of the file, the following statement indicates the Spectre format: `simulator lang=spectre`. Mixed language files (Spectre and Spice statements) are not currently supported.

When a Spectre file is loaded into a SmartSpice GUI, and Spectre compatibility mode is activated, the file will be automatically converted to the SmartSpice format. It allows for easy inspection of the conversion results and process errors generated by SmartSpice.

1.8.3 Supported Features

The library supports translation of common statements, device instances, models and analysis.

The following common statements are supported:

- `options`
- `save`, supports the following arguments:
- `parameters`
- `include`
- `ic`
- `global`

Any device instance is supported if it references a known model. If model is unknown or not found, the instance is processed as a Verilog reference with the prefix `yvlg_` added.

The following models are supported:

- `resistor`
- `capacitor`
- `inductor`
- `vsource, pccvs, pvcvs, svcvs, vcvs, zccvs, zvcvs`
- `isource, cccs, pcccs, pvccs, scccs, svccs, vccs, zcccs, zvccs`
- `diode`
- `bjt, bjt504, bjt504t, bjtd504, bjtd504t`
- `bht`
- `hbt`
- `vbic`
- `jfet`
- `mos1, mos2, mos3, bsim1, bsim3v3, bsim4`

The following analysis are supported:

- ac analysis
- dc analysis
- tran analysis

1.8.4 Configuration

The library allows you to configure parameters processing for analysis, models and device instances. To add a configuration, the following statement(s) should be added to the netlist in process:

```
spconfig {
    line1 ...
    line2 ...
    line3 ...
    ...
    lineN ...
}
```

Where `line` is: `type name action parameters`.

- `type`: The type of processed statement (model, analysis or device).
- `name`: The name of the statement:
  - `model`: The built-in model name.

- analysis: The built-in analysis name.
- device: The built-in device name.

- action: The action type (add, delete or change).
- parameters: The required parameters:
  - add: name=parameter_name value=parameter_value
  - delete: name=parameter_name
  - change: oldname=parameter_name newname=parameter_name
    [value=parameter_value]

If value is not specified, a new parameter will use the old value.

If value is specified, it is possible reference the old value as !value!.

Example

```
spconfig {
       model model1 add name=p1 value=1
       model model2 delete name=p2
       model model3 change oldname=p2 newname=p200
       model model4 change oldname=p2 newname=p200 value=100
          model model5 change oldname=p2 newname=p200 value=10 *
!value! - 5
       analysis tran add tranopt=5
       device model5 add p1=1
       }
```

This statement may be added to any part of Spectre netlist.

In a case a separate configuration file is created, it should be included into the Spectre netlist with the following statement:

```
include "myfile.ext"

File format:

* First line is ignored
simulator lang=spectre
spconfig {
...
}
```

1.8.5 SmartSpice Syntax Inclusion

It is possible to add SmartSpice specific statements to the netlist in process. If a top level netlist is in Spectre format, it is necessary to create a separate file with SmartSpice statements, and include it into the Spectre netlist using the following statement:

```
include "filename.ext"
```

1.8.6 Spectre Compatibility

A new command line option -psf_dir has been added to SmartSpice.

Example

```
smartspice -psf_dir name_of_the_new_directory
```

This option specifies a folder where PSF files will be saved. By default, PSF files are placed into a ../psf directory.

1.8.7 Spectre PSF Data Format

For both ASCII and binary versions of the PSF data, the format changed when Spectre was moved from sitting under the Cadence IC5x directory to its own installation MMSIM tree. This is what we will call “old” format to “new” format.

Running SmartSpice in `-spectre` mode uses the “new” format which is compatible with the PSF reader in SmartSpice. The “old” format is for customers running SmartSpice under the old socket integration in the Cadence environment, which is no longer supported.

1.8.8 Transient Analysis Timestep Controls in SPECTRE Mode

In `-spectre` mode SmartSpice will adjust the timestep (`TSTEP`) and the parameter `TMAX` (maximum internal timestep) if it was not given as option or transient analysis parameter. Timestep (`TSTEP`) will be set to 1/10th of minimum `trise/tfall` time in PWL or PULSE sources or to 1/10th of minimum period in SIN sources. The parameter `TMAX` will be adjusted to `'RMAX * minimum trise/tfall'` in PWL or PULSE sources or to `'RMAX * TSTEP'` in case of SIN sources. It prevents SmartSpice from making very large steps in some cases and helps to simulate oscillators.

1.8.9 Spectre Compatibility for ahdl_include

Syntax

```
ahdl_include "filename"
```

where

`filename`	Verilog-A source file
`-master`	option

Use this statement to specify Verilog-A source file `filename` in SPECTRE compatibility mode. Several `.ahdl_include` cards can be used.

Note: See `.verilog "filename" [<module_name>]`

Example

1. Verilog-A soure file capacitor.va:

```
...
module capacitor (vpos, vneg)
...
```

2. SmartSpice netlist:

```
simulator lang=spectre
ahdl_include "capacitor.va"
```

1.9 Spectre Compatible Syntax for Output Statements

The following is Spectre compatible syntax:

`.SAVE node_name`	Saves a `node_name` voltage.
`.SAVE :currents`	Saves currents of all devices.
`.SAVE :pwr`	Saves a circuit dissipated power.
`.SAVE :all`	Saves all node voltages.
`.SAVE :static`	Saves all resistors currents.
`.SAVE dev_name:currents`	Saves a device current.
`.SAVE dev_name:pwr`	Saves a device dissipated power.
`.SAVE dev_name:symbol`	Saves current on terminal of device. The terminals are specified by the device type. Support devices: J, M and Q.
`.SAVE dev_name:x`	Saves current on device with terminal number `x`. Support devices: J, M and Q.
`.SAVE xcall:num`	Saves a current for specified subcircuit node.

Where:

`node_name`	a name of node
`dev_name`	a device name
`num`	a terminal number
`symbol`	a special terminal symbol

For two-terminal devices the symbol takes the values:

`p`	a positive terminal (first terminal)
`n`	a negative terminal (second terminal)

For J-devices the symbol takes the values:

`d`	a drain terminal
`g`	a gate terminal
`s`	a source terminal

For M-devices the symbol takes the values:

d	a drain terminal
g	a gate terminal
s	a source terminal
b	a bulk terminal

For Q-devices the symbol takes the values:

c	a collector terminal
b	a base terminal
e	an emitter terminal
s	a substrate terminal

Output statement processing has been improved. The backslash feature supports the following set of symbols: | : + - < > ( ) \.

SPECTRE library supports the following:

1. Autodetect S-parameter data file format.
2. Set transformer parameters n1 and n2 to 1 by default.
3. Improved handling of STEP parameter in AC analysis statement. If step parameter is specified in SPECTRE netlist, the output statement is switched into the following syntax form:

   ```
   .AC START=<fstart> STOP=<fstop> STEP=<fstep>
   ```
4. Added support of '\-' and '\+' combinations in the SAVE statement.

Example

```
.SAVE :pwr
.SAVE r1:p d1:n
.PLOT m1:2 m1:3
.PRINT q2:c q2:e

.subckt Inverter ip_inv op_inv
m_M0 op_inv ip_inv 0 0 Nch w=15u l=12u m=1
m_M1 op_inv ip_inv vcc! vcc! Pch w=20u l=10u m=1
.ends Inverter
x_I0 net9 net18 Inverter
.print x_I0:1 x_I0:2
```

In the last example, SmartSpice prints values of current from the subcircuit external nodes with number 1 and 2. The numbering of subcircuit nodes begins from 1.

1.10 HSIM Compatibility

1.10.1 Initialization

The HSIM simulator simplifies the calculations of DC operating point and transient analysis waveforms to allow it to simulate large hierarchical circuits. SmartSpice has a `-fast` command line mode, which also allows it to simulate large hierarchical circuits but importantly still retain the true SPICE accuracy. The -FAST sub mode uses input data reduction when hierarchy is met in the netlist, and effects only the parser and preprocessor stage. FAST sub mode is only for subckt parameters, X calls, model cards (binned also) and M, R, C devices, (full parametrization is supported). In the -FAST sub mode SmartSpice evaluates hierarchical circuits using SIZE and SPEED optimization in the preprocessing stage of reading the input deck.

In `-hsim` mode, SmartSpice activates `-fast` sub mode, and a number of features have been added to allow HSIM input decks to be used in SmartSpice.

1.10.2 Supported Parameters

HSIMVECTORFILE	Digital input and output files for HSIM can be specified: `.param HSIMVECTORFILE=name.vec`
HSIMDELVTO	If parameter `HSIMDELVTO` is specified in the netlist, the value `delvto_value` will be assigned to the instance parameter `DELVTO` for all supported individual transistors in the circuit or for all transistors in the subcircuit `subcircuit_name`. `.param HSIMDELVTO=devto_value <subckt=subcircuit_name>`

Example

```
.PARAM HSIMDELVTO=0.5 subckt=sub_inv
```

This example sets the `delvto` parameter to 0.5 for all transistors in the `sub_inv`.

HSIMTOP	A subcircuit name which will be used as a top-level subcircuit. `.PARAM HSIMTOP=<top_subckt_name>`
HSIMCHECKMOSBULK	When `HSIMCHECKMOSBULK=1` SmartSpice will check the bulk node of all MOSFETs in the input desk. If the bulk node of a NMOSFET transistor is greater than 0.5v or the bulk node of a PMOSFET transistor smaller than 0.5v SmartSpice will print the following table:

`HSIMCHECKMOSBULK` output during Transient Analysis, 27 °C:

```
time device model type vbs condition
0.0000e+000 mp00 pm0 pmos 0.0000e+000 < 0.5
2.0000e-012 mp00 pm0 pmos 4.0000e-004 < 0.5
2.6800e-009 mn00 nm0 nmos 5.3600e-001 > 0.5
2.7200e-009 mn00 nm0 nmos 5.4400e-001 > 0.5
```

Where:

time	analysis time
device	device name
model	model name
type	model type (nmos or pmos)
vbs	the bulk voltage
condition	< 0.5 or > 0.5

HSIMVECTORFILE	Digital vector input and output files for HSIM can be specified with HSIMVECTORFILE parameter. .param HSIMVECTORFILE=vec1

1.10.3 Post Processing

SmartSpice supports optional HSIM syntax for output dot statements in all simulation modes by default.

Syntax

```
.PRINT <anytype> outvar1 <outvar2 ...>
+ <subckt = sub_name> <level = val1> <filter = pattern>
```

The following optional settings are supported in this release:

subckt=sub_name	The output applies to the specified node name(s) and/or element name(s) within all instances of the specified sub-circuit name. This subckt setting is equivalent to placing any output statements (.print, .save, .plot and .probe) within the sub-circuit definition.
level=val	This setting is effective only when the wildcard character is specified in the output variable. The level value val specifies the number of hierarchical depth levels when the wildcard node/element name matches. When val is set to 1, the wildcard match applies to the same depth level where the output statement is located. When val is set to 2, it applies to the same level, and to one level below the current level where output is located. When val is set to -1, the wildcard match applies to all the depth levels below and including the current level of output statement. The default value of val is -1.
filter=pattern	This setting is effective only when the wildcard character is specified in the output variable. Nodes/elements that match the pattern specified in the filter clause will not be printed.

Examples

```
.print v(x1.*) v(q*) subckt=sub1
```

```
.save v(*) level=1
.print v(x1.*) level=3
.print v(x1.x2.*) filter= x1.x2.n*
```

In the first example, SmartSpice prints node voltages for all nodes which match `x1.*` and `q*` in all instances of subcircuit `sub1`.

The second example saves all top-level node voltages.

The third example prints node voltages within sub-circuit instance `x1`. The printout includes nodes within `x1`, and the nodes in two levels below `x1`. The nodes located more than two levels deeper than the level of instance `x1` are excluded.

The fourth example prints the voltages of all nodes in `subckt x1.x2` that do not start with `n`.

1.10.4 Command line option flag -fast (smartspice -fast)

In `-fast` sub mode, SmartSpice turns on/off some features to efficiently simulate large hierarchical circuits:

1. Turns on SmartSpice variable `dhe_expr`.

This variable allows significant speed up in the parser stage if A, R, C, and L elements are defined in the underlying subcircuit cells, and contain values as expressions.

2. Turns off a variable `subcktcheck`.

This variable allows SmartSpice to skip the checking of subcircuits which are defined with the same name. Default is TRUE (do the check) in all simulator modes (except `-fast/-hsim` mode).

3. Turns on HSIM style parameters (in `-hsim` mode only)

Some optional features are also available in `-fast` sub mode and may manually be set:

4. Setting a variable `dhe_level` to `true` turns on the mechanism for exploiting latency for resistor, capacitor, inductor, and E, F, G, H elements if parameters for these elements are set using expressions (e.g., `R n1 n2 r='a+b*v(out)'`). `dhe_latency_tol` sets the tolerance for `dhe_level`. Default is 1e-03.

Note: Those SmartSpice variable must be specified in the `SmartSpice.ini` file or in the .control block in the input deck. For more details on the variables, see Section 5.5 Variables.

1.10.5 Statistics and Debugging Option Flag for Hierarchical Tables

```
smartspice -fast -dhestat  …..,
```

Where `-dhestat` is an optional commandline flag for debugging that works only in -fast mode, and prints detailed information on all X- subcircuit calls of particular cell and the number of instances in that cell after sourcing an hierarchical input deck.

1.11 Eldo Compatibility

Eldo compatibility consists of control language syntax compatibility and model compatibility.

To activate Eldo compatibility mode, SmartSpice should be run with `-eldo` command line argument:

```
smartspice -eldo [rest of the command line]
```

It is possible to activate Eldo compatibility mode using `SmartSpice.ini` file. The following string should be present in the file:

```
set eldo = true ; to activate Eldo compatibility mode
set eldo = false ; to deactivate Eldo compatibility mode
```

Command line argument `-eldo` always overrides initialization file property (i.e., has highest priority).

1.11.1 Control Language Syntax

Comment Strings and Blocks

The character '!' comments a string from current position to the end of the line.

Example

```
ma1 n1 n2 n3 n4 MODN ! comment here
```

To comment a block inside of a netlist `#com` and `#endcom` may be used.

Example

```
#com
comment strings
are here
#endcom
```

1.11.2 Expressions

Single quotes (`‘ ’`) can be used in expressions.

Example

```
.model nmod nmos vt0='val2*val3+1.5'
```

1.11.3 Bus Commands

Implemented the following bus commands: `.SETBUS`, `.PLOTBUS`.

Command `.SETBUS` creates a bus:

```
.SETBUS bus_name bit_name_0 {bit_name_n}
```

where:

- `bus_name` is a name of created bus,
- `bit_name` is a name of bit in the bus (the first bit is the most significant).

Example

```
.SETBUS mainbus b0 b1 b2 b3
```

The command `.PLOTBUS` plots all bits in a bus:

```
.PLOTBUS bus_name [VTH[1]=value1 [VTH2=value2]]
```

where `bus_name` is a name of a bus created by `.SETBUS`.

VTH[1]=value1, if this voltage threshold is set, analog signal is plotted as a signal.

VTH2=value2, when the second voltage threshold is set the command calculates a value with the following:

When only VTH1 is defined:

```
value < VTH1, logical 0,
value > VTH1, logical 1.
```

When both VTH1 and VTH2 are defined:

```
value < VTH1, logical 0,
value > VTH1 and value < VTH2 , logical X (indeterminate value),
value > VTH2, logical 1.
```

1.12 Use ‘spiceserver’ to Cut Down License Check-out Time

(for UNIX and LINUX only)

`spiceserver` is a simple program to improve the license check out time between two consecutive batchmode simulations. This program contains three tasks:

1. Start a license.

```
spiceserver -start [-n <project name>] [-h <host>]
                   [-v <smartspice       version>]
```

2. Run simulation.

```
spiceserver [-run] -f <deck file> [-o <output file>] [-e <error
file>]
            [-r <result file>] [-n <project name>]
```

3. Return license(s).

```
spiceserver -stop [-n <project name>]
```

or

```
spiceserver -stopall
```

Usage:

1.	**-start**	Check out the license.
	-n	Specifies project name. Each project name will check out one SmartSpice license. If `-n` is not specified, `spiceserver` will only allow one license to start.
	-h	Specifies where to run the simulation. If `-h` is not specified, the simulation will run on the local machine.
	-v	Choose a version of SmartSpice. If `-v` is not specified, SmartSpice will use the default version.
2.	**-run**	Start the simulation.
	-f	Specifies input file name.
	-o	Specifies output file name.
	-e	Specifies the error file name.
	-r	Specifies the rawfile name.
	-n	Simulation will run on specified project if applicable. Otherwise, SmartSpice will run on the default project.
3.	**-stop**	Return the license.
	-stopall	Return all licenses.

Note: Note: If `-stop` or `-stopall` is not specified, all the license(s) that have been checked out will remain active.

Example

A script can be written to automate these processes as follows:

```
spiceserver -start
spiceserver -f inputdeck.in -o outputfile.out -r rawfile.raw
spiceserver -f inputdeck2.in -o output2.out -r rawfile2.raw
spiceserver -stop
```

The first statement starts a project and checks out a license. The second statement simulates `inputdeck.in`, and generates `outputfile.out` and `rawfile.raw`. The third statement simulates `inputdeck2.in`, and generates `output2.out` and `rawfile2.raw`. The fourth statement returns the license.

1.13 Parallel SmartSpice

1.13.1 Parallel Computing

Shortening design cycles through a design methodology can be achieved using many tools and techniques. In the case of analog or mixed-signal design and verification, especially in the IC segment, SPICE simulation is a recognized bottleneck. The throughput of a design team can be hampered by very long device-level SPICE simulations.

In the physical verification arena for example, complex chips tend to yield millions of parasitic elements that are grafted into an extracted post layout netlist. SPICE simulations on this type of circuit typically last days per run. For all practical purposes, SPICE simulators today can only handle partial chip level netlists, like a critical path or a clock tree within a multi-million transistor chip.

Techniques like netlist reduction or table lookup methods to speed up SPICE simulations are common place now. Furthermore, parallel computing has emerged as a viable add-on to shave off some more simulation time. Using multi-CPU computers to run simulations is spreading quickly in design companies for three main reasons:

1. Multi-CPU computers have become affordable to even small startup companies.
2. Few software vendors have produced commercial-grade parallel versions of their software.
3. A multi-CPU computer tends to pack more memory than a single processor computer, which in turn speeds up single-CPU simulation runs by avoiding paging.

Silvaco has pioneered the EDA industry in the field of parallel computing, jumping on the bandwagon at an early stage during the nineties. Several Silvaco TCAD driven CAD tools support parallel computing, including SmartSpice, its industry standard analog simulator. For many years, Silvaco customers have been able to run SmartSpice on multiprocessors, be it Sun servers, or multiprocessor PCs running Windows or Linux.

Nevertheless, performance of a parallel application is still highly misunderstood. “Why isn't my application going four times faster using 4 CPUS?” is a frequently asked question.

1.13.2 Parallelization Models in SmartSpice

Cray vector computers feature multiple adders and multipliers within a CPU. No matter how your application software was written, Cray would make it go faster, as long as your application dealt mainly with arrays of data and performed vector operations. Thinking Machine (TMC) Computers using a two dimensional array of ALUs used the same approach. The same instruction of code was executed at the same time on a two dimensional array of data. This is known as the SIMD (Single Instruction, Multiple Data) computing model.

Unlike vector and SIMD computers, today's multiprocessor computers are built using commodity off-the-shelf microprocessors. The application programmer is left with two basic models of parallel computing:

1. **Task parallelism:** You have many unrelated tasks to perform and you distribute them among the available processors.
2. **Data parallelism:** You only have one task to perform on a large data set. Distribute the data on the processors, and each will perform the same task on its subset of data.

Task parallelism is well suited to DOE (Design Of Experiment) type scenarios where you run multiple unrelated simulations. In this case, an instance of the simulator is forked for each data set, and they all run in parallel on the same machine. This is the pinnacle of parallel computing. Barring shared resource contentions, like memory and I/Os, using 4 CPUs to run

4 simulations would effectively yield a 4x speedup compared to running all 4 simulations on one CPU. This is the approach taken by SmartSpice when a .ALTER statement is encountered in the command input file. A .ALTER statement directs the simulator to rerun a similar simulation while varying one or more parameters of the design. Reading in and parsing the input file could be a task consuming operation depending on the size and hierarchy of the design. Therefore, the time spent parsing the netlist, which is always done on one CPU, is an overhead as far as parallel computing is concerned. Only the numerical simulation itself is parallelized.

Unfortunately, the effect of a relatively small portion of the overall execution time that remains serial can be devastating to the global speed improvement. As shown in Figure 1-1, even if only 10% of the 1 CPU runtime remains sequential in the parallel run, using 8 CPUs will only yield a maximum speedup of 4.7. This is assuming the optimal case of the parallel execution being at 100% performance, which is seldom the case in reality.

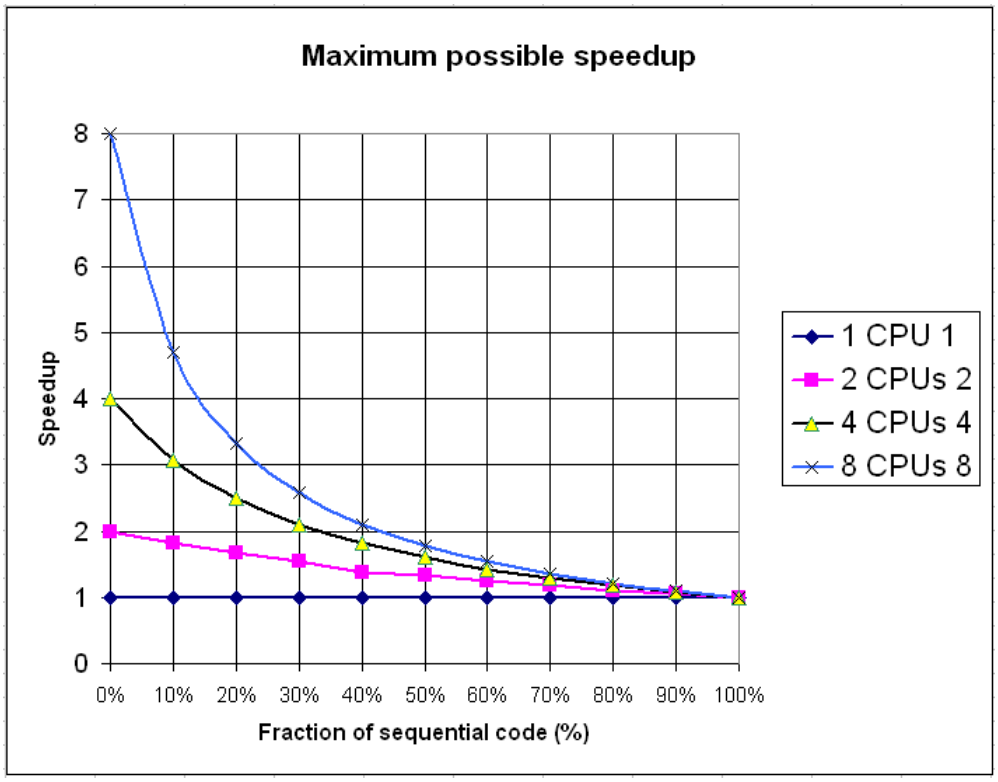


Figure 1-1 Speedup limitation due to sequential section of code

Most SPICE simulations require a steady state evaluation in DC mode followed by a transient analysis over a certain period of time. The time stepping approach requires the knowledge of node voltages and branch currents at each time step before continuing on to the next time step. So by nature, the calculations are serial between time steps, and no parallelism can be invoked at this level. This is why SmartSpice relies on data parallelism within each time step calculation, in all simulations not involving a .ALTER statement. This means that the parallelization needs to be effective at each time step calculation. Therefore, if a simulation takes a long time to execute because it performs a very large number of time steps, there is no guarantee that parallel SmartSpice will be able to reduce that execution time. It all depends on what happens within each time step, as will be explained below.

Data parallelism has a major drawback; it is only effective if the calculations on the different data subsets are independent. Otherwise, synchronization is required to ensure data integrity so that:

- No CPU is using an out of date value of a variable.
- No CPU is using a variable value that has not yet been calculated.
- No 2 CPUs are attempting to update the same variable at the same time.

In a real execution, these constraints translate into a stop-and-go execution by all the CPUs involved. CPUs stop calculating to wait for other CPUs to calculate needed values or access shared resources. In particular, parallel SmartSpice exhibits this behavior at two levels:

1. **Element model evaluation:** For any device model (transistor, resistor, capacitor, etc.), SmartSpice evaluates the model equations and submits results into a global admittance matrix. This matrix is shared among all the CPUs, and therefore a synchronization is required whenever one of the CPUs tries to update the content of this matrix. This is an overhead.
2. **Linear solver:** SmartSpice relies on an LU factorization to solve the linear system of equations within each time step. Without going into details, a lot of stop-and-go happens at this stage.

Usually, the element model evaluation (also known as Load) parallelizes fairly well. As a general rule, the parallel efficiency is better when:

- The circuit has a lot of devices (thousands at least).
- There are more active devices (transistors). It doesn't help having too many RC’s.
- The device models used are more complicated.

It is not unusual that the efficiency of the model calculation reaches 90% or more of CPU utilization on a multiprocessor computer.

On the other hand, the efficiency of the linear solver is not very well characterized, although a few comments can be made. In general, the parallel solver's efficiency deteriorates when:

- The circuit is small.
- The circuit's topology is almost a linear array of devices.
- A lot of extracted parasitics with small values are present.

In SmartSpice, these two phases of the calculation can be monitored using the `.OPTIONS ACCT=2` statement. As shown in the example below, the `Load time` reflects the time spent evaluating the device models, and the L-U decomposition time reflects the time spent calculating the LU decomposition in parallel.

Example

Running on 1 CPU:

```
smartspice -b <input-file>

equations (Circuit Equations) = 2540
loadtime (Load time) = 186.83
lutime (L-U decomposition time) = 310.83
```

Running on 2 CPUs:

```
Smartspice -b -P 2 <input-file>

equations (Circuit Equations) = 2540
```

```
loadtime (Load time) = 98.29
lutime (L-U decomposition time) = 212.83
```

If the deck contains one or more .ALTER statements and you specified -P n (and not -nodist) in the command line, where n is the number of CPUs, SmartSpice's internal functions will read the deck, extract part of the netlist which form entire circuits, and write out each circuit in the same directory where the composite netlist is situated. It also files the names of the .ALTER statements, which are contained in the table pointed to by names built up from the names of the composite basic netlist with the suffixes -n (n is the number of the CPU which is taken by extra process) at the right side. The name of the file does not contain an extension.

After running SmartSpice, you will create an extra child process. Each child process will handle a separate circuit, which is situated in a separate file without an extension. The parent process waits for the children to finish. If we specified –P 4, then 4 extra processes will be created by the parent process of SmartSpice. The total amount of SmartSpice processes in memory is 5, including 4 child processes and 1 parent. After finishing the simulation, the child will remove its own circuit simulation file for you. The .out and .raw files will have the suffix –n, which says what part of the basic netlist is described. Four .raw and .out files are created, one for each subnetlist from the composite altered netlist. The above mechanism is a powerful tool for increasing the speed of the simulation, and obeys the following rules.

We want to spawn additional SmartSpice processes, subject to the following constraints:

1. The number of subprocesses spawned should at no time exceed the number of specified CPUs on the command line, with option -P. In addition, this should not exceed the actual number of CPUs on the machine. Therefore, on a single-CPU machine, no subprocesses will be spawned.
2. If there are more .ALTERs than CPUs, the .ALTERs should be queued until CPUs are freed up.
3. One (full) license should be used for the first SmartSpice, and one multi-threaded license for each subprocess.

Parallel alter will be canceled if you specify the option –nodist in the command line of SmartSpice.

1.13.3 Solver Multithreading

Starting from 3.17.3 (3.16.5), by default, SmartSpice will request the maximum available CPU's for solver multithreading. If it's necessary to limit to N the number of requesting processors for solver multithreading, use "-PS N" command line option. The default XMS solver internally could reset this number to 1 if the matrix size is smaller than 20000 or value set from the option mtsolverthreshold. Otherwise it will use for multithreading all processors requested from SmartSpice. It is planned to use the same matrix size limit for one CPU operation for PAM solvers.

Examples

To run SmartSpice on 4 threads, and the solver on 2 threads:

```
smartspice -P 4 -PS 2
```

To run SmartSpice on the maximum number of CPU's available and the solver with 1 thread:

```
smartspice -PS 1
```

By default, SmartSpice requests the maximum CPU for the solver.

1.14 Operating Environment

SmartSpice can be used with languages other than english, and the 26 character set support has been expanded through UNICODE functionality for other languages/character sets such as japanese and chinese, etc. The following section shows how far these languages/character sets can be used in SmartSpice.

Application	Unix	Windows

Input deck, passed to SmartSpice at command line, absolute or relative, containing UNICODE characters,e.g.:
`>smartspice /home/user/XXXX/file.in or`
`>smartspice ./XXXX/file.in, where XXXX`
`some UNICODE characters`
	OK	OK

Input deck, passed to SmartSpice at command line, relative, not containing the UNICODE characters, but which may expand to absolute path with UNICODE, e.g.:
`>cd /home/user/XXXX`
`>smartspice file.in`
	OK	OK

Inside input deck (`.INCLUDE`, `.LIB`, etc.), included or library files, saved in UTF-8 format without signature	OK	OK
Input decks, containing UNICODE, opened and sourced via GUI, using program's menus	OK	OK
UNICODE character representation in GUI elements, e.g.: Output, Run-Time screen, Tasks List, Desks List, etc.	OK	OK
Command, entered via Command History edit box in GUI, containing UNICODE characters, or working with file system (such as 'cd')	OK	OK
Input decks with Verilog-A models, located on paths with UNICODE, e. g.: `/home/user/XXXX/model.va`	OK	Failed
Under GATEWAY with projects located on paths with UNICODE	OK	OK
Command mode	OK	OK

Several environment variables were added to control SFLM output behavior and multithreading information printout behavior.

SFLMPOPUP

Previously, if there was an error (e.g., “licenses not available error”), a popup was shown with options to retry or abort SmartSpice. Now, if the `SFLMPOPUP` environment variable is set and equal to 0, SFLM will not popup an error message in case of an error in batchmode. Batchmode will behave exactly as a silent batchmode; in case of an error, if it is not fatal, SFLM will silently wait until licenses become available. Default value is 1.

SFLMMTERR

On multiple CPU platforms, SmartSpice tries to use all CPUs by default. However, if insufficient extrathread licenses are available, an error message is issued and SmartSpice is reset to single thread mode. When SmartSpice is run with the `-P` option, or any of CPU

reporting variables are set, then SFLMMTERR is forced to 1. Extrathread licensing errors (if any) will be printed.

In some particular cases this error message might become annoying (e.g., if SmartSpice is used on multiple CPU platforms in single thread mode only, and extrathread licenses are always unavailable). To control this error message, the printout SFLMMTERR environment variable can be used.

If SFLMMTERR is set and equals 1, the error printout will always be shown.

If SFLMMTERR=0, the error printout will always be suppressed.

If it is unset, SmartSpice will try to determine whether extrathread licenses are present. If they are not, an error printout will be suppressed. If they are present, but are currently unavailable, an error printout will be shown. Default value is 0.

SFLMMULTICOREONLY

If SFLMMULTICOREONLY is set and equals 1, SmartSpice will not switch back to 1 CPU mode on multi-CPU machines if Extrathread/Multicore licenses are not available.

1.15 An Overview of This Manual

Chapter 2 Graphical User Interface

The graphical interface to SmartSpice is described to enable users to quickly setup and run simulations (test examples are shipped with the simulator).

Chapter 3 Statements

References all SmartSpice input statements including descriptions, syntax, and examples for each statement.

Chapter 4 Expressions

Describes SmartSpice expressions syntax, functions, predefined macros, constants, operators, and gives examples of their use.

Chapter 5 Commands

Describes the interactive commands and features of the program including command descriptions, syntax, variables, expressions, functions, control commands, and examples.

Chapter 6 Outputs

Describes SmartSpice output variables and gives examples of their use. Of special note is a description of the SmartSpice `.MEASURE` statement, which measures user-specified circuit activity to reduce the volume of output data and minimize calculation time.

Chapter 7 IBIS Model Support

Basic Input/Output Buffer Information Specification (IBIS) supported in SmartSpice. This allows signal integrity testing and so forth to be done without disclosing propriety circuit information.

Chapter 8 Verilog-A Simulation Flow

This shows the chain of events from compiling the Verilog-A code to inclusion in a simulation deck in SmartSpice and subsequent simulation.

Chapter 9 Verilog-A Debugger

SmartSpice is equipped with a powerful internal debugger tool that is used to debug and verify verilog code modules prior to inclusion in a simulation run.

Chapter 10 Verilog-A Examples

A set of examples are included to show a practical implementation of a behavioral model in Verilog-A code together with some cases of mixed.

Chapter 11 Continuous Model Approach

A special formulation of the Continuos model card. This is an alternative to the binned model approach where the desired operating region is broken up into sections (bins). This binned model can suffer discontinuity problems at the bin boundaries.

Chapter 12 TSMC Modeling Interface (TMI2)

TMI is the TSMC's Model Interface for compact SPICE device modeling.

Chapter 13 Monte Carlo and Worst-Case Analysis

Explains Monte Carlo analysis, distribution functions, and output data. It includes syntax, parameter descriptions, and examples for Monte Carlo analysis.

Chapter 14 Pole-Zero Analysis

Explains pole and zero analysis including sample input decks, circuit diagrams, and output data.

Chapter 15 Transient Noise Analysis

Explains Transient Noise Analysis including sample input decks.

Chapter 16 Timing Jitter Analysis

Noise in circuits (such as oscillators, PLL, clock data recovery, etc.) cause timing uncertainty in switching transients. SmartSpice can be used to characterize this time window of uncertainty to produce a more robust design.

Chapter 17 Reliability Analysis Using Single Event Effects

Explains the causes of Single Event Effects (SEEs).

Chapter 18 Optimizer

Explains the Optimizer features: function and performance, measure optimization, the `.MODIF` statement, and parametric analysis.

Chapter 19 Mismatch Analyses

Variation in the physical manufacture of devices leads to a variability in the electrical performance. This chapter shows how to simulate this variability in performance so that the circuit design can take this variation into account. This will enable the designer to get a good yield to chip specification.

Chapter 20 Variation Block

This describes a block used in the input deck or associated files to do Model and parameter variations. The circuit can therefore be simulated for process variations that will change the yield of devices to a specification.

Chapter 21 Using Solvers in SmartSpice

SmartSpice parsers the input deck and forms a matrix of elements representing the circuit to be simulated. The solver works on this matrix to provide a number of solutions during an Analysis run. The speed of finding a solution will depend on the accuracy and the solver/ settings used. This chapter describes the different solvers and their associated `.OPTIONS` settings that can be used.

Chapter 22 Rubberband

The SmartSpice Rubberband feature allows you to select and modify model and instance parameters, and interactively watch in SMARTVIEW how the simulation changes in real time.

Chapter 23 Expression Accelerator (EAC) library

Simulation times for input decks containing large expressions can be accelerated using this EAC library.

Descriptions of when and how to use this construct in SmartSpice are given.

Chapter 24 SmartSpice Distributive Capabilities for Monte Carlo and Alter

Simulation input decks containing multiple variations of Monte-Carlo, sweep, Alters etc. have the possibility to be run across multiple cpu's on a single machine and across multiple designated machines to maximise performance and run time.

Chapter 25 Job Queuing Systems

Explains how to queue multiple SmartSpice jobs using a Grid Engine.

Chapter 26 Installing and Using SmartSpice in the Analog Artist Environment

To make SmartSpice easily accessible to schematic designers, a SmartSpice interface has been created for the Cadence Design Framework™. This chapter describes how this interface functions, and how it is installed and configured.

Chapter 27 Speed/Performance Control Settings

This option is used to activate a faster simulation mode of SmartSpice.

Chapter 28 Convergence Problems - Troubleshooting and Solving

Guide to the flow of achieving convergence in the circuit simulator, and recommendations where convergence is an issue.

Chapter 29 Debugging Features

SmartSpice can be run in a batch mode or an interactive GUI mode. In the GUI mode features are available to show how the input deck information is simulated. This chapter helps the user understand what is wrong with the input deck information.

Chapter 30 Stimulus Editor

Before running a simulation the input stimulus sources (Vsource) can be checked for timing and modified, if required. This saves time in running a simulation and then discovering the input sources are incorrect.

Chapter 31 SmartSpice RLC-Reduction Tool

SmartSpice now contains an RC reduction tool to reduce parasitic elements to be evaluated.

This simplifies using an extra tool to do the reduction and saves resources running SmartSpice.

Chapter 32 Simulation Statistics Output

At the end of a simulation run SmartSpice outputs statistics showing a number and type of devices in the circuit. Time spent in different phases of the simulation help indicate any time wasted.

Chapter 33 Examples

Examples of circuits, input decks, and output data to illustrate basic and advanced SmartSpice features.

Chapter 34 Encryption Module

To protect proprietary information SmartSpice can use any file from an input deck in an encrypted form to Simulate. This allows collaboration on a circuit without individual contributor contents being disclosed. Encryption can be used at any level of the circuit from full input deck down to individual model files.

Chapter 35 SmartSpice Licensing

SmartSpice can be set to release the licence if it is not being used in a specified time and therefore be used more efficiently.

Appendix A Transient Simulation with Large Data Sets

Explains how SmartSpice allocates memory and documents the SMARTSPICE memory management functions.

Chapter 2
Graphical User Interface

2.1 Getting Started

This chapter explains how to “Drag-and-Drop” a file in windows onto the SmartSpice window to quickly load and run a file. This leads to a quick setup of the tool, and users can quickly start using SmartSpice to perform simulations without a long familiarization period to run simple cases.

Simply drag a deck from the Explorer Window and drop it over the SmartSpice Window. SmartSpice will source the dropped deck, and then you can run the analysis to get results.

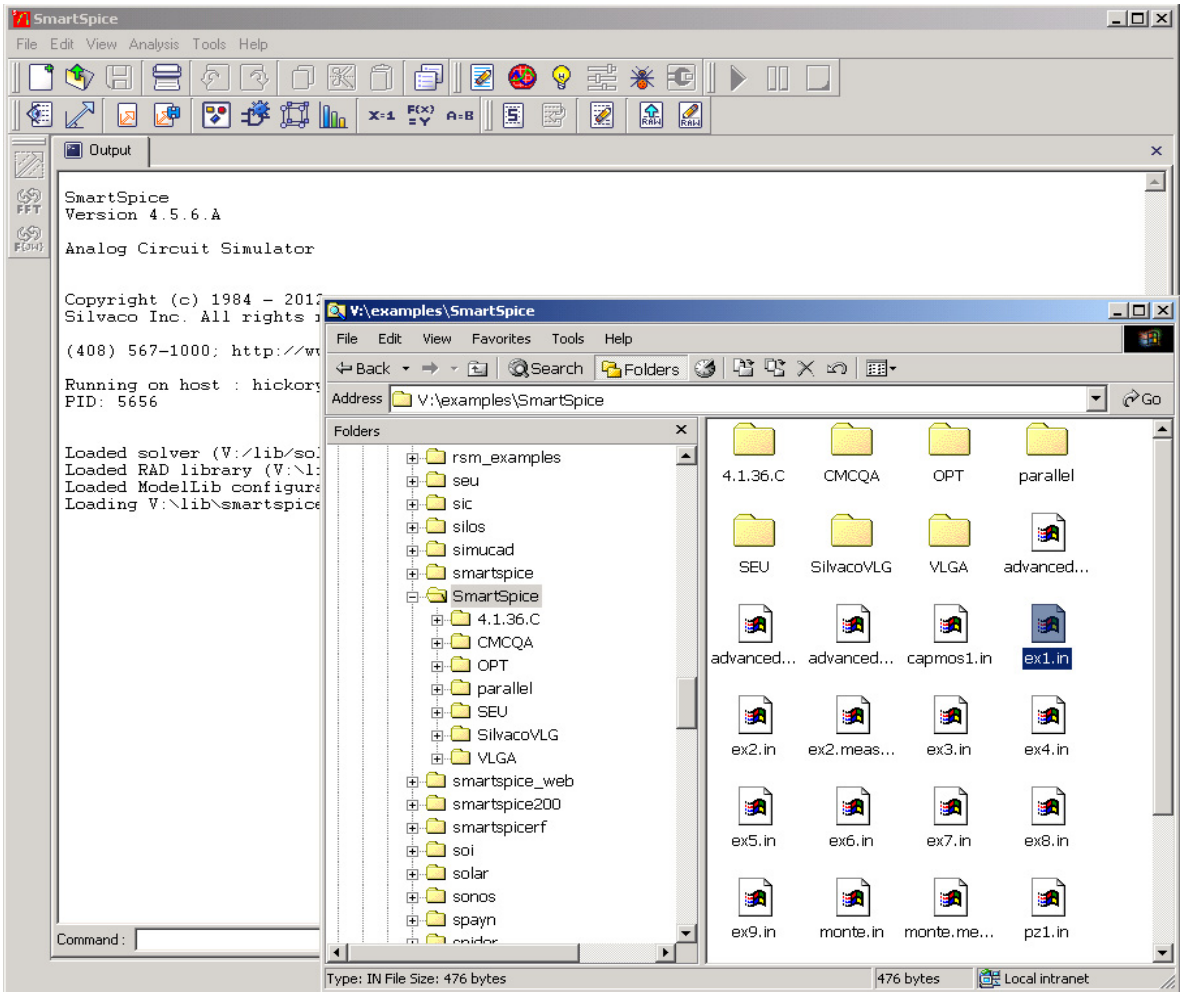


Figure 2-1 Drag and Drop Functionality

2.1.1 Tabbed Documents

In this mode, document windows are tabbed together within the editor. Tabbed document windows are useful for organizing and switching between multiple open documents. This mode is accessible through **View→Tabbed Windows** menu item with a checked state.

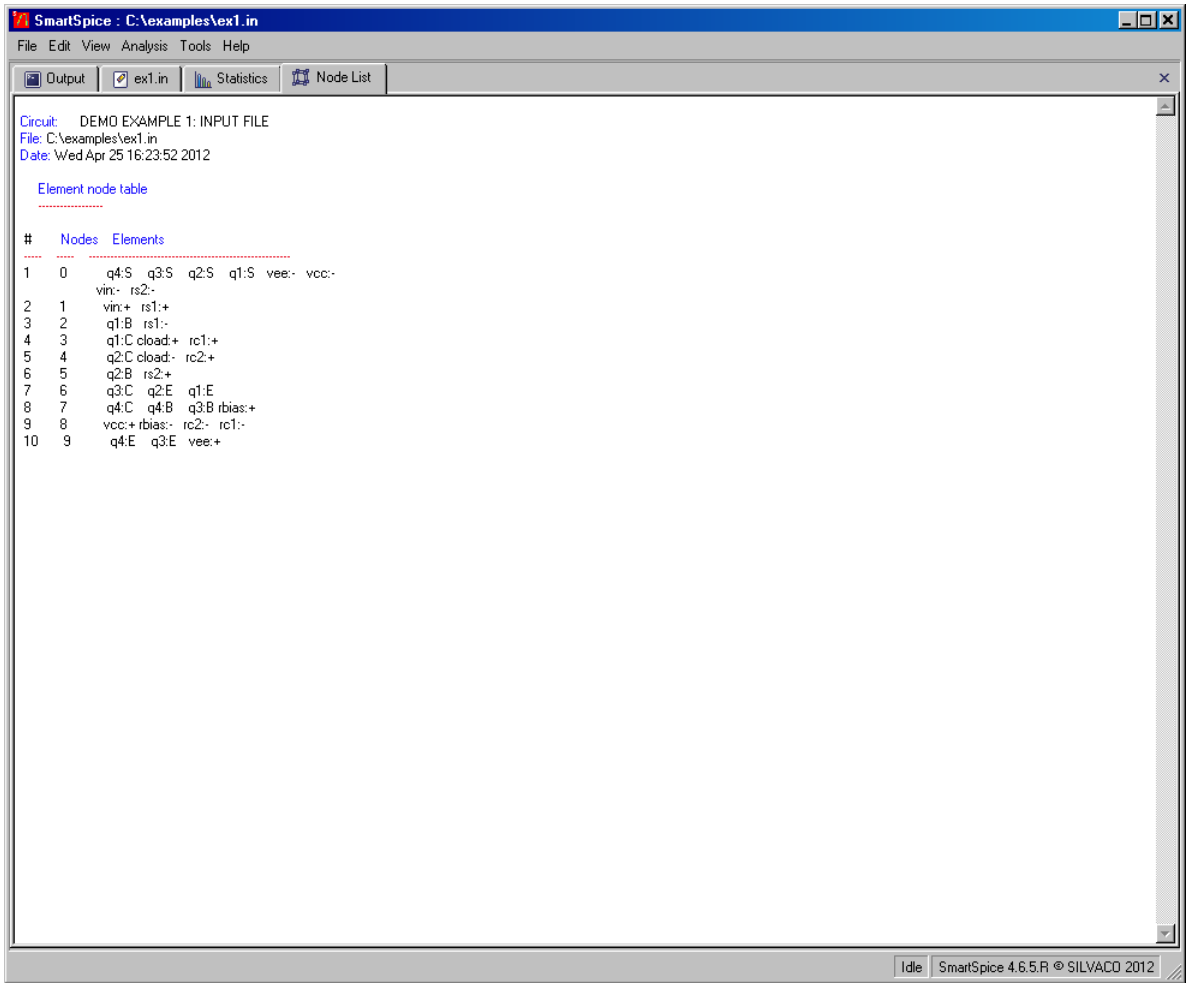


Figure 2-2 Tabbed Documents

2.1.2 MDI Environment

In this mode, documents are opened in a Multiple Document Interface (MDI) environment. MDI document windows are useful for gaining the screen space that is otherwise taken up by the tabs in the tabbed documents environment. When working in MDI mode, an additional Window menu with the following tile options appears. They are:

- **Cascade:** Arrange windows so they overlap.
- **Tile:** Arrange windows in non-overlapping pattern.
- **Tile Horizontally:** Arrange windows in horizontal non-overlapping pattern.

You can switch between documents by pressing **Ctrl + Tab**, or you can use the list of opened windows located on the bottom of the Window menu.

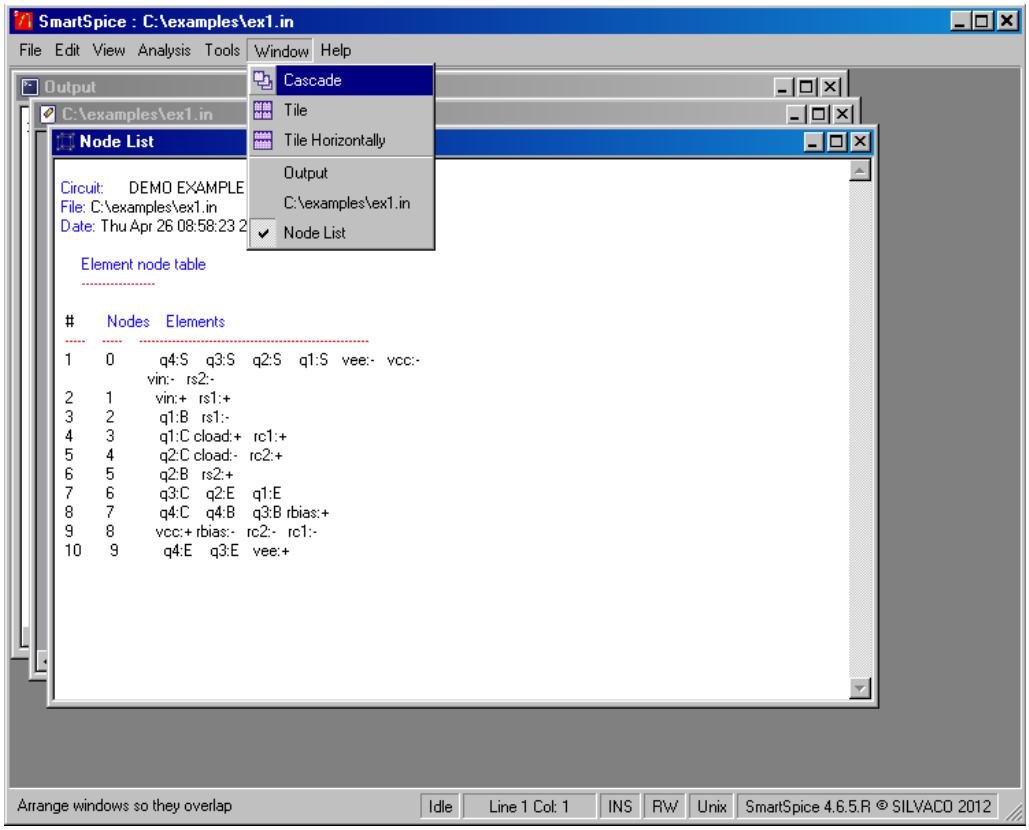


Figure 2-3 MDI Environment

2.1.3 Assistant Bar

This is an additional window that may hold the tabs with miscellaneous views. It can be turned off either by pressing the **x** button in the top corner, or by closing each tab with the **x** button in the bottom corner. Later, the tabs may be reopened through the **View** menus items.

If not hidden, Assistant Bar can be found in two states:

- Docking state (it docks to any border side of main window)

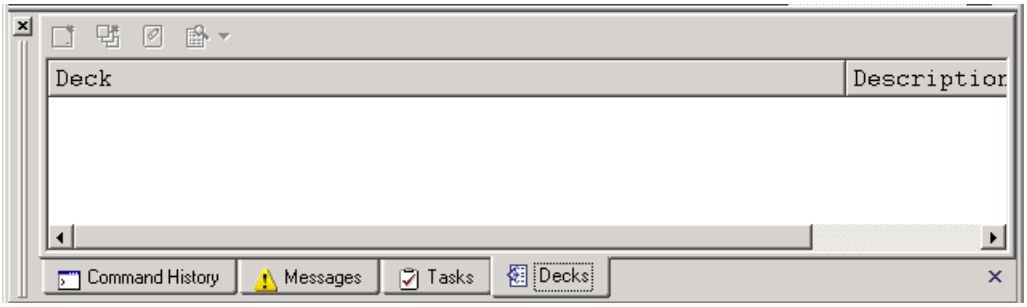


Figure 2-4 Assistant Bar - Docking State

- Floating state (floats over main window, not docking)

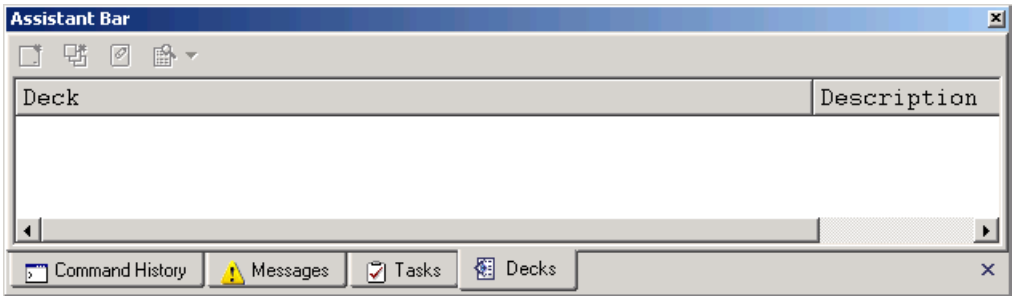


Figure 2-5 Assistant Bar - Floating State

Transition from one state to another is performed with the mouse, by grabbing a gripper in a **Docked** state (the bar on a left of a window, see Figure 2-4) or a caption (the window's top) in a **Floating** state and then dragging it around main window's border or to the desired location, or by double clicking on gripper/caption.

2.1.4 Integrated Editor

The standalone editor used in previous versions was fully integrated into the SmartSpice main window. There are no essential differences in functionality. For a description of the details of the editor's GUI elements, please refer to the SEDIT manual.

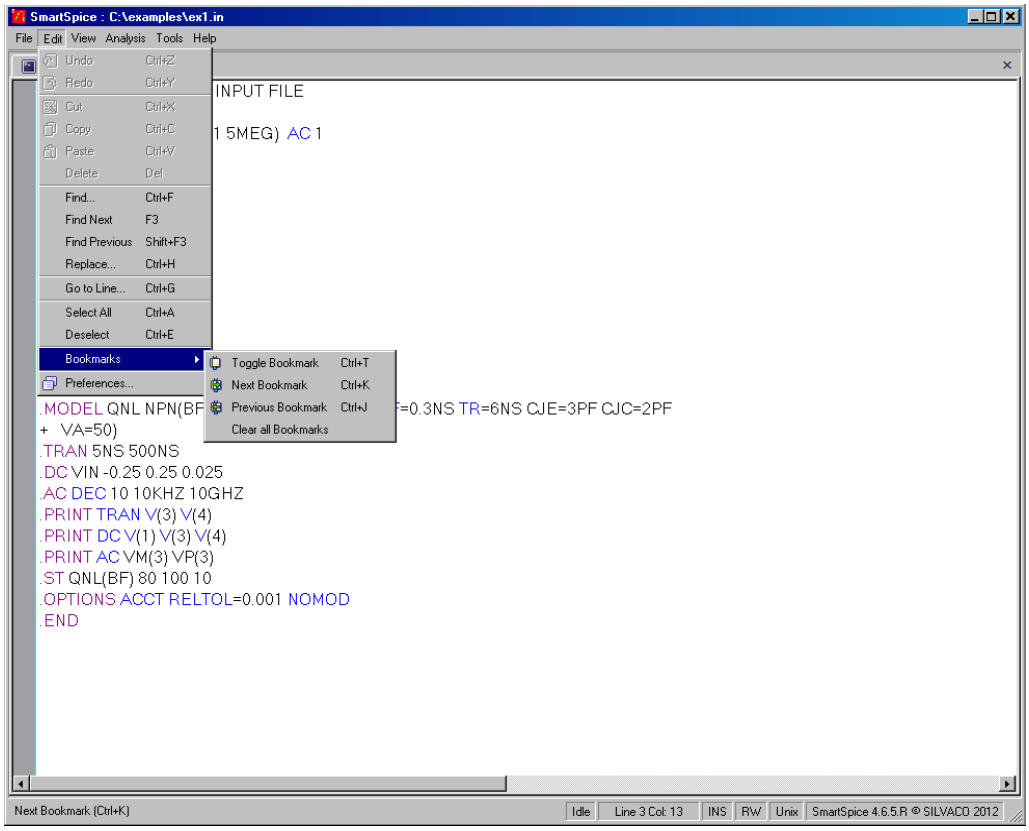


Figure 2-6 Integrated Editor

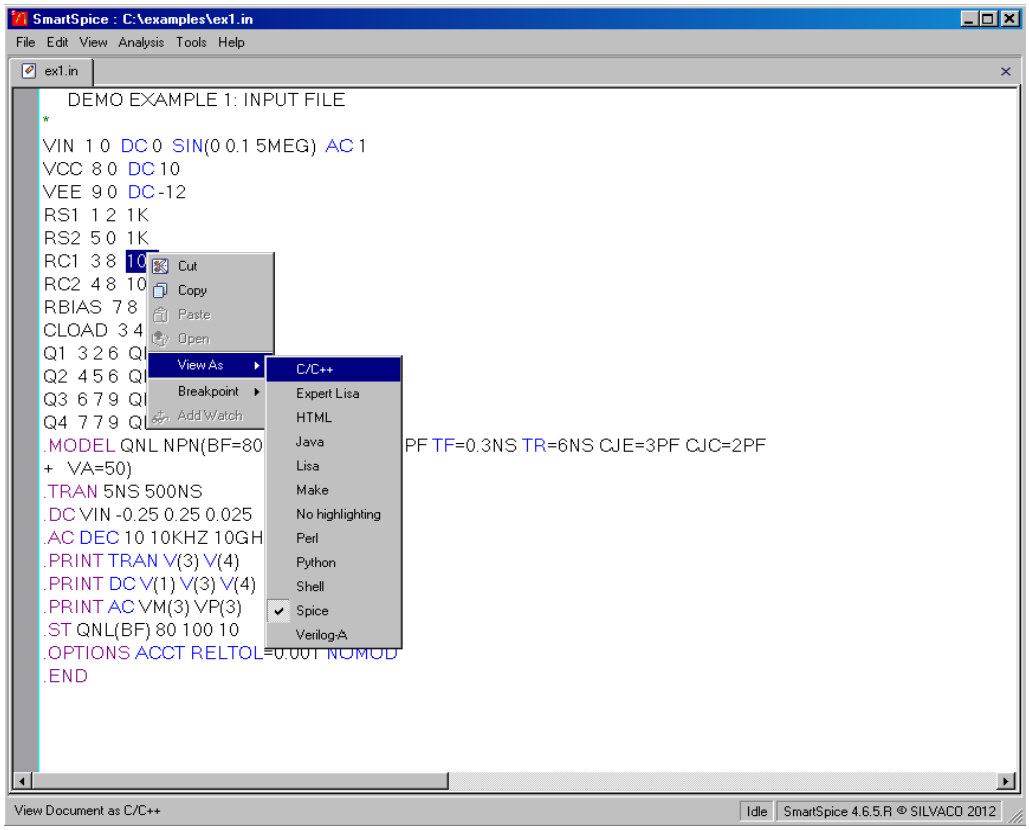


Figure 2-7 Integrated Editor - Miscellaneous Syntax Highlighting Support

The editor-related pages in the **Preferences** dialog (see the SEDIT manual).

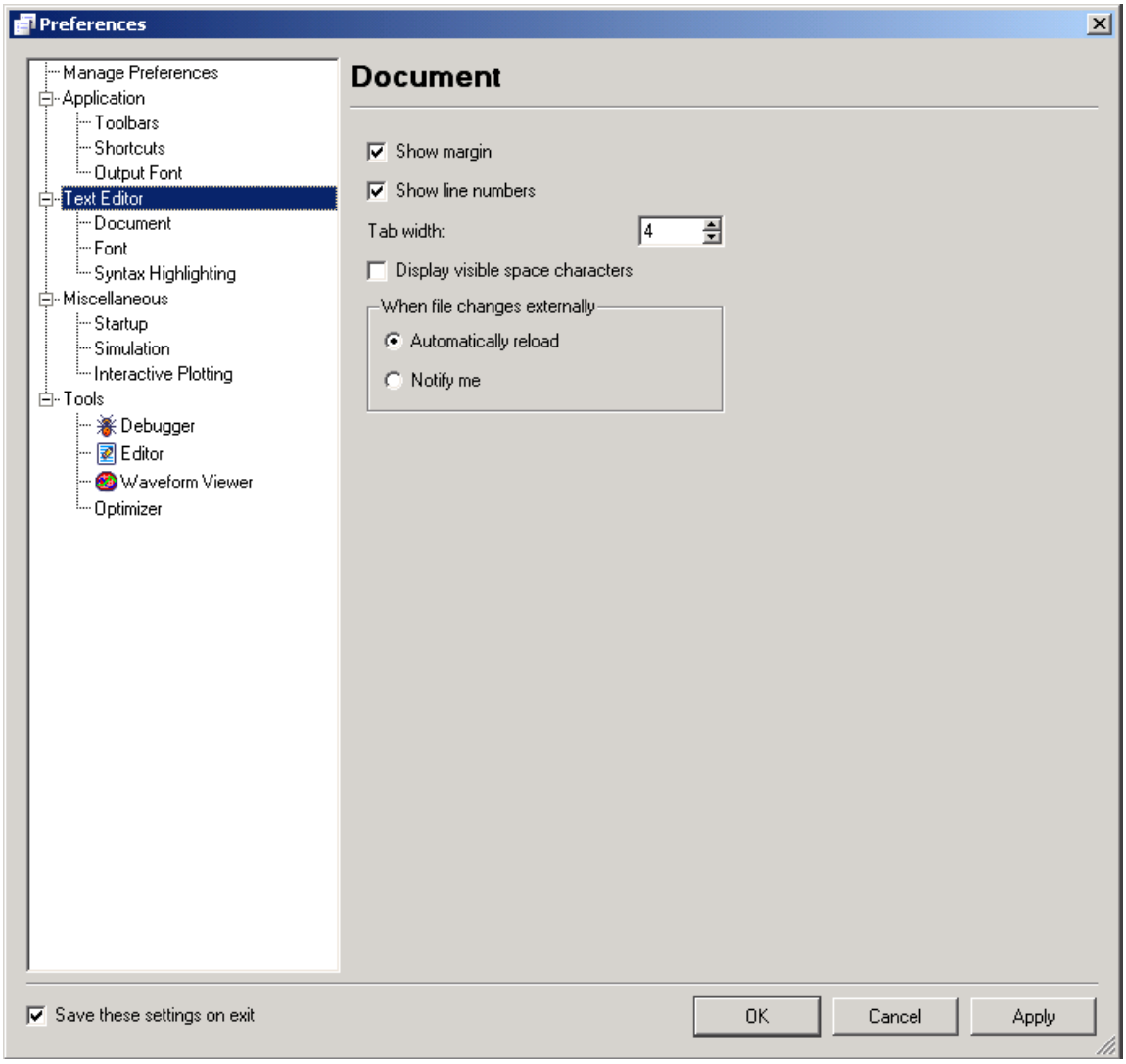


Figure 2-8 Integrated Editor - Document Settings Preferences Page

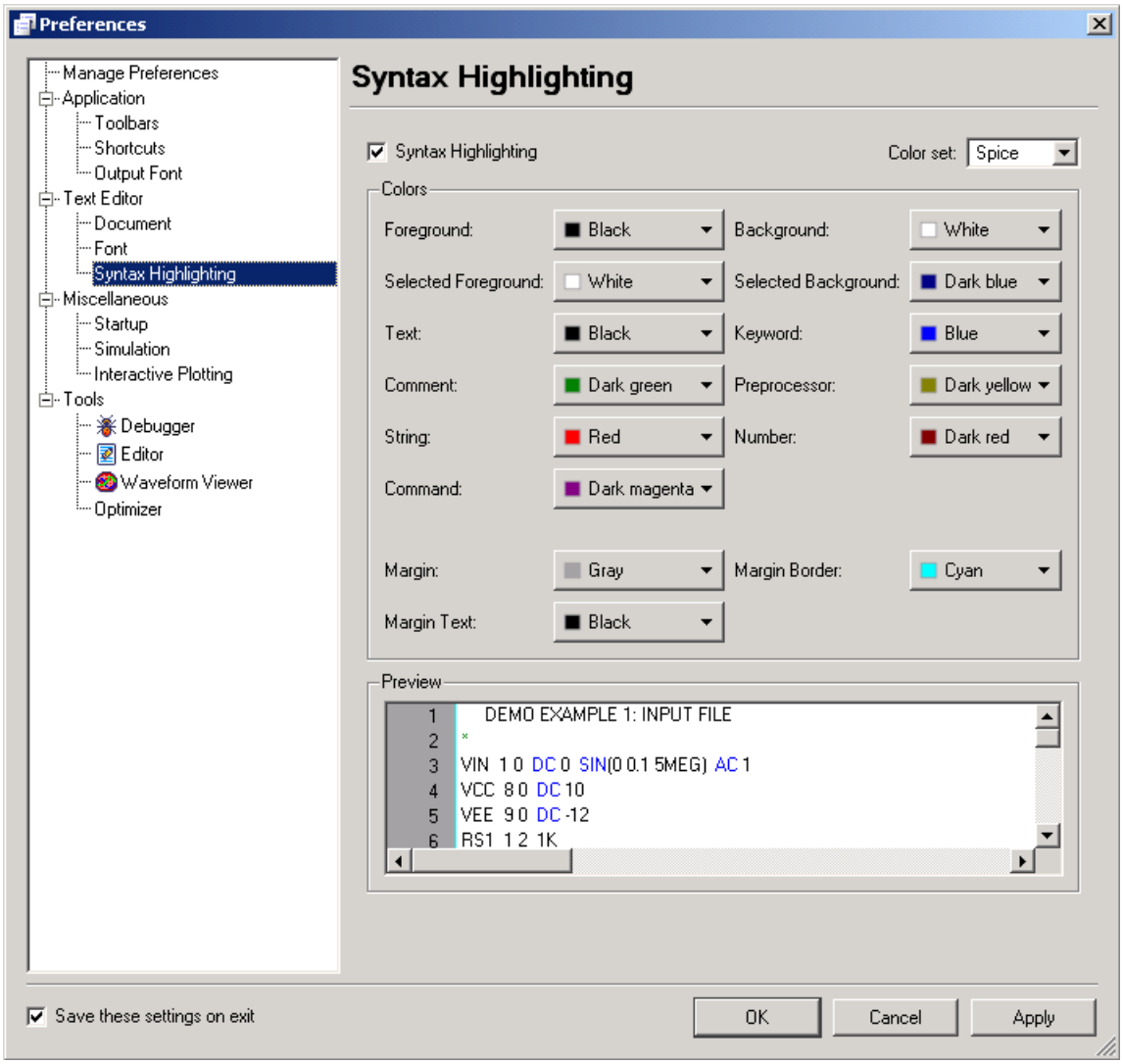


Figure 2-9 Integrated Editor - Syntax Highlighting Preferences Page

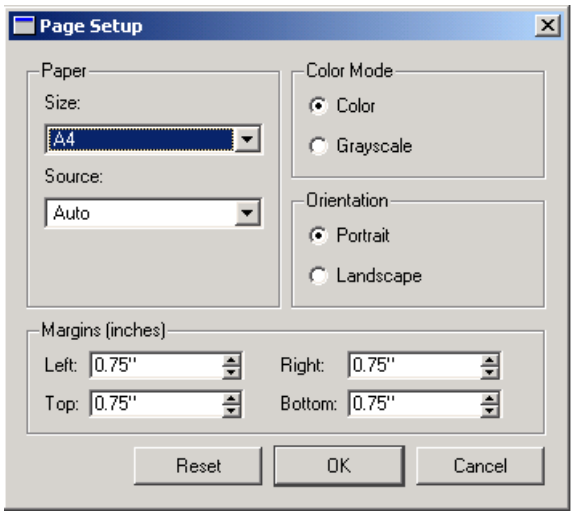


Figure 2-10 Page Setup Dialog (see SEDIT Manual)

Figure 2-11 Edit Toolbar (see SEDIT manual)

2.2 Interactive GUI Functions

This section aims to describe the operation and functionality of the graphical user interface (GUI) to the SmartSpice simulation engine. The GUI feel and operation is common across all supported platforms so that all the descriptions contained in this document are applicable across every supported operating system.

For more information on SmartSpice commands/SPICE statements, please refer to the additional SmartSpice documentation (see Section 2.26 Help).

2.3 SmartSpice GUI

The SmartSpice GUI can be launched in interactive mode using the following operating system command:

```
smartspice -V <version>
```

Where <version> denotes the version of SmartSpice to start (e.g., 2.8.0.R). Alternatively on some operating systems (e.g., Windows), the application can be started using the application shortcut created during installation.

Once SmartSpice has finished initializing, the **SmartSpice GUI** main window will be displayed:

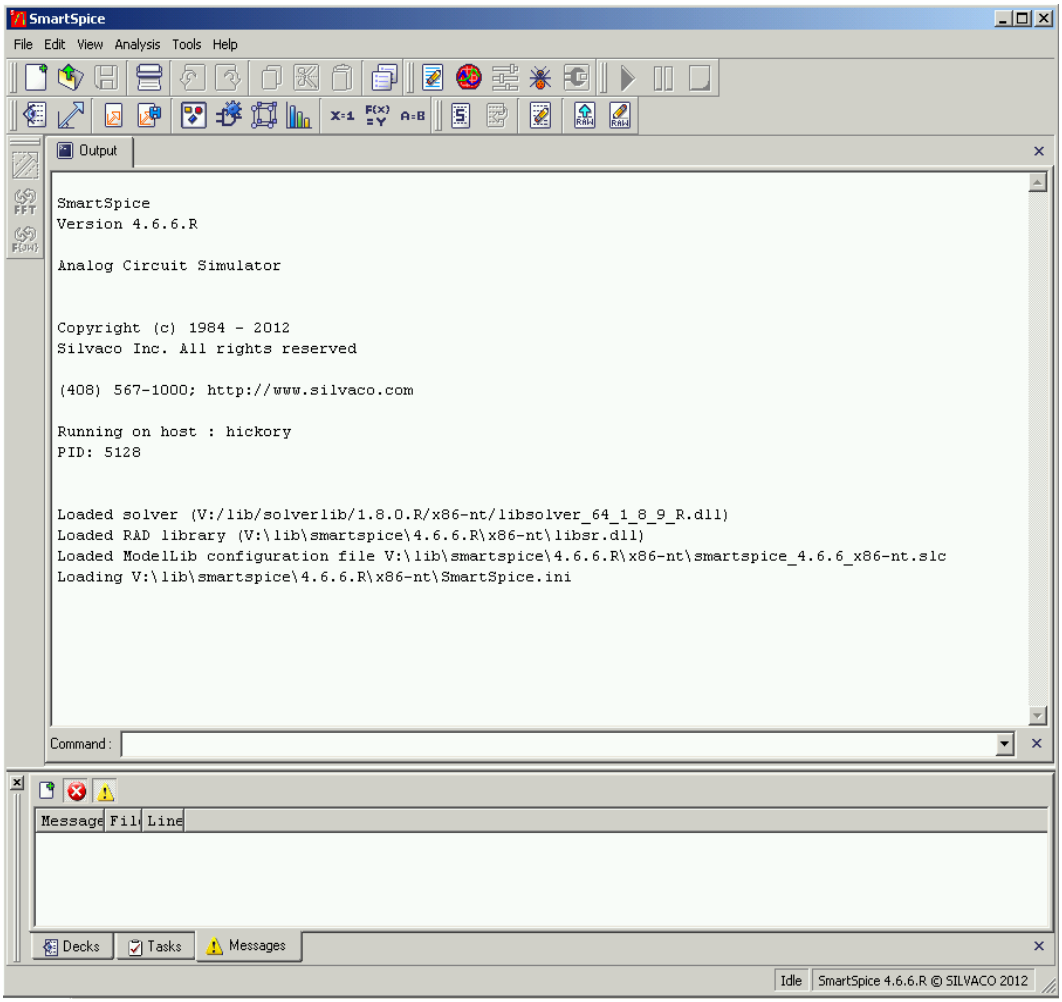
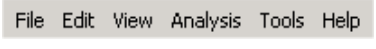


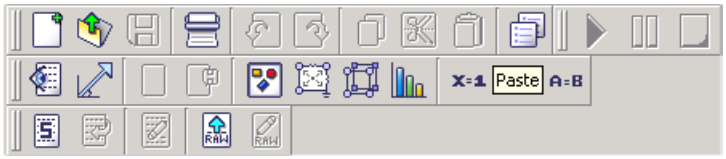
Figure 2-12 SmartSpice GUI Main Window

The main window comprises several sections:

- **Menu:** Provides access to all SmartSpice functionality. Enabled functionality may vary according to the simulation engine state. As each menu item is selected, a description of the menu action will be displayed in the status bar.



- **Toolbar:** The toolbar action icons are shortcuts to operations contained within the menus described above. A tool-tip phrase will be displayed when the mouse is hovered over each toolbar icon to explain its purpose. It is possible for you to customize each toolbar by adding or deleting icons in the toolbar (see Section 2.23.3 Toolbars Panel for details on how to customize each toolbar).



- **Output:** Displays comments generated throughout the application's use. The output can be saved or printed by selecting **Save Output→Print** from the **File** menu.

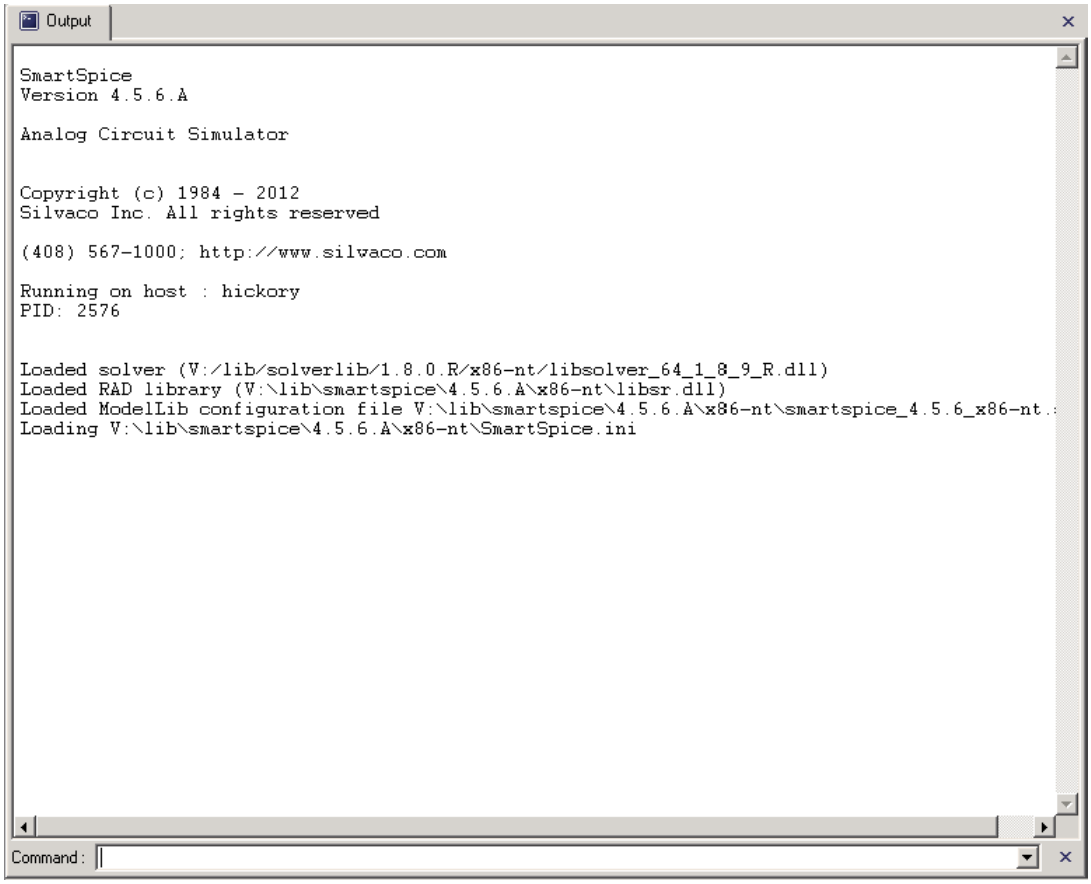


Figure 2-13 Output Window

- **Command Line:** Used to directly enter SmartSpice commands using the keyboard:

Figure 2-14 Output Window - Command Line

The Command line is combined with the list of available SmartSpice commands at the given moment. You can select a command from the list to bring it to the input box to run it or modify. If you don't need the input control, you can close it by clicking the **x** button at the bottom right corner of the **Input/Output** window. Later you can re-open it from the context menu **Toggle Command Line** by right-clicking on window's interior:

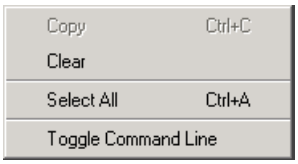


Figure 2-15 Output Window - Context Menu

Other items available form the context menu:

- **Copy:** copy the selection to the clipboard
- **Clear:** Clear the window
- **Select All:** Select entire contents

SmartSpice can layout child windows in two ways:

- **Tabbed Documents:** In this mode, document windows are tabbed together within editors. Tabbed document windows are useful for organizing and switching between multiple open documents. This mode is accessible through **View→MDI environment** menu item with the unchecked state.

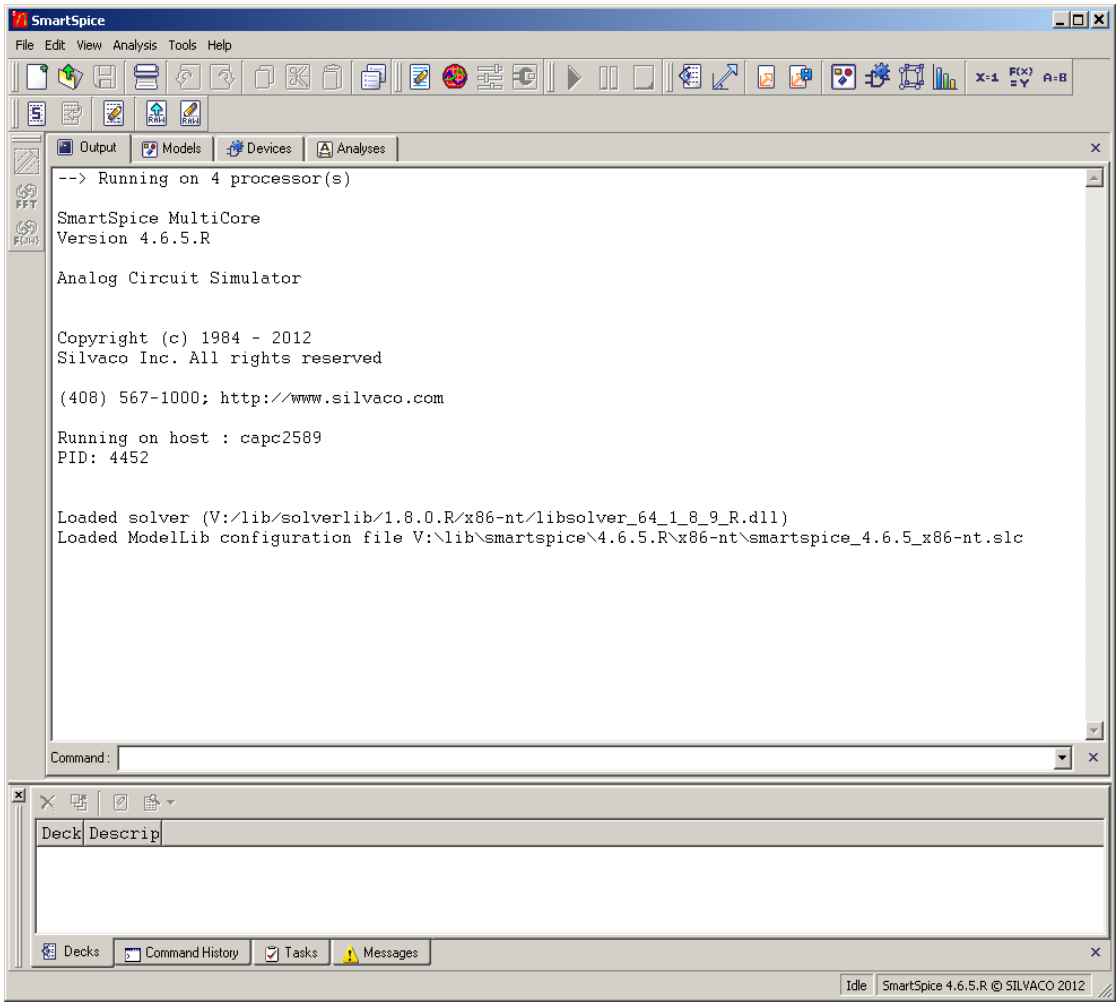


Figure 2-16 Tabbed Documents

- **MDI Environment:** In this mode documents are being opened in a multiple document interface (MDI) environment. MDI document windows are useful for gaining screen space that is otherwise taken up by the tabs in the tabbed document's environment. When working in MDI mode, an additional **Window** menu with the tile options appears. They are:

- **Cascade:** Arrange windows so they overlap.
- **Tile:** Arrange windows in non-overlapping pattern.
- **Tile Horizontally:** Arrange windows in horizontal non-overlapping pattern.

You can switch between documents by pressing **Ctrl+Tab**, or you can use the list of opened windows located on the bottom of the **Window** menu.

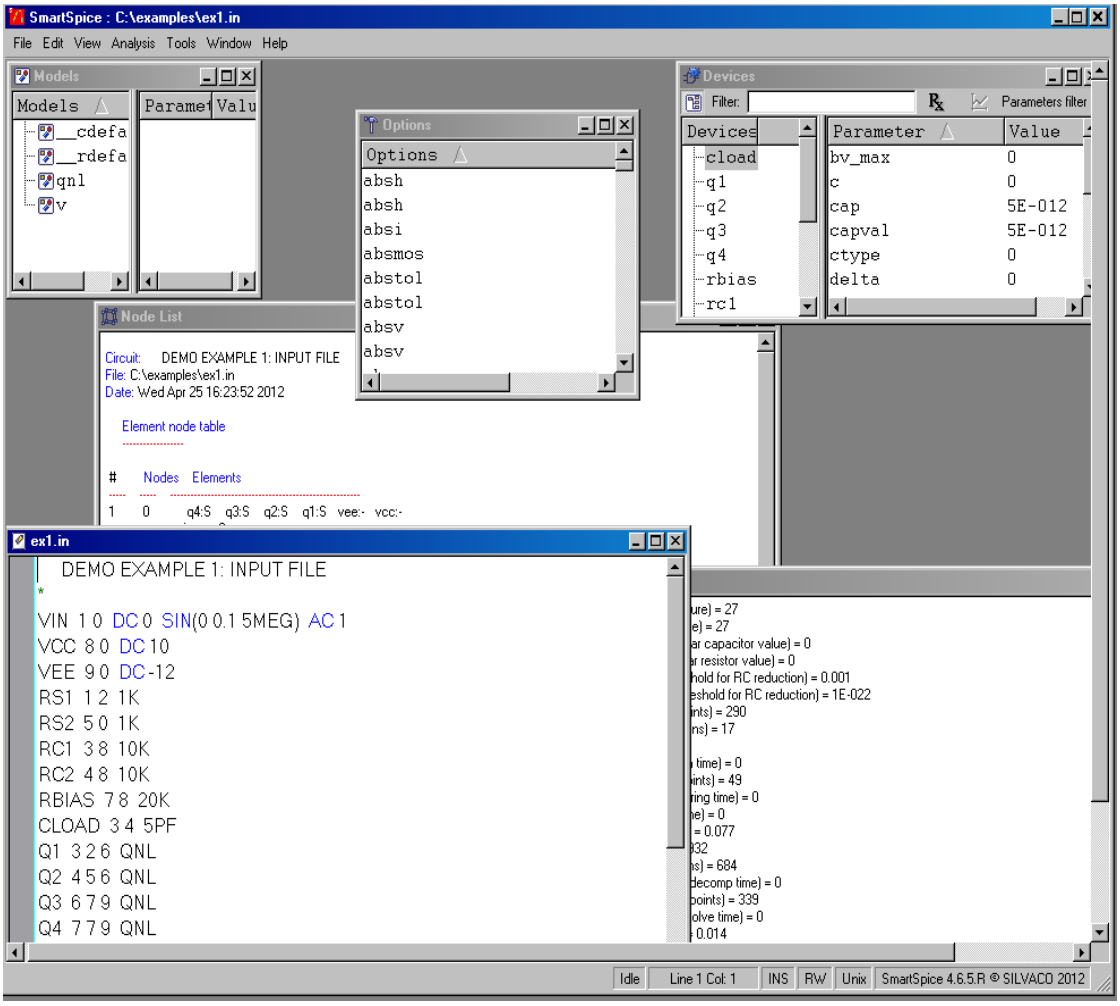


Figure 2-17 MDI Environment

Windows can be opened in a floating state. Just right click on a header and select **Dockable** from the popup menu:

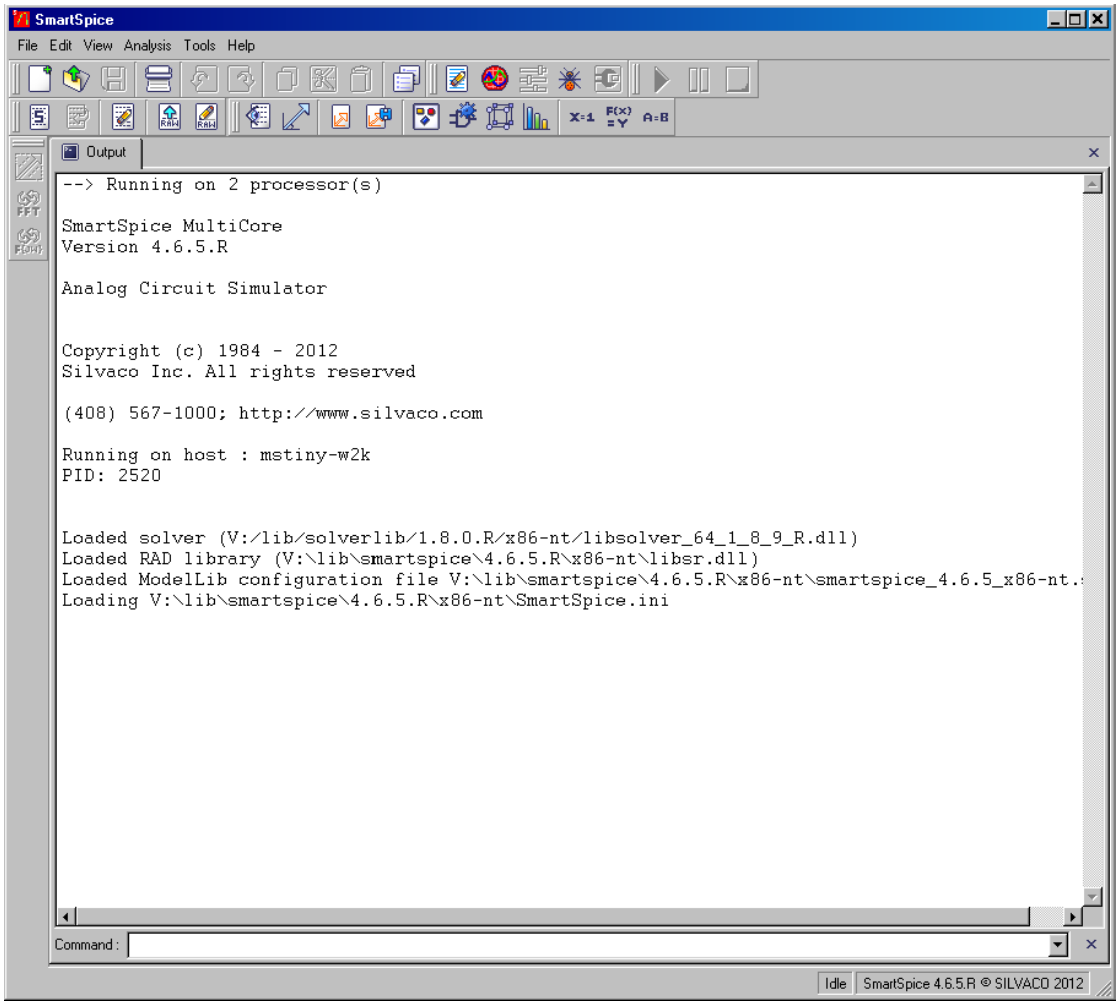


Figure 2-18 Dockable

- **Assistant Bar:** This is an additional window that can hold tabs with miscellaneous views. It can be turned off either by pressing the **x** button in the top corner, or by closing each tab with the **x** button in the bottom corner. Later, the tabs may be reopened through **View** menu items.

  If not hidden, **Assistant Bar** can be found in two states:

  - **Docking State** (docks to any border side of main window)

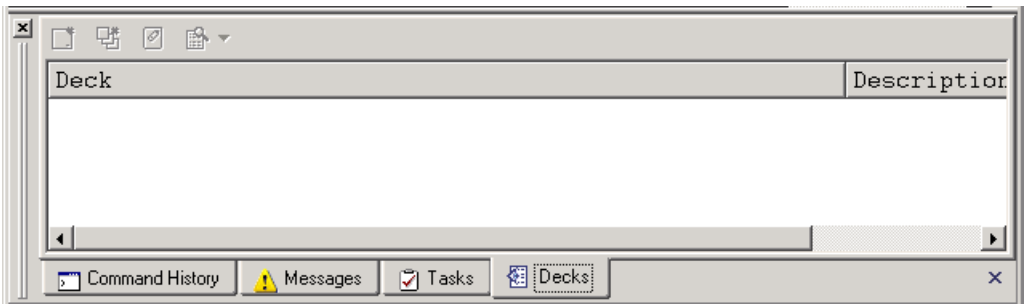


Figure 2-19 Assistant Bar in Docking State

- **Floating State** (floats over the main window, not docking)

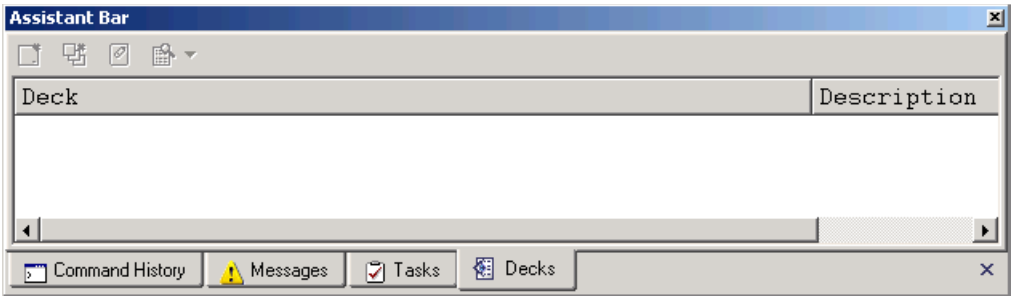


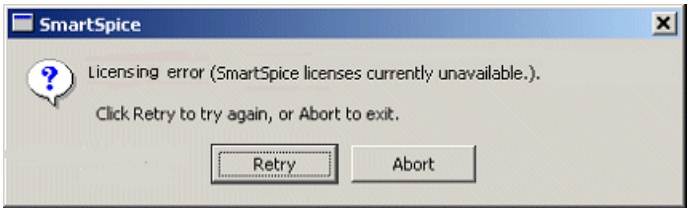
Figure 2-20 Assistant Bar in Floating State

Transition from one state to another is performed with the mouse by holding it by the gripper/caption and dragging it around main window's border to the desired location, or by double clicking on the gripper/caption.

2.3.1 SFLM Licensing

SmartSpice is licensed using the Silvaco Flexible License Manager (SFLM).

If there are no licenses available when the application is started, a dialog box will be displayed to determine whether to reattempt to retrieve a license:



- **Retry:** Click to re-attempt to retrieve a license.
- **Abort:** Click to quit SmartSpice.

2.3.2 Navigation

The SmartSpice GUI can be navigated in several ways: keyboard, mouse or cursor feedback.

Keyboard

The SmartSpice GUI can be driven entirely using the keyboard:

- **Menu Actions:** All menu actions can be accessed by pressing the **Alt** key and selecting a keyboard accelerator for the required menu item, denoted by the underscore ('_') in the menu item name (e.g., press **Alt**, then **F** for File to display the **File** menu, followed by **S** for Source to display the **Source** dialog). Menu items can also be performed by using the shortcut accelerators (see Section 2.23.5 Shortcuts Panel for more details on how to customize these shortcuts).
- **Command Line:** SmartSpice commands can be entered using the command line control at the bottom of the main window. This control should have input focus when the main window is the current window in focus. The **Tab** key can be used to alternate the control that has the current input focus.

Mouse

When using the mouse to drive the GUI, the following describes the typical usage of the mouse buttons:

- **Left Mouse Button:** Used to select an object or an action. Unless otherwise stated, the term mouse click refers to this mouse button.
- **Right Mouse Button:** Opens a drop-down context menu. The drop-down menus will be different depending upon the control at the current mouse location.

Cursor Feedback

When a long operation is being performed, the wait cursor will be displayed to inform you to wait for the current operation to finish.

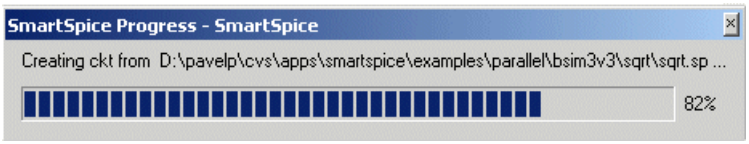
2.4 Example Usage

The following example describes how to perform the most common SmartSpice functionality:

- Sourcing an input deck
- Performing a simulation
- Plotting from analyses results using the **Vectors** dialog
- Saving the analyses results by writing a rawfile

Please refer to the previous sections for more information:

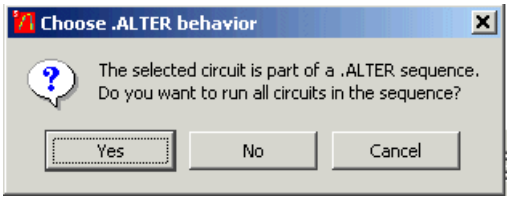
1. Select **Source** from the File menu, browse to:
   `examples\parallel\bsim3v3\sqrt\sqrt.sp`
   and click **Open**. A progress dialog will appear while the deck is sourced:



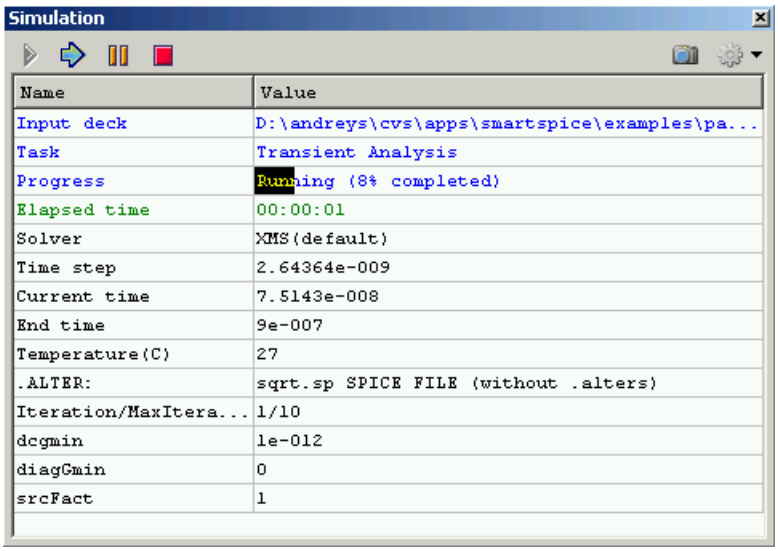
The Main window's output section will display notification when the deck has been sourced:

```
--> source 'D:\pavelp\cvs\apps\smartspice\examples\parallel\bsim3v3\sqrt\sqrt.sp'
Loading D:\pavelp\cvs\apps\smartspice\examples\parallel\bsim3v3\sqrt\sqrt.sp
```

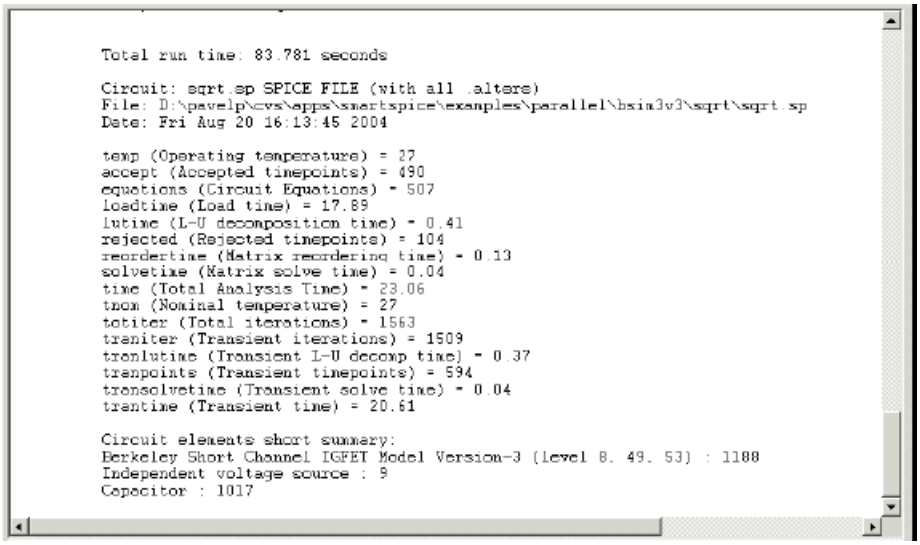
2. Select **Run** from the Analysis menu to perform the simulation. As the deck contains `.ALTER` statements, a dialog will be displayed to confirm whether to run all circuits in the sequence:



3. Click **No** and the **Run Time** dialog will be displayed for the duration of the simulation:



4. The **Run Time** dialog will close when the simulation is finished, and the Main window's output section will display a simulation summary:



5. Select **Vectors** from the **Display** menu to display the **Vectors** dialog:

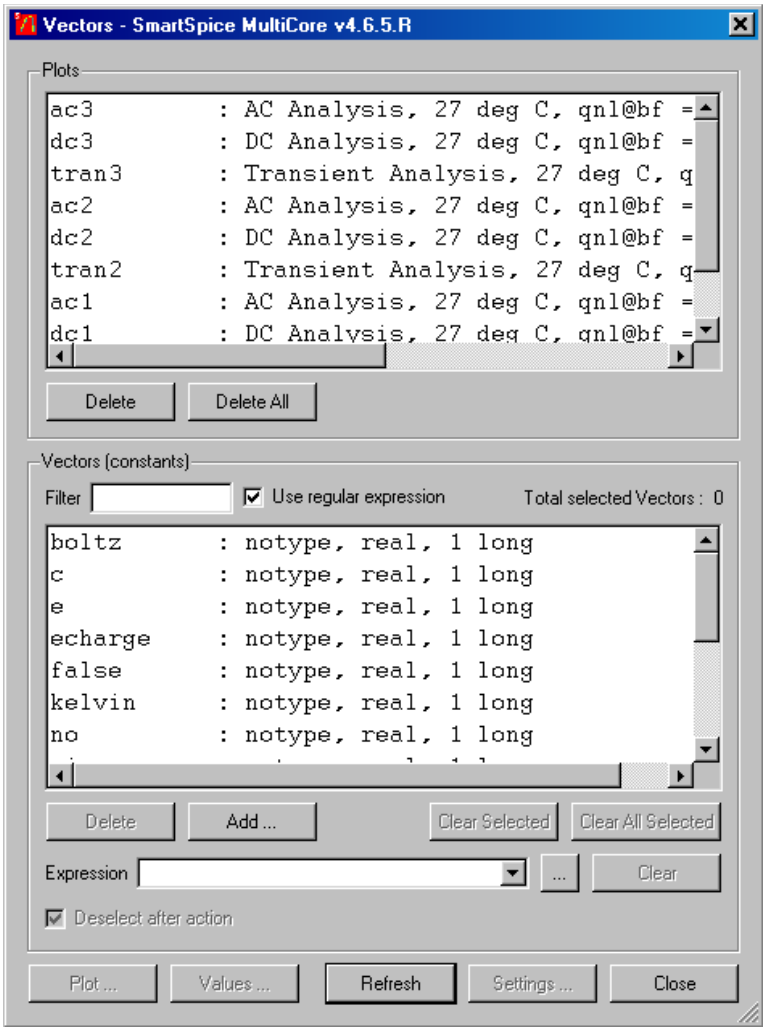


Figure 2-21 Vectors Dialog

The last performed analysis will be selected.

6. Select some vectors from the **Vectors** list:

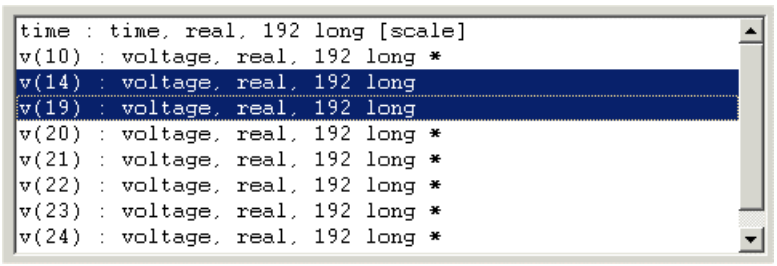


Figure 2-22 Vectors List

7. Click **Plot** to display the selected vectors in the **Waveform Viewer** using the current plot settings:

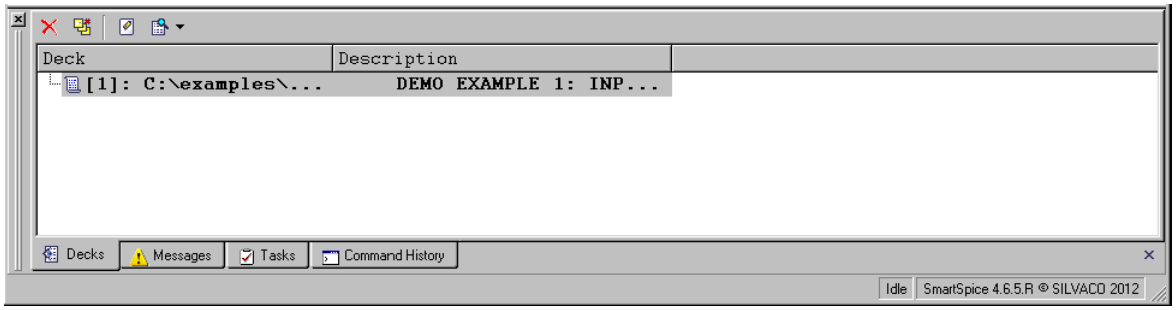


Figure 2-23 Current Plot Settings

8. Select **Run** from the File menu to re-perform the simulation. Click **Yes** to run all circuits in the sequence, and wait for the simulation to be completed.

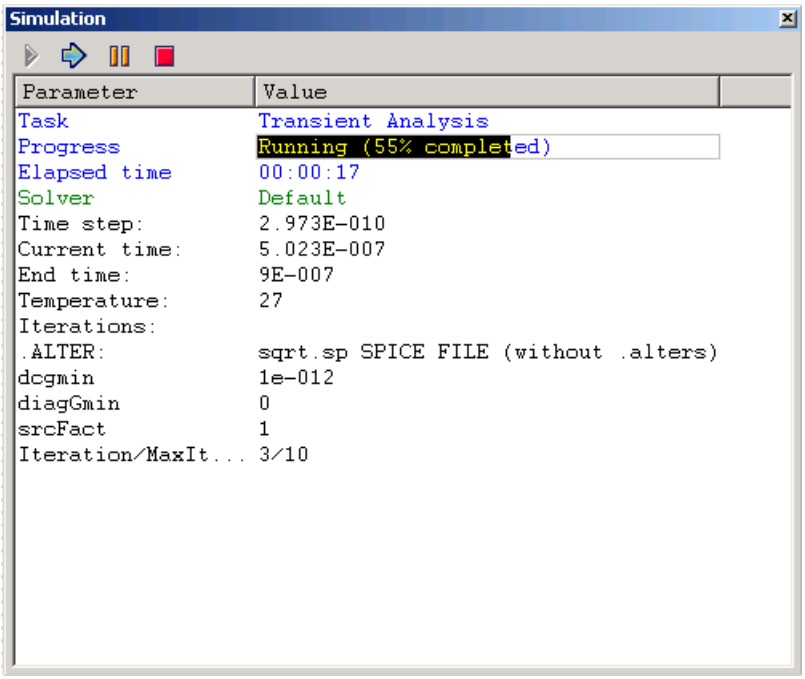


Figure 2-24 Simulation Progress Window

9. Select the most recent analysis and the same vectors:

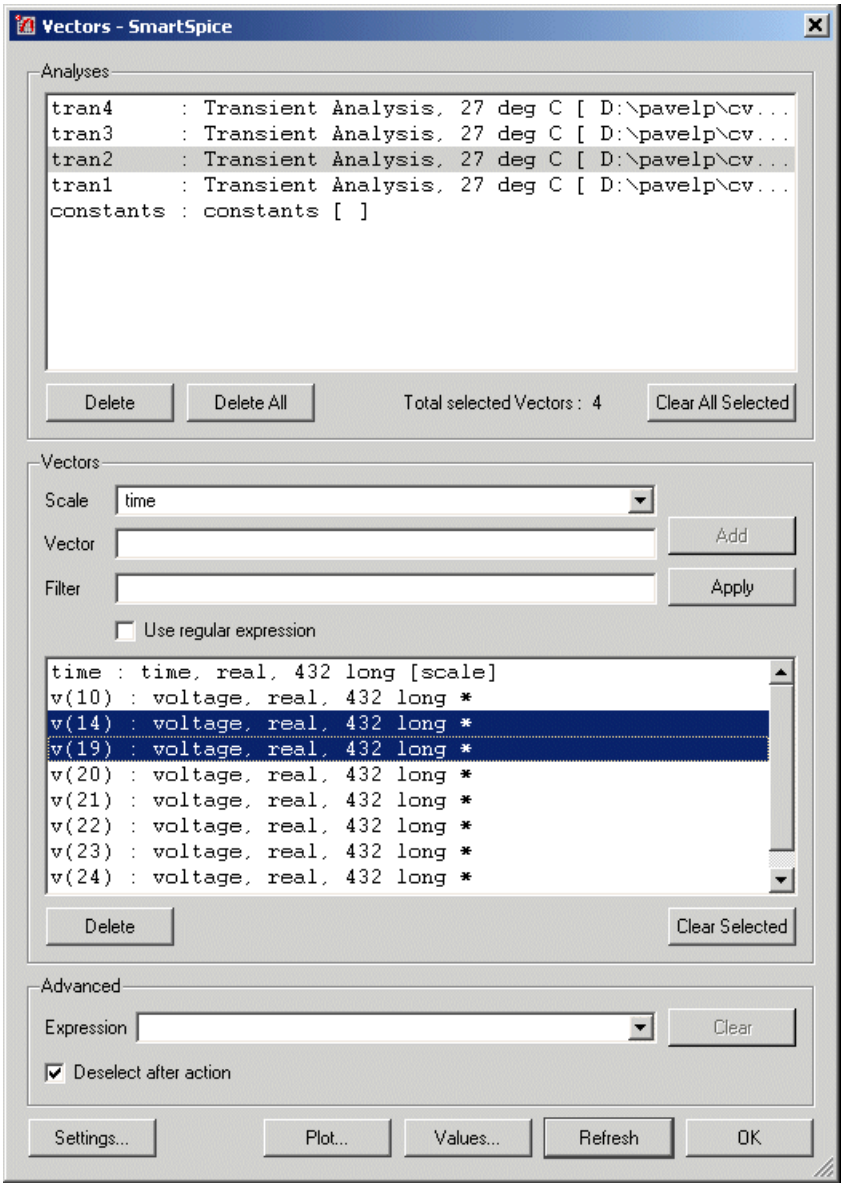


Figure 2-25 Analysis and Associated Vector Display

10. Click **Settings** to open the **Plot Settings** dialog, then select **Use selected plot/chart and Replace traces**:

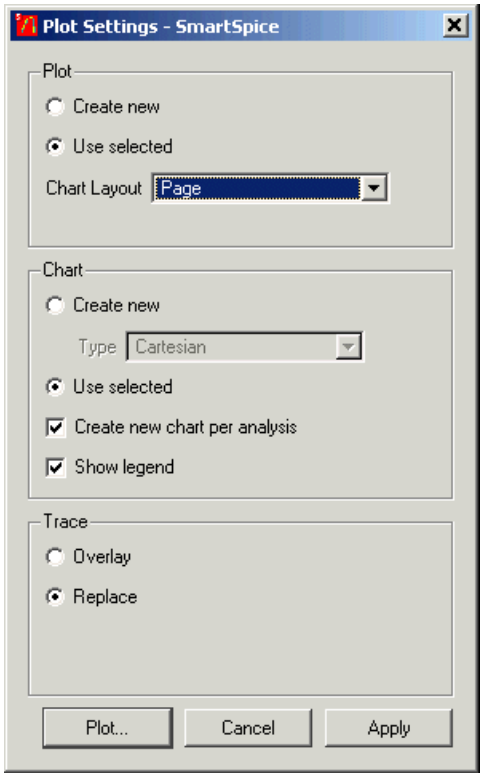


Figure 2-26 Plot Settings Dialog

11. Click **Plot** to display the selected vectors in the **Waveform Viewer** using the new plot settings:

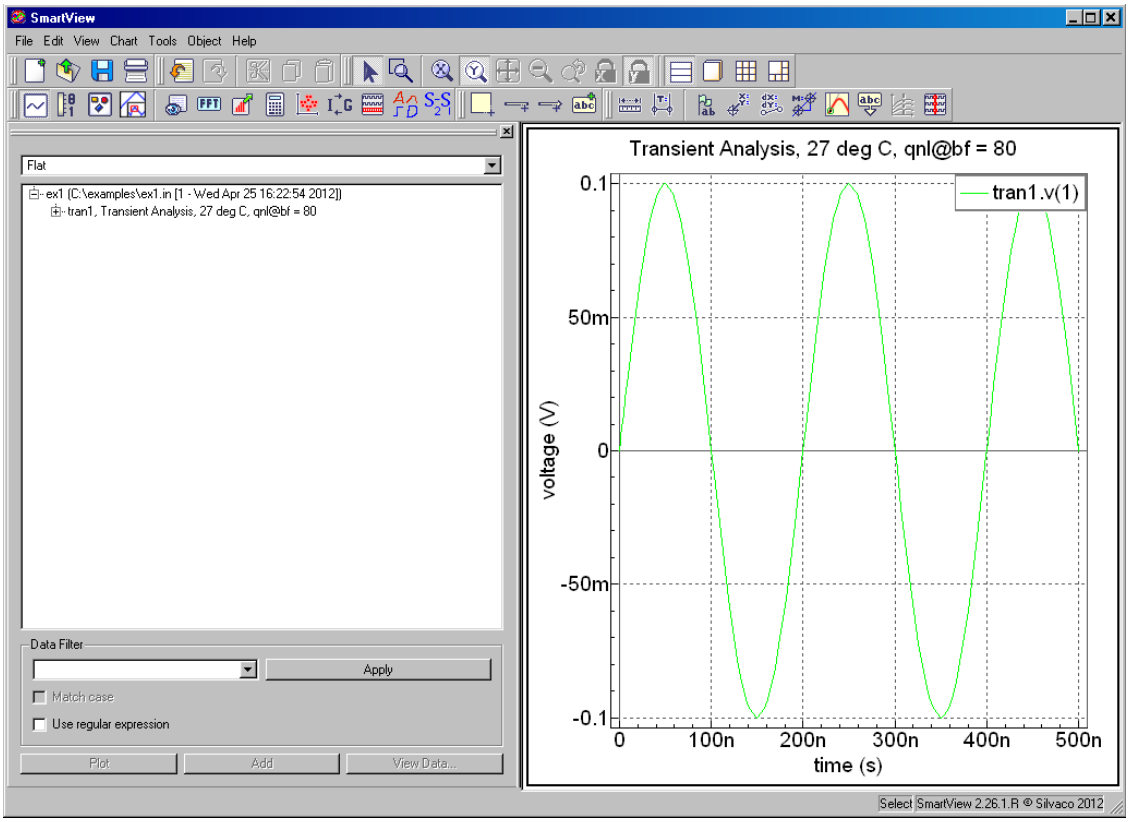


Figure 2-27 New Plot Settings

12. Select **Write Rawfile** from the **File** menu, browse to the required directory, enter the name of the new rawfile and click **Save**:

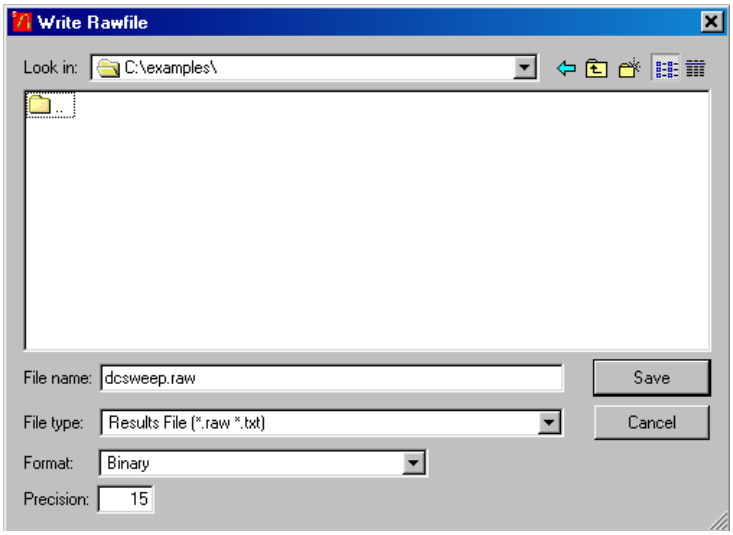


Figure 2-28 Write Rawfile

2.5 File Handling

The **File Browser** dialog used within SmartSpice may be invoked in a number of different ways depending on the context of its usage:

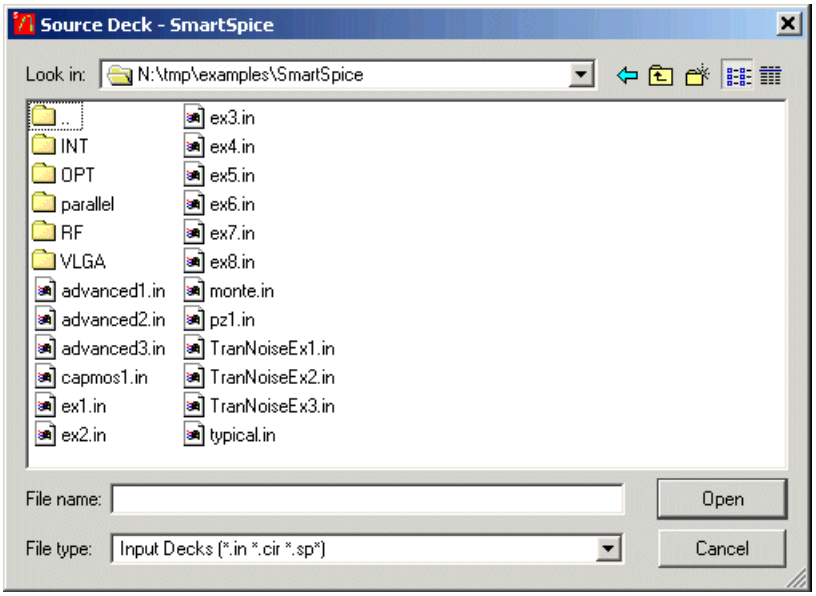


Figure 2-29 File Browser

However, in all cases, there are a number of common features:

- **Look In:** Displays the current directory whose files will be listed. You can type in the entire path to the location of the file you wish to load.
- **File List:** Displays the list of files in the current directory that match the current file type (see below). The file list can be ordered in a number of ways (alphabetically, file size, file date) by right clicking on the file list and selecting from the **Sort** menu. This menu also has a **Show hidden files** option that can be ticked so the file list contains hidden files.
- **File Name:** The name of the file that will be chosen for the context of the related action. If a file extension is not given as part of the file name, the default file extension will automatically be added.
- **File Type:** Filters the file list according to the specified file extension. To display all files (including extension-less files), use the **All Files (*)** filter.

2.6 Input Decks

2.6.1 Creating an Input Deck

Input decks can be created using the integrated TEXT EDITOR or an external one (see Section 2.23.9 Editor) by selecting **New** from the **File** menu:

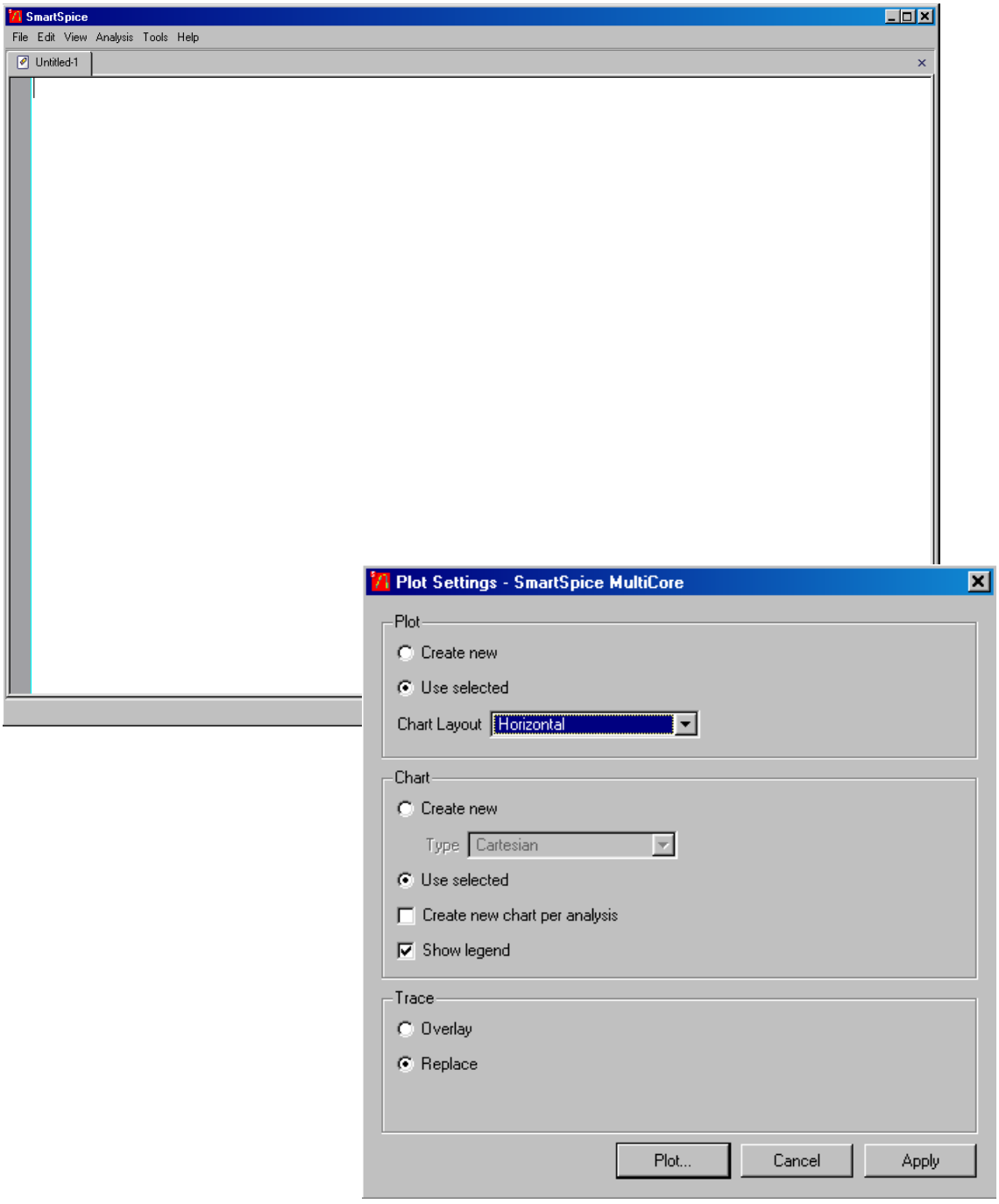


Figure 2-30 Creating a New Input Deck

For more information on the Silvaco Text Editor, refer to the Text Editor User's Manual.

2.6.2 Editing an Input Deck

Input decks can be opened for editing using the specified TEXT EDITOR (see Section 2.23.9 Editor) by selecting **Edit** from the **File** menu:

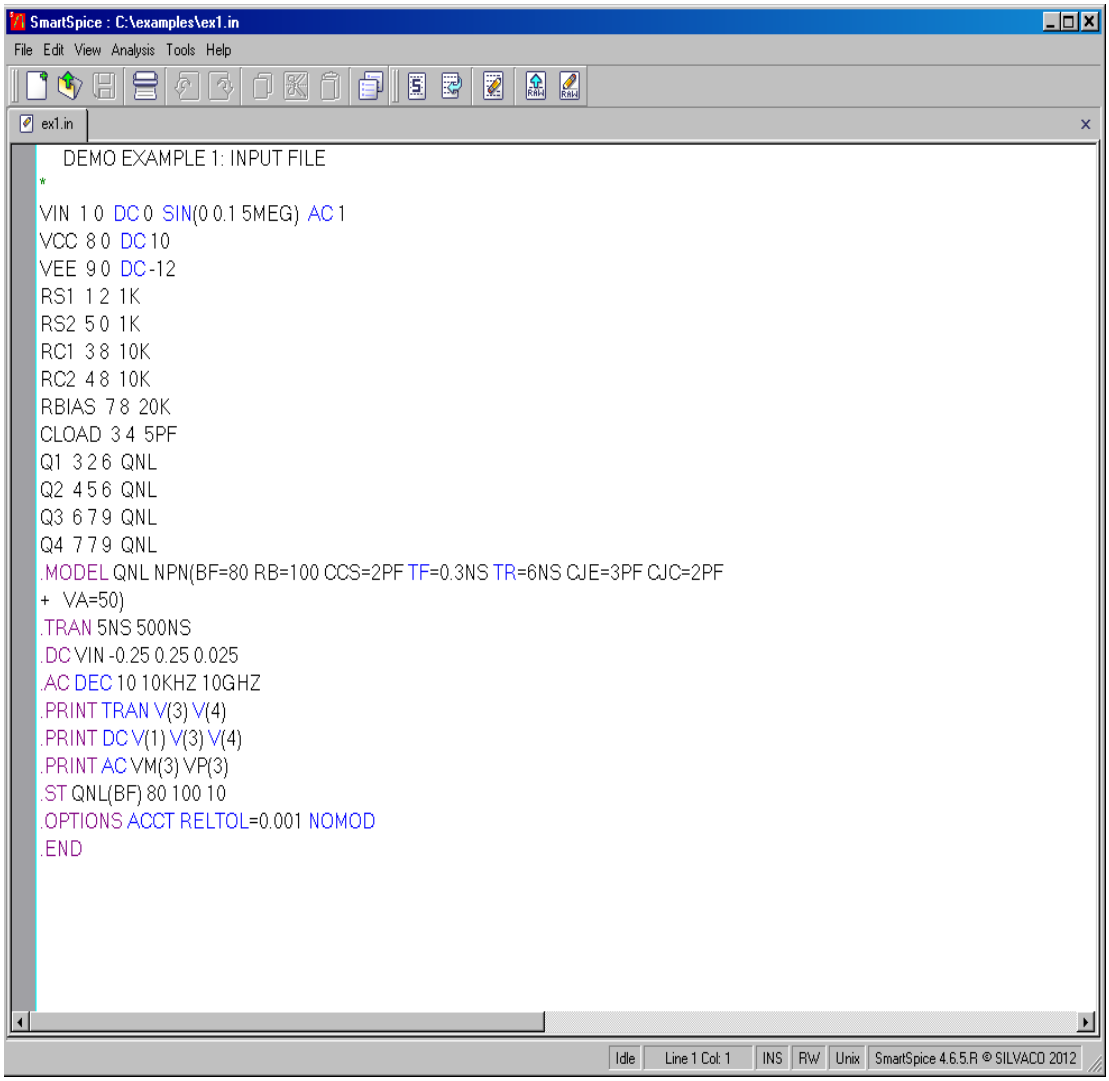


Figure 2-31 Editing an Input Deck

For more information on the Silvaco Text Editor, refer to the Text Editor User's Manual.

2.6.3 Sourcing an Input Deck

The **Source** action is used by SmartSpice to load an input deck and all related `.include/.lib` files/models into memory, and perform various error checking procedures on the created circuit.

There are several ways to read an Input deck into the simulation engine:

- Select **Source** from the **File** menu to display the **Source Deck** file browser dialog, and browse to the selected deck.
- Select a recently used deck in the **Recent Deck List** from the **File** menu.

Multiple decks can be read by using the Drag-and-Drop feature:

1. From a supported file browser, highlight the deck(s) to source.
2. Click and drag the files over the **Main** window before releasing the mouse button.

A **Progress** dialog will be displayed as each deck is sourced:

For more information, refer to the `source` command description in Chapter 5 Commands.

2.6.4 Updating a Sourced Input Deck

An Input deck that has already been sourced, but is subsequently modified, can be re-sourced by selecting **Update** from the **File** menu. This will create a new circuit using the updated input deck. A **Progress** dialog will be displayed as the deck is re-sourced.

For more information, refer to the `source` command description in Chapter 5 Commands.

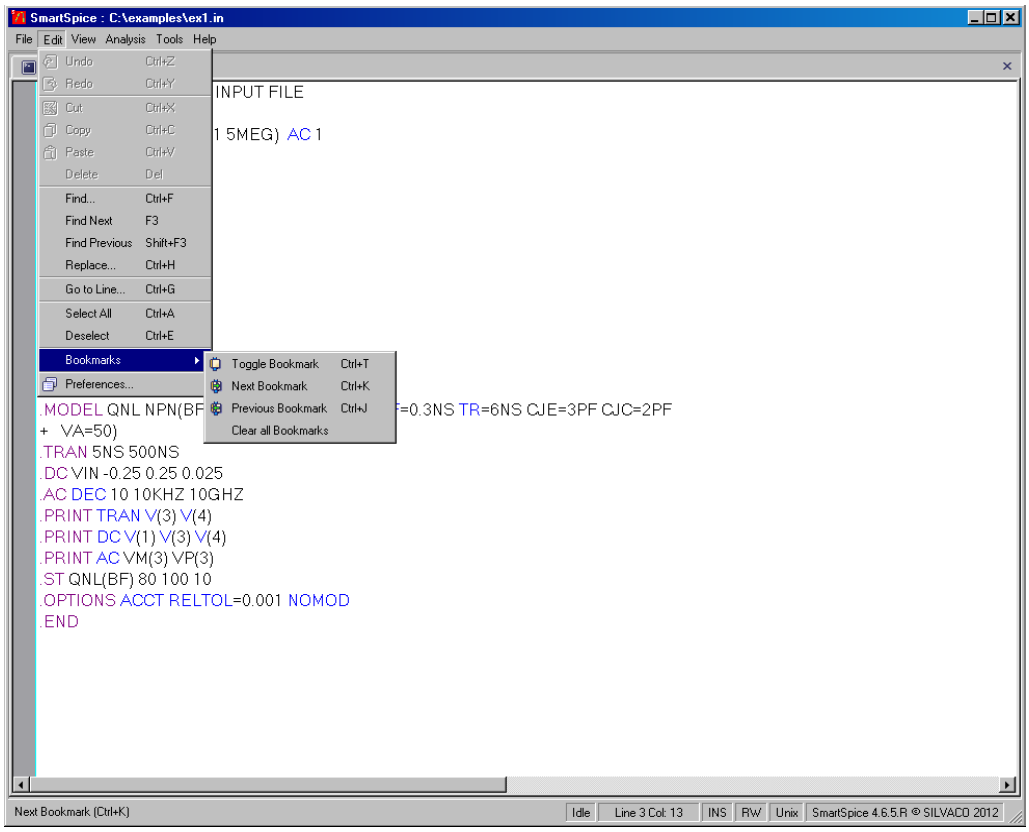


Figure 2-32 Integrated Editor's Menu

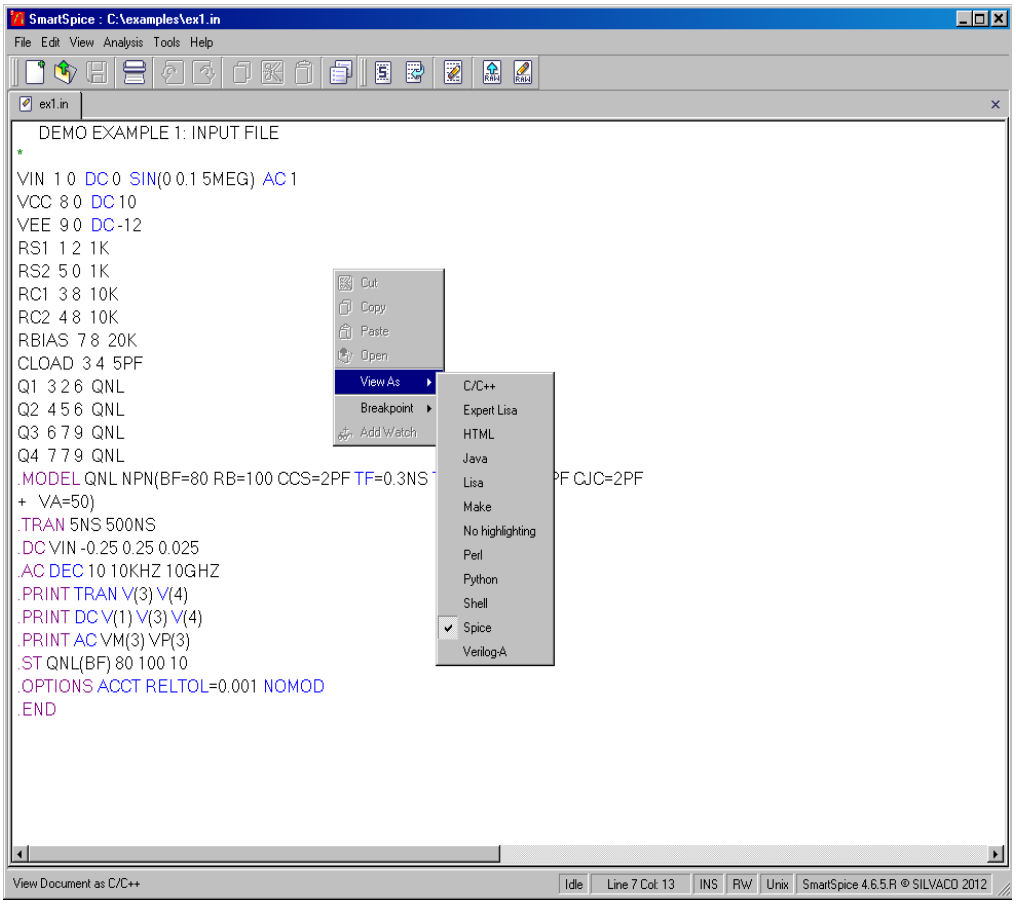


Figure 2-33 Editor Related Menu (See the SEDIT Manual)

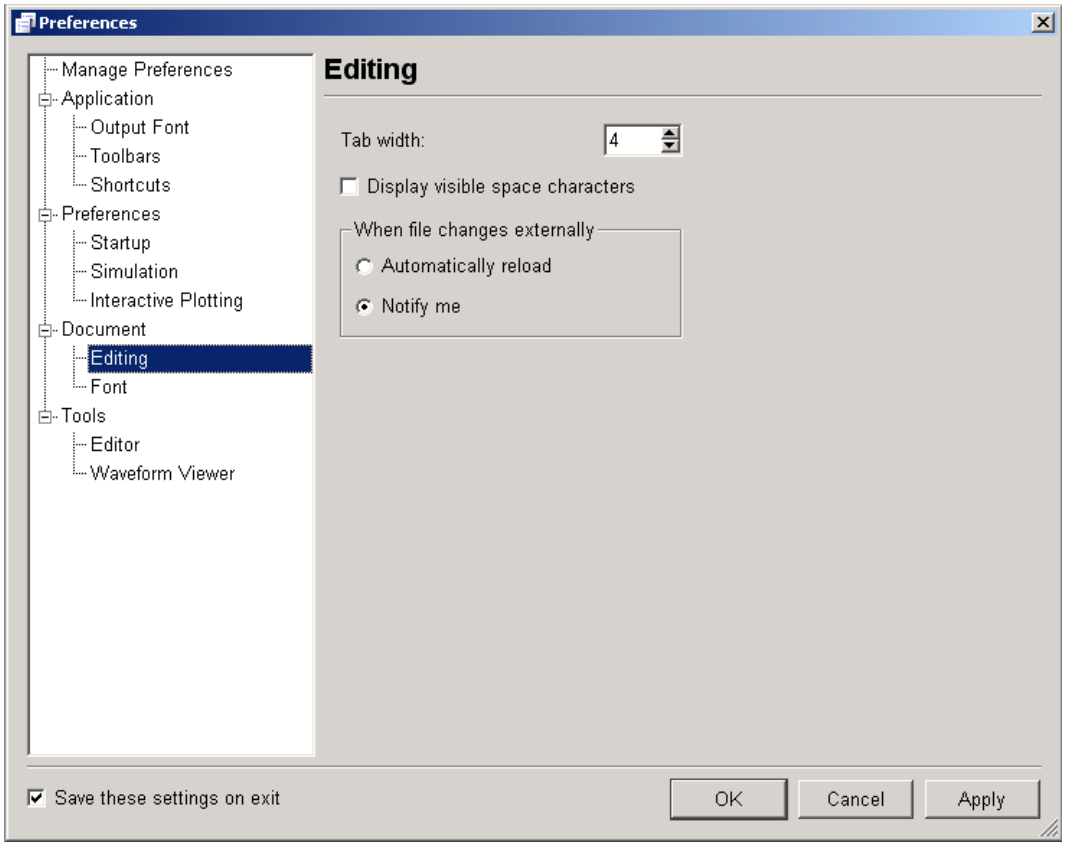


Figure 2-34 Editor Related Page in the Preferences Dialog (See the SEDIT Manual)

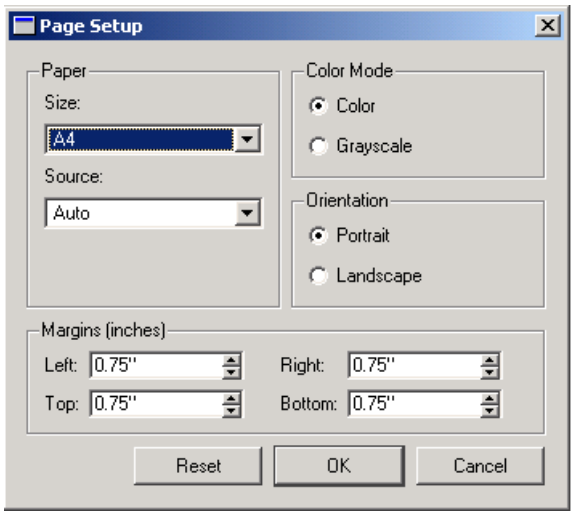


Figure 2-35 Page Setup Dialog (See the SEDIT Manual)

Figure 2-36 The Edit Toolbar

2.7 Rawfiles

2.7.1 Loading a Rawfile

Rawfiles can be loaded using the **Load Rawfile** dialog by selecting the **Load Rawfile** menu action:

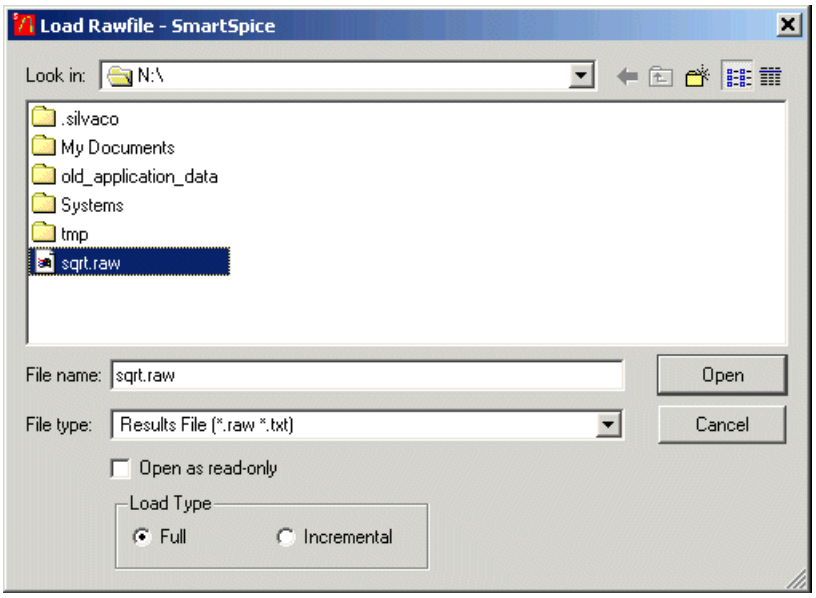


Figure 2-37 Loading a Rawfile

- **Open as read-only:** Opened file cannot be written to.
- **Load Type:** Files larger than 10-20Mbytes should be opened using the **Incremental** option to reduce load time and memory usage.

For more information, refer to the `load/iload` command descriptions in Chapter 5 Commands.

2.7.2 Writing a Rawfile

Vector data can be written to a rawfile using the **Write Rawfile** dialog by selecting the **Write Rawfile** menu action:

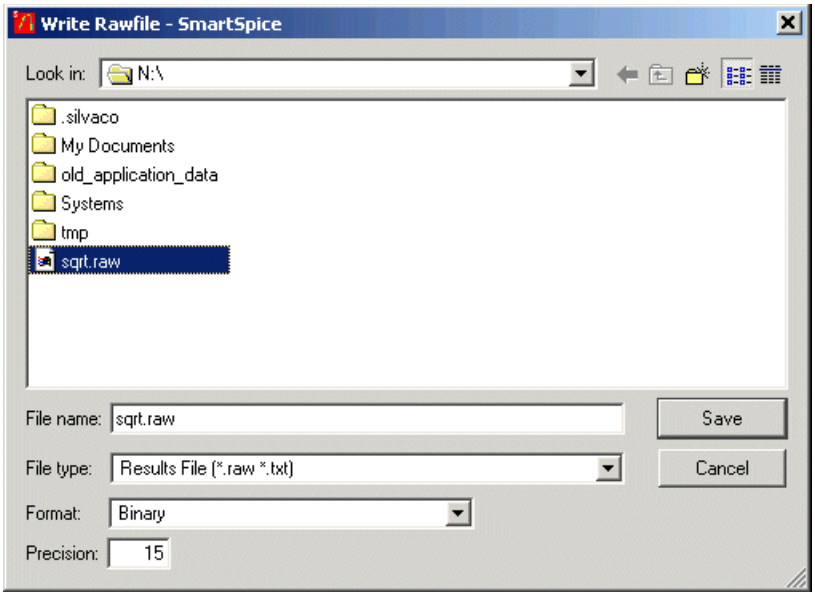


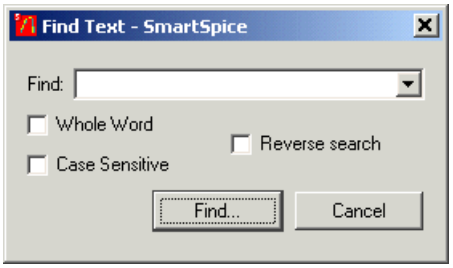
Figure 2-38 Writing a Rawfile

- **Format:** File format (binary by default).
- **Precision:** Vector data precision.

For more information, refer to the `write` command description in Chapter 5 Commands.

2.8 Find Text Dialog

Output information can be found in the **Output Log** window using **Find Text** dialog by selecting **Find in Output** menu action from the **Edit** menu:



Enter the text you are searching for into the **Find** combo box. Optionally, you can reverse the direction of the search, choose to match by whole word, or do a case sensitive search. If the end of the output data is reached, you can continue searching from the beginning of the output. You can also repeat an old search using the **Find** combo box, which saves all search parameters in the drop-down list.

2.9 View Decks Window

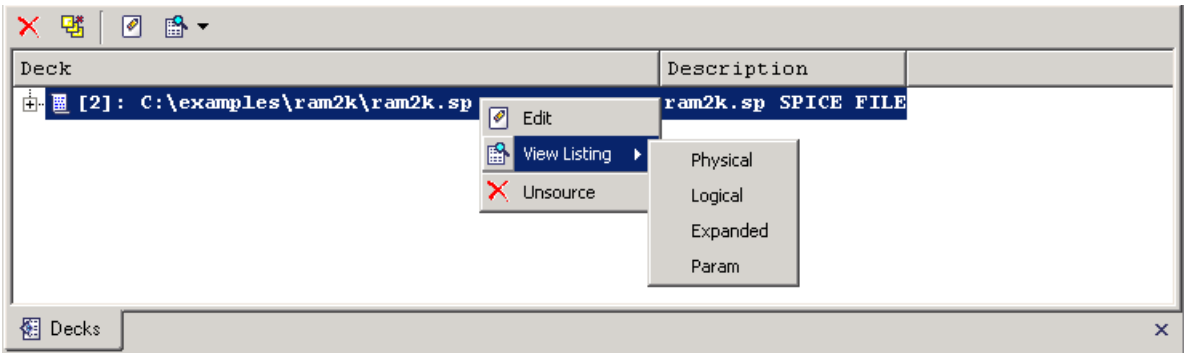


Figure 2-39 View Decks Window

The **Decks** browser can be opened with **View→Decks** menu item or using a key accelerator **Ctrl+D**. It displays a list of the loaded decks (circuits) with sub-tree of include files (if any). The sub-tree is built after a parent deck is sourced. The current input deck is highlighted in bold font.

Some manipulations are available from the attached toolbar or popup menu.

The toolbar's buttons (from left to right) are:

- **Unsource:** Unsource (delete) selected deck from simulator.
- **Unsource All:** Unsource (delete) all decks from simulator.
- **Edit:** Openthe selected input deck or include a file for editing.
- **View Listing:** Implements a submenu containing a set of options for listing a type of selected deck to be displayed. It is the alias for a `listing` command, the arguments of which are:
  - **Physical**
  - **Logical**
  - **Expanded**
  - **Param**

The items in context menu duplicate those on a toolbar.

You can hover a mouse pointer over a toolbar's icon to display a tooltip.

2.10 Commands History Window

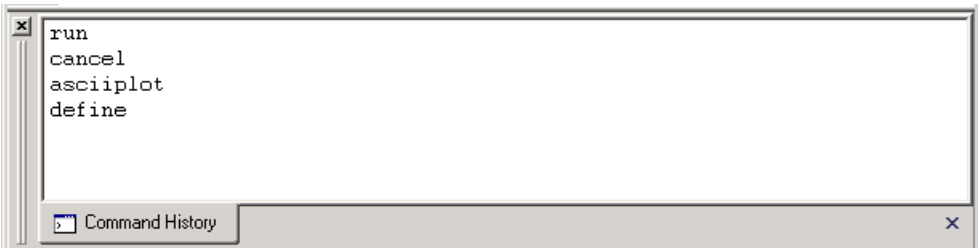


Figure 2-40 Commands History Window

All commands entered using the input box will be remembered and displayed with this window. It provides quick access to recently entered SmartSpice commands. You can select a command from the list and re-run it again with double-click or pressing the **Enter** key. It can be opened from the main menu with the **View→Command History** menu option. You can clear the contents of the window by right clicking on inside area and selecting **Clear** from the context menu.

2.11 Task List Browser

The **Task List** browser combines a list of accumulated tasks or jobs (left pane) and a list of vectors related to a selected task. It can be reached from the main menu using **View→Task List** item.

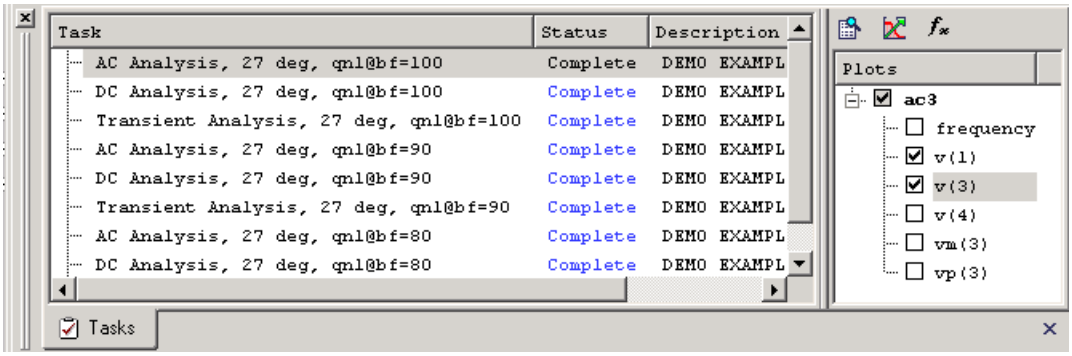


Figure 2-41 Task List Browser

List of tasks are divided to three columns, which are:

- **Task:** The task's properties, including analysis name, simulation, temperature and the parameters this task was run with.
- **Status:** Status of competition of a task. The status can be one of the following:
  - **Running**
  - **Paused**
  - **Stopped**
  - **Complete**
  - **Skipped**
  - **Deleted**
  - **Unknown**
- **Description:** The description taken from the input deck.

The task list can be cleaned up using a context menu item **Clear** (right click on a window). Also, from context menu, the attached input deck can be opened for editing, or a listing of a certain type can be displayed for circuit to which the selected task is related:

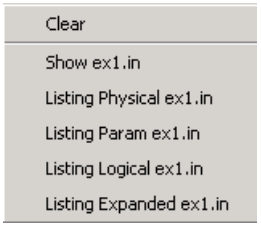


Figure 2-42 Task List Browser's Context Menu

The **Plots** pane represents a list of vectors or constants related to the selected task with three actions available over the checked items:

- **View Data:** Opens a viewer with textual table of vectors values.
- **Plot:** Launch a SmartView with graphical representation of the selected vectors.

- **Expression Wizard** (see Section 2.25 Expression Wizard)

2.12 Message List Window

The message list displays the errors and warnings emerged during simulation and allows a quick access to a line in an input deck which caused it.

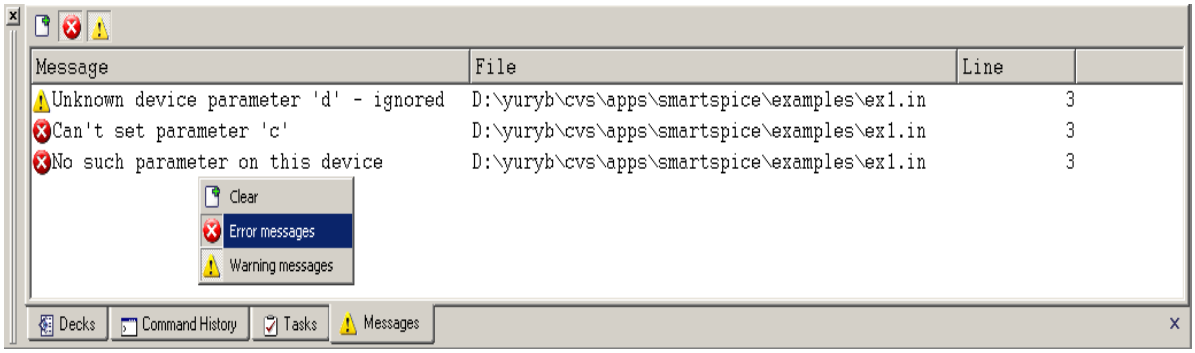


Figure 2-43 Message List Window

This window consists three columns: **Message**, **File name** and **Line number**. The **Message** displays specific text of error or warning. The columns **File name** and **Line number** display a file name and a line number where the error or warning appears.

To activate the **Message List** window (by default off), select the **Message List** command from the **View** main menu.

The **Error/Warning Log** window collects all error and warning messages.

Double click the left mouse button on any message in **Error/Warning Log** with a non-empty file reference starts a text editor and loads the file. The lines which caused an error will have a red background color, and a warning will have a yellow background color.

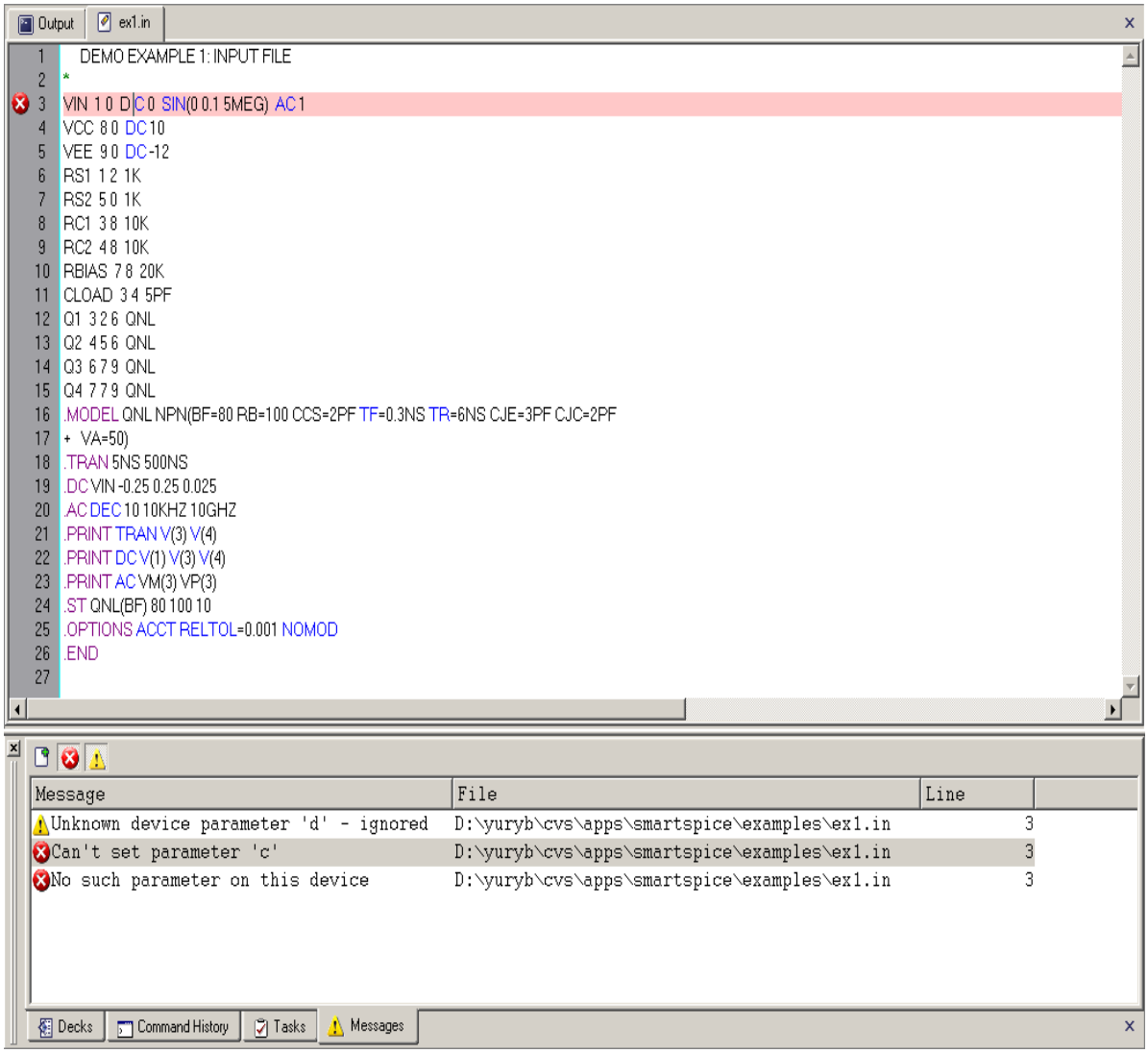


Figure 2-44 Line in the Input Deck Where an Error Occurred

Double click the right mouse button on the **Error/Warning Log** window activates a context menu with three options:

- **Clear:** Remove all messages.
- **Error messages:** When this option checked SmartSpice displays only errors.
- **Warning messages:** When this option checked SmartSpice displays only warnings.

SmartSpice automatically clears the **Error/Warning Log** when it sources a new deck, or re-sources an old one.

2.13 Save Vectors Dialog

The **Save Vectors** dialog is used to determine which vector data will be calculated and saved during a simulation, and can be displayed by selecting **Save Vectors** from the **Circuit** menu:

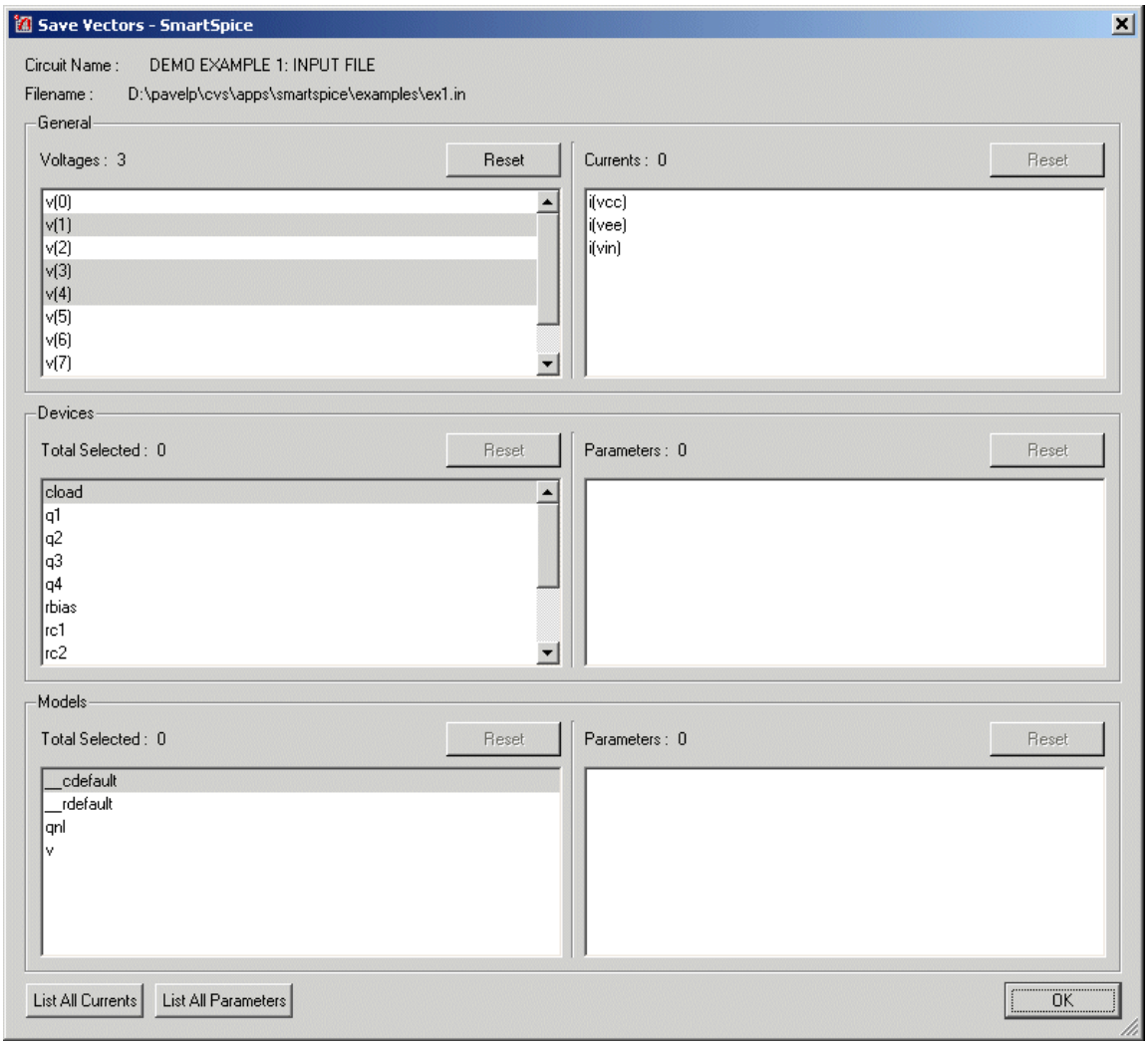
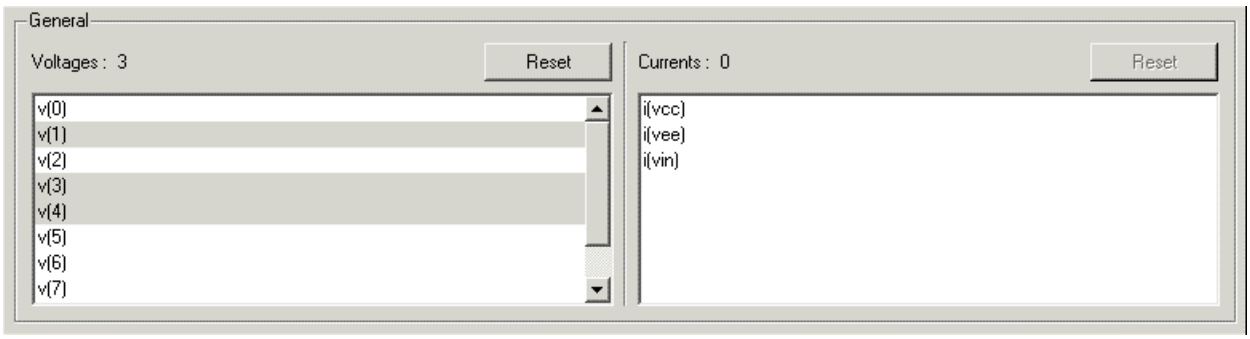


Figure 2-45 Save Vectors Dialog

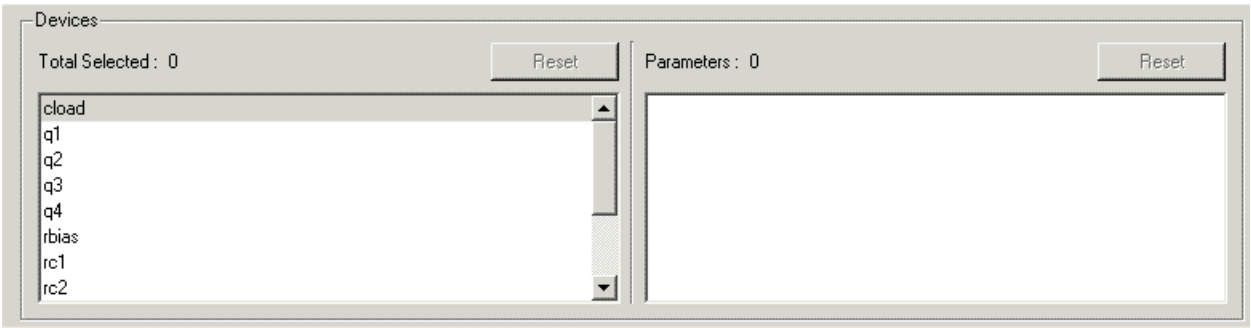
This dialog comprises several sections:

- **Deck Summary:** Displays **Circuit Name** and **Filename**:

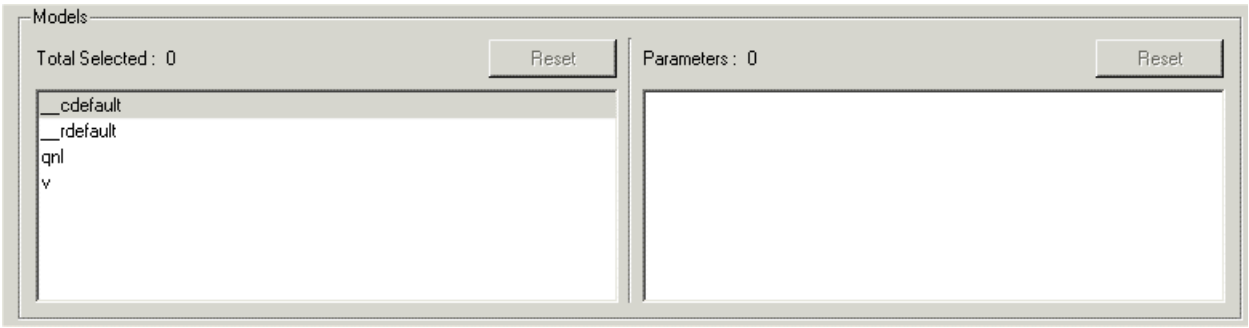
- **General:** Displays multiple selection lists and total number of selected **Voltages** and **Currents**. Click **List All Currents** to display the full list of Currents:



- **Devices:** Displays single selection list of **Devices**.
- **Device Parameters:** Displays a multiple selection list of **Device Parameters**. The total number of Device Parameters for the currently selected device is shown as the **Parameter** value. The total number of selected Device Parameters for all Devices is shown as the **Total Selected** value:



- **Models:** Displays single selection list of Models.
- **Model Parameters:** Displays a multiple selection list of **Model Parameters**. The total number of Model Parameters for the currently selected Model is shown as the **Parameter** value. The total number of selected Model Parameters for all Models is shown as the **Total Selected** value:



These sections consist of several sub-components:

- **Total Selected:** The number of vectors that will be calculated and saved.
- **Reset:** Unselects all selected vectors for the corresponding list.
- **Element List:** The list of vectors to be calculated and saved. Vectors will be highlighted by default if they are specified in the control statement.

Not all available currents will be displayed for efficiency; only currents through independent voltage sources, inductors, voltage controlled voltage sources, current controlled voltage sources, and non-linear dependent voltage sources are displayed by default. Click **List All Currents** to list all of the available currents.

Similarly, not all parameters are displayed by default. Click **List All Parameters** to list all of the available parameters.

Click the **OK** button to close the dialog.

2.14 Trace Vectors Dialog

The **Trace Vectors** dialog is used to determine which vector data will be output during a simulation, and can be displayed by selecting **View→Circuit** menu:

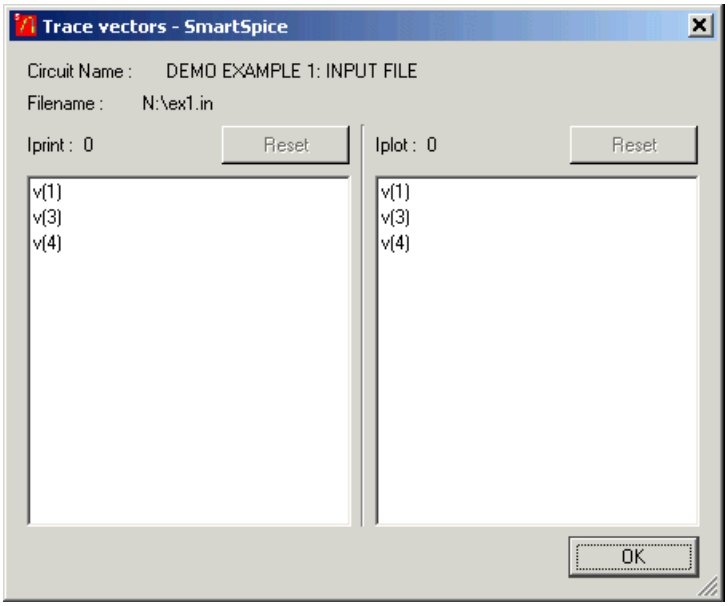


Figure 2-46 Trace Vectors Dialog

This dialog comprises several sections:

- **Deck Summary:** Displays **Circuit Name** and **Filename**:

Click the **OK** button to close the dialog.

2.14.1 Interactive Printing

This section displays a multiple selection list of Vectors whose values will be displayed on the **Run Time** dialog during a simulation:

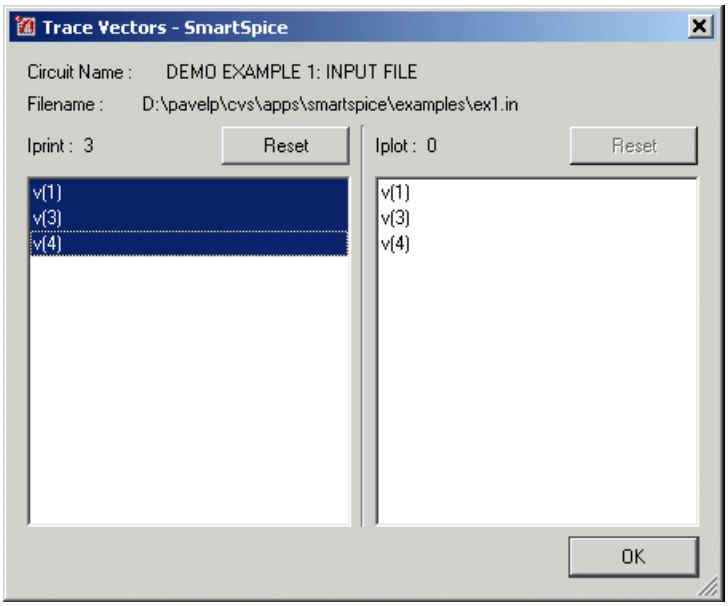


Figure 2-47 List of Vectors

This section consists of several sub-components:

- **Iprint:** The number of vectors whose data will be displayed in the run time dialog.
- **Reset:** Unselects all selected vectors for the corresponding list.
- **Element List:** The list of vectors whose data will be displayed in the **Run Time** dialog to be calculated and saved. Only vectors that are calculated and saved will be displayed in the list. Vectors will be highlighted by default if they are specified in the control statement.

The selected Vectors will be displayed in the **Run Time** screen during a simulation.

2.14.2 Interactive Plotting

This section displays a multiple selection list of Vectors that will be plotted during a simulation:

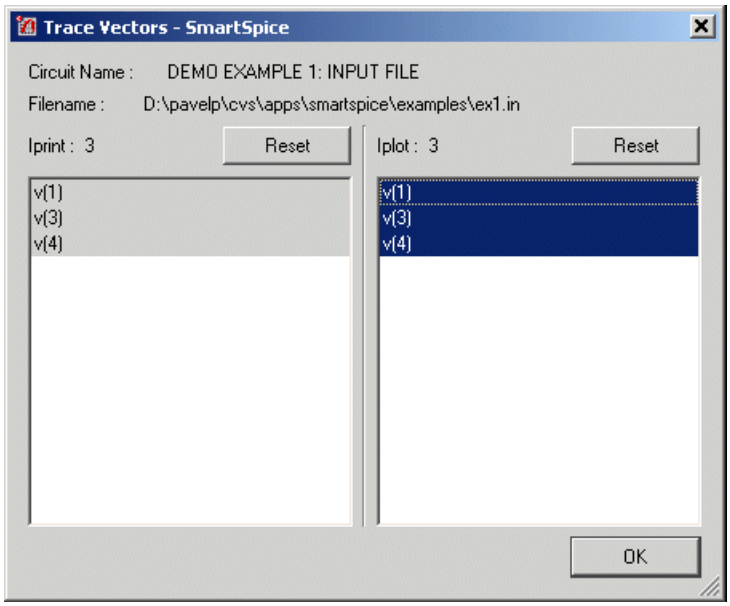


Figure 2-48 Vectors to be Plotted During a Simulation

This section consists of several sub-components:

- **Iplot:** The number of vectors whose data will be displayed using the waveform viewer.
- **Reset:** Unselects all selected vectors for the corresponding list.
- **Element List:** The list of vectors whose data will be displayed using the **Waveform Viewer**. Only vectors that will be calculated and saved will be displayed in the list. Vectors will be highlighted by default if they are specified in the control statement.

The selected Vectors will be displayed in the waveform viewer during a simulation:

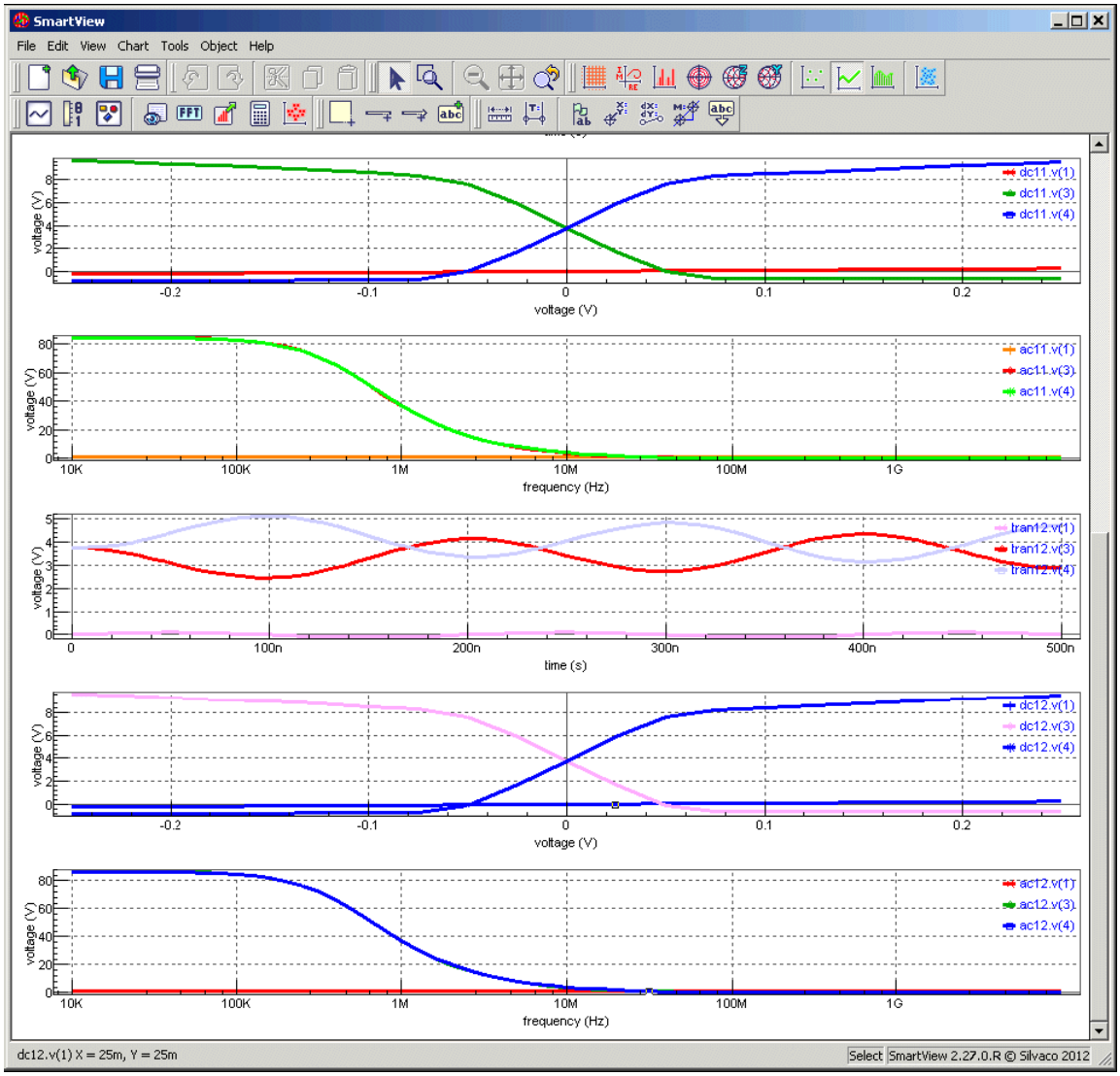


Figure 2-49 Waveform Viewer

2.15 Performing a Simulation

2.15.1 Run

A simulation can consist of multiple analyses and post-processor actions that can be executed by selecting **Run** from the **Analysis** menu. The **Run Time** dialog will be displayed for the duration of the simulation (see Section 2.15 Performing a Simulation, and Section 2.17 Run Time Screen).

For more information, refer to the `run` command description in Chapter 5 Commands.

2.16 Analyses Manager Dialog

The **Analyses Manager** dialog can be used to start individual types of analysis when there is a circuit generated from the currently sourced input deck:

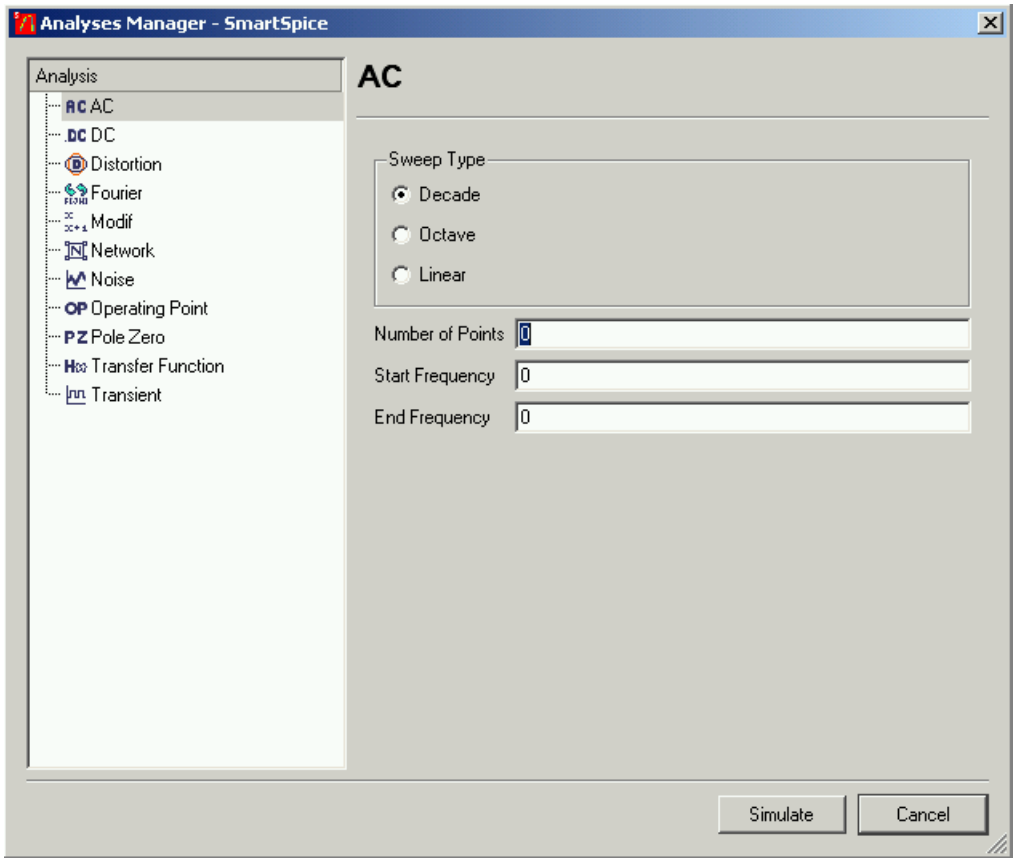


Figure 2-50 Analyses Manager Dialog

Where applicable, the following parameter validation is performed to ensure that the simulation is started using the correct command format:

- **Alpha:** Alphabetical letters (A-Z).
- **Alphanumerical:** Alphabetical letters (A-Z) and numbers (0-9).
- **Engineering:** This is an engineering format specific to SmartSpice:

Table 2-1: Engineering Format

String	Multiplier
T	10^{12}
G	10^{9}
x	10^{6}
meg	10^{6}
k	10^{3}
m	10^{-3}
u	10^{-6}
n	10^{-9}
p	10^{-12}
f	10^{-15}
a	10^{-18}

Controls will be enabled if they are available in the analysis' current context. The following buttons apply to all analysis dialogs:

- **Simulate:** Perform the simulation. Will only be enabled if all validation rules have been passed.
- **Cancel:** Close the dialog.

2.16.1 AC Analysis

An AC analysis can be performed to create a linearized small-signal model at the operating point of the current circuit for computing the frequency response over a user-specified range of frequencies. The analysis parameters are displayed by selecting **AC** from the Analysis menu:

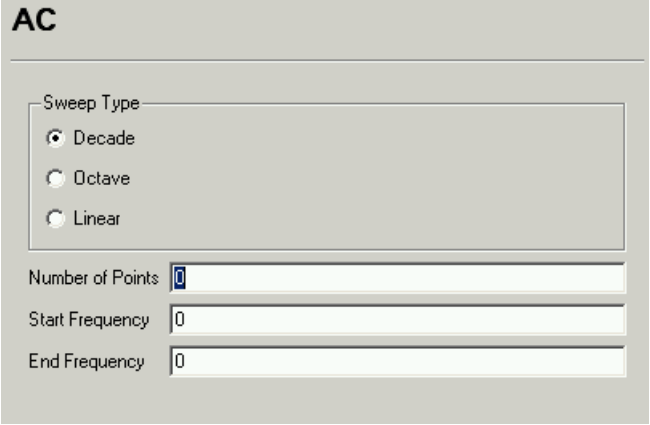


Figure 2-51 Selecting AC in the Analysis Menu

- **Decade:** Sweep by decades.
- **Octave:** Sweep by octaves.
- **Linear:** Linear sweep.
- **Number of Points:** Positive integers > 0.
- **Start Frequency:** Engineering format > 0.0.
- **End Frequency:** Engineering format > 0.0.

Additional validation rules include:

```
Start Frequency < End Frequency
```

For more information, refer to Chapter 3 Statements, Section .AC (AC Analysis).

2.16.2 DC Analysis

A DC analysis can be performed to compute a DC transfer curve for the current circuit with capacitors opened and inductors shorted. The analysis parameters are displayed by selecting **DC** from the Analysis menu:

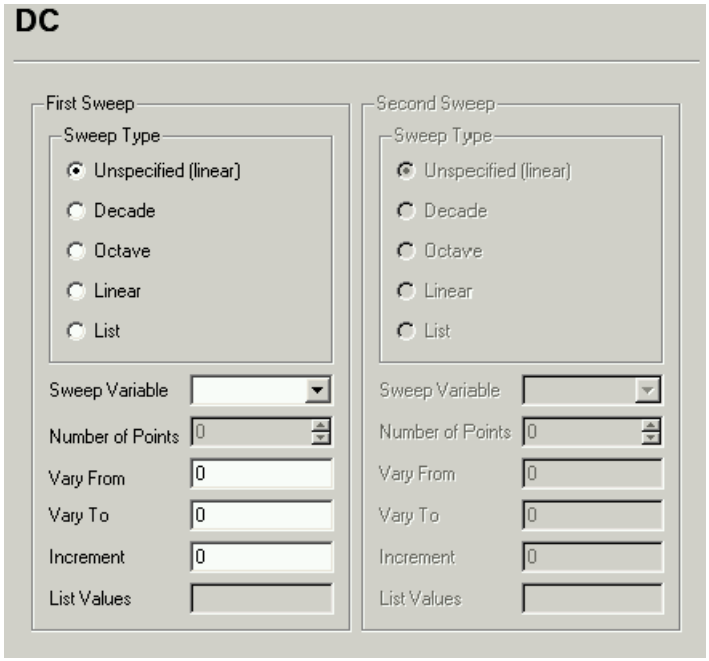


Figure 2-52 Selecting DC from the Analysis Menu

- **Decade:** Sweep by decades.
- **Octave:** Sweep by octaves.
- **Linear:** Linear sweep type.
- **List:** Sweep over this list of values.
- **Sweep Variable:** Input deck defined swept parameter.
- **Vary From:** Engineering format.
- **Vary To:** Engineering format.
- **Increment:** Engineering format.
- **List Values:** Alphanumeric.

For more information, refer to Chapter 3 Statements, Section .DC (DC Analysis).

2.16.3 Transient Analysis

A Transient analysis can be performed to calculate the current circuit's behavior over a specified time interval. The analysis parameters are displayed by selecting **Transient** from the Analysis menu:

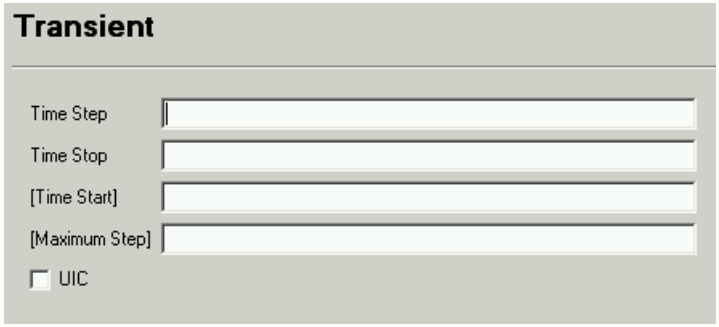


Figure 2-53 Selecting Transient from the Analysis Menu

- **Time Step:** Engineering format >= 0.0.
- **Time Stop:** Engineering format >= 0.0.
- **Time Start** (optional)**:** Engineering format > 0.0.
- **Maximum Step** (optional)**:** Engineering format >= 0.0.
- **UIC:** Use initial conditions.

Additional validation rules include:

```
Time Start < Time Stop
Time Step < Maximum Step
```

For more information, refer to Chapter 3 Statements, Section .TRAN (Transient Analysis).

2.16.4 Linearizing Transient Vectors

The selected vectors from a selected Transient analysis can be forced to conform to a linear scale by selecting **Linearize** from the **Transform** menu.

For more information, refer to the `linearize` command description in Chapter 5 Commands.

2.16.5 Transient Analysis FFT

A Fast Fourier Transformation (FFT) can be performed on the selected vectors from a selected Transient analysis by selecting **FFT** from the **Transform** menu.

For more information, refer to the `fft` command description in Chapter 5 Commands.

2.16.6 Noise Analysis

A Noise analysis can be performed on the current circuit to calculate the noise contributions of each device to the output port and the equivalent input noise. The analysis parameters are displayed by selecting **Noise** from the **Analysis** menu:

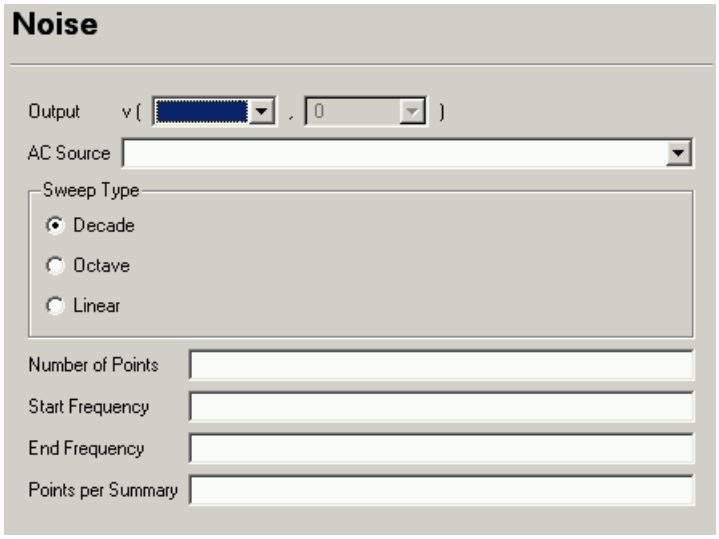


Figure 2-54 Selecting Noise from the Analysis Menu

- **Decade:** Sweep by decades.
- **Octave:** Sweep by octaves.
- **Linear:** Linear sweep type.
- **Output:** Output port.
- **Number of Points:** Positive integers > 0.
- **Start Frequency:** Engineering format > 0.0.
- **End Frequency:** Engineering format > 0.0.
- **Points per Summary:** Positive integers > 0.
- **AC Source:** Input deck defined AC Source.

Additional validation rules include:

```
Start Frequency < End Frequency
```

For more information, refer to Chapter 3 Statements, Section .NOISE (Noise Analysis).

2.16.7 Distortion Analysis

A Distortion analysis can be performed on the current circuit to complete a multi-dimensional Volterra series analysis using multi-dimensional Taylor series to represent the nonlinearities at the operating point. The analysis parameters are displayed by selecting **Distortion** from the Analysis menu:

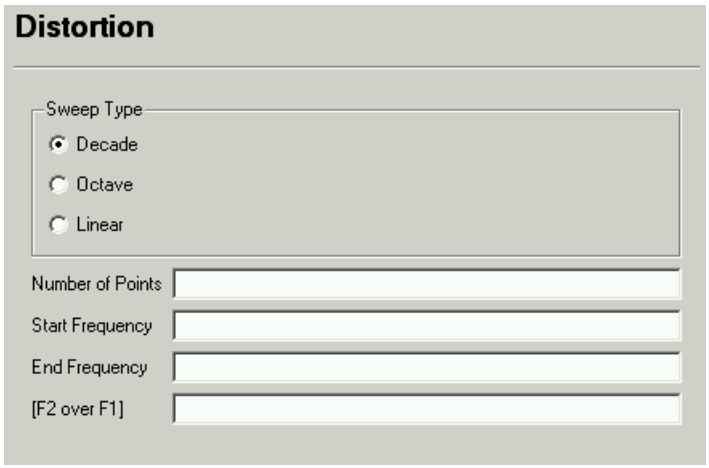


Figure 2-55 Distortion from the Analysis Menu

- **Decade:** Sweep by decades.
- **Octave:** Sweep by octaves.
- **Linear:** Linear sweep type.
- **Number of Points:** Positive integers > 0.
- **Start Frequency:** Engineering format >= 0.0.
- **End Frequency:** Engineering format >= 0.0.
- **F2 over F1** (optional)**:** Engineering format >= 0.0.

Additional validation rules include:

```
Start Frequency < End Frequency
```

For more information, refer to Chapter 3 Statements, Section .DISTO (Distortion Analysis).

2.16.8 Transfer Function Analysis

A Transfer Function analysis can be performed on the current circuit to define the small-signal output and input for the DC small-signal analysis. The analysis parameters are displayed by selecting **TF** from the **Analysis** menu:

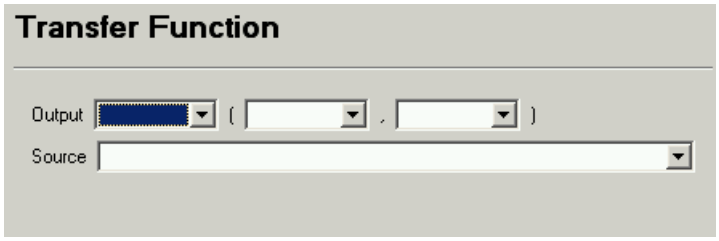


Figure 2-56 Selecting TF from the Analysis Menu

- **Output:** Output port.
- **Source:** Input deck defined Source.

For more information, refer to Chapter 3 Statements, Section .TF (Transfer Function).

2.16.9 Network Analysis

A Network analysis can be performed to extract all types of two-port network parameters from the current circuit. The analysis parameters are displayed by selecting **Network** from the **Analysis** menu:

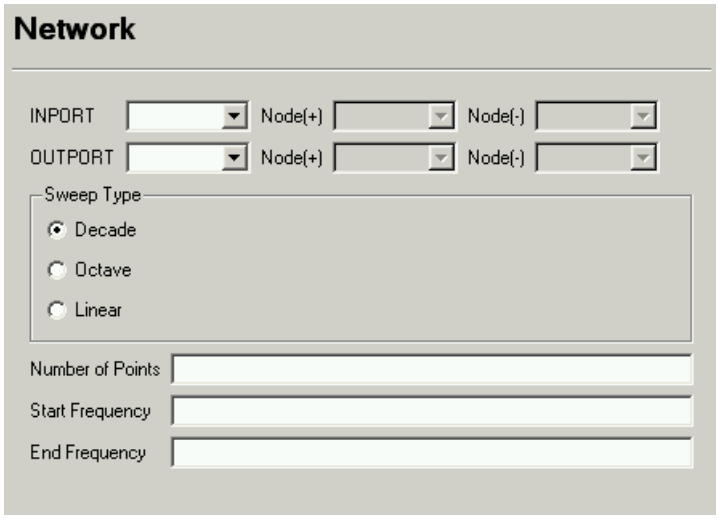


Figure 2-57 Selecting Network from the Analysis Menu

- **Decade:** Sweep by decades.
- **Octave:** Sweep by octaves.
- **Linear:** Linear sweep type.
- **Number of Points:** Positive integers > 0.
- **Start Frequency:** Engineering format >= 0.0.
- **End Frequency:** Engineering format >= 0.0.
- **Port 1 AC Source:** Input deck defined AC Source.

- **Port 2 AC Source:** Input deck defined AC Source.

Additional validation rules include:

```
Start Frequency < End Frequency
```

For more information, refer to Chapter 3 Statements, Section .NET (Small-Signal Network Analysis).

2.16.10 Operating Point Analysis

An Operating Point analysis can be performed to determine the DC operating point of the current circuit with inductors shorted and capacitors opened. The analysis parameters are displayed by selecting **OP** from the **Analysis** menu:

Figure 2-58 Selecting OP from the Analysis Menu

For more information, refer to Chapter 3 Statements, Section .OP (Operating-Point Analysis).

2.16.11 Pole Zero Analysis

A Pole Zero analysis can be performed on the current circuit by creating a linearized small-signal model at the operating point of the circuit, and computing the poles and zeros of the transfer function. The analysis parameters are displayed by selecting **PZ** from the **Analysis** menu:

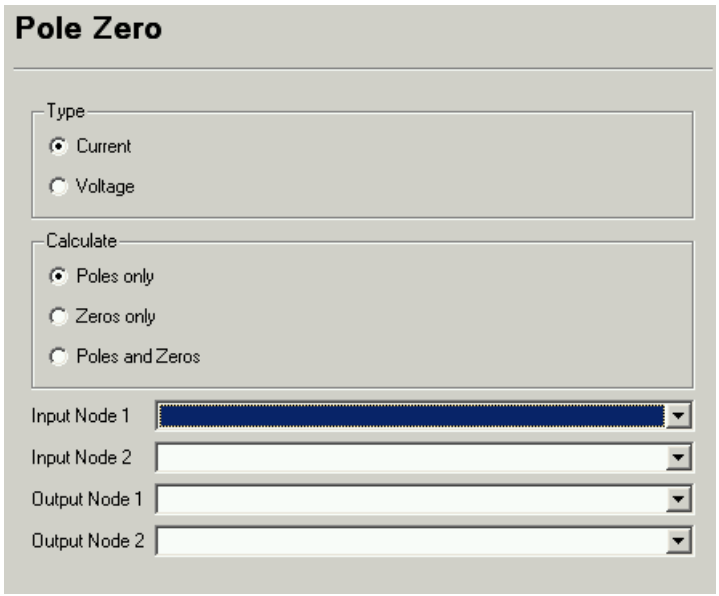


Figure 2-59 Selecting PZ from the Analysis Menu

- **Current:** Use the (output voltage)/(input current) transfer function form.
- **Voltage:** Use the (output voltage)/(input voltage) transfer function form.
- **Poles only:** Only compute the poles.
- **Zeros only:** Only compute the zeros.
- **Poles and Zeros:** Compute both the poles and zeros.
- **Input Node 1:** Input deck defined node.
- **Input Node 2:** Input deck defined node.
- **Output Node 1:** Input deck defined node.
- **Output Node 2:** Input deck defined node.

Additional validation rules include:

```
Input Node 1 != Input Node 2
Output Node 1 != Output Node 2
```

For more information, refer to Chapter 3 Statements, Section .PZ (Pole-Zero Analysis).

2.16.12 MODIF Analysis

A MODIF analysis can be performed to simulate the current circuit for some sets of parameters as they are modified over a user-specified range of values. The analysis parameters are displayed by selecting **MODIF** from the **Analysis** menu:

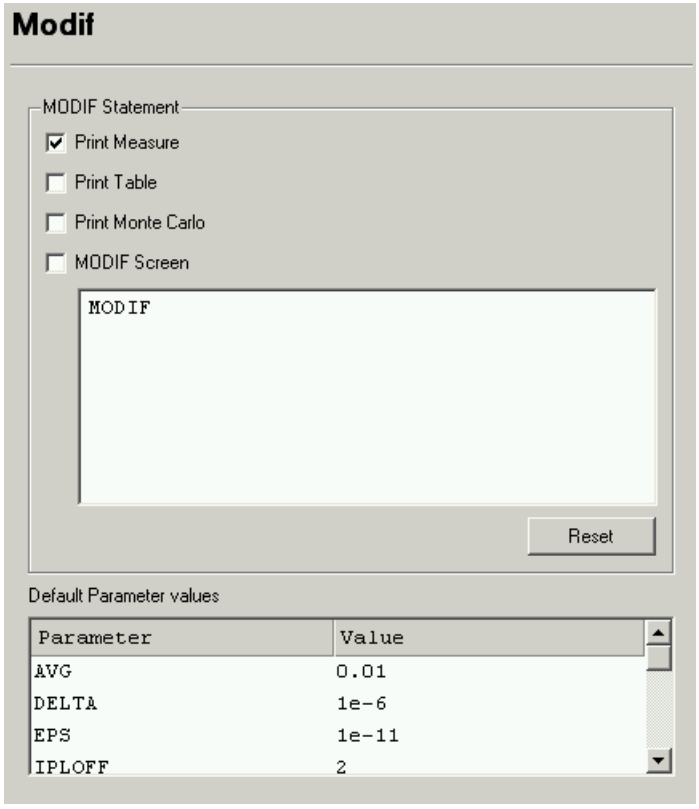


Figure 2-60 Selecting MODIF from the Analysis Menu

- **Print Measure:** Adds the **PROFF** option to the MODIF statement to suppress online printing of measures specified and calculated by means of `.MEASURE` statements for the current set of parameters.
- **Print Table:** Adds the **PRTBL** option to the MODIF statement to print the final table of all parameters and measures calculated for the current set of parameters.
- **Print Monte Carlo:** Adds the **PRMC** option to the MODIF statement to print Mean, Variance, Sigma, and Average Deviation for each measure calculated during Monte-Carlo analysis.
- **MODIF Screen:** Adds the **SCREEN** option to the MODIF statement to display detailed simulation information in the **MODIF Screen** dialog during the analysis:

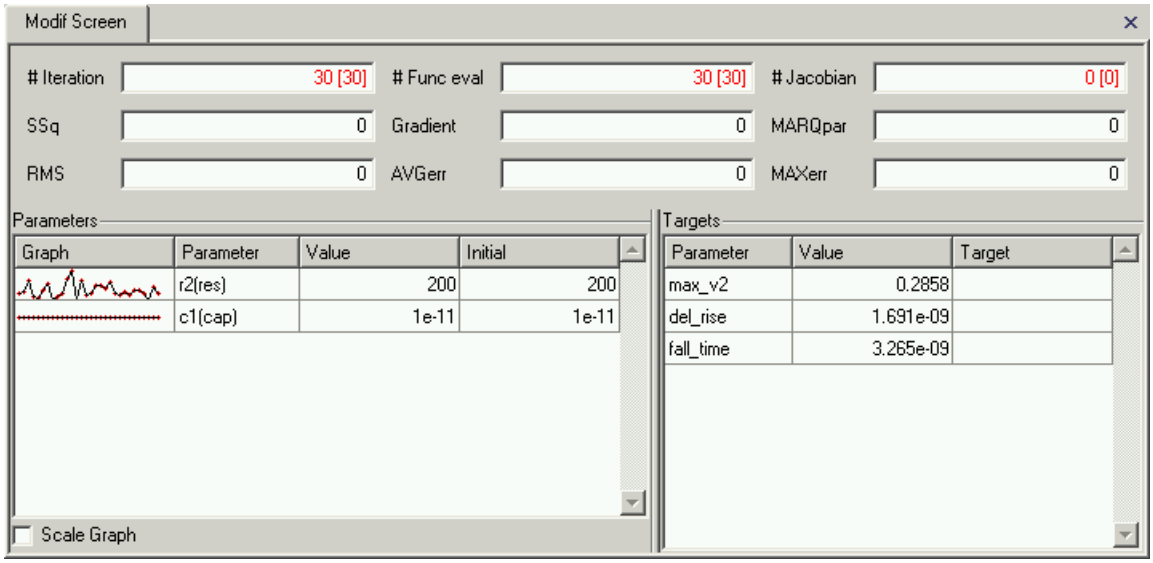


Figure 2-61 MODIF Screen Dialog

- **Stop:** Abort the MODIF analysis.
- **Close:** Closes the **MODIF Screen** dialog.

Additional MODIF analysis dialog validation rules include:

- MODIF statement must begin with the keyword MODIF.

For more information, refer to Chapter 3 Statements, Section .MODIF (Parameter Modification).

2.16.13 Fourier Analysis

A Fourier analysis can be performed on the results of the current selected Transient analysis by selecting **Fourier** from the Transform menu:

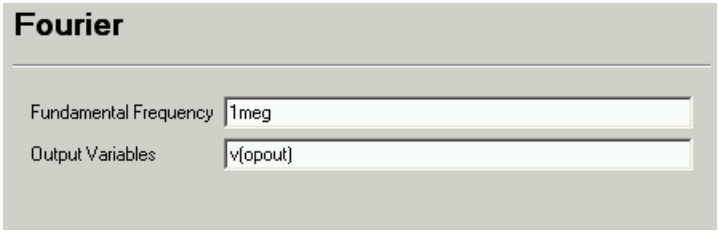
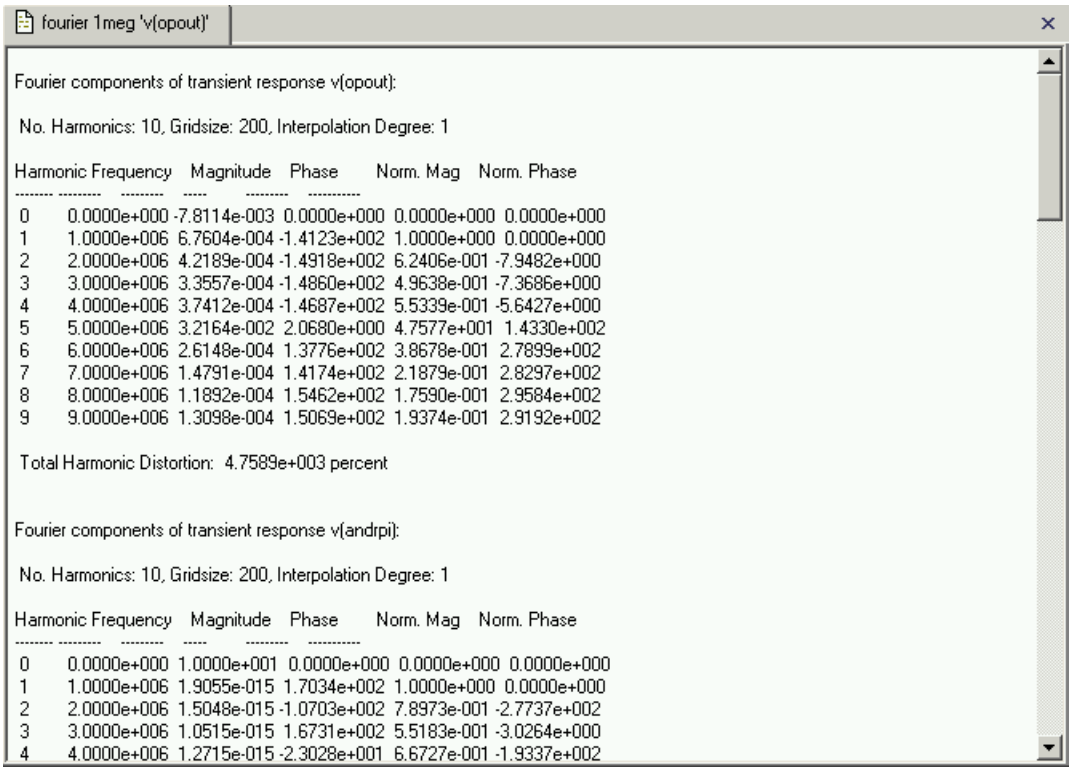


Figure 2-62 Selecting Fourier from the Transform Menu

- **Fundamental Frequency:** The fundamental frequency of the Fourier analysis.
- **Output Variables:** Output variables for which harmonics are calculated.

For more information, refer to Chapter 3 Statements, Section .FOUR (Fourier Analysis).

The Fourier analysis' calculated harmonics are displayed in the **Fourier Output** dialog:


fourier 1meg 'v(opout)'

Fourier components of transient response v(opout):

No. Harmonics: 10, Gridsize: 200, Interpolation Degree: 1

Harmonic	Frequency	Magnitude	Phase	Norm. Mag	Norm. Phase
0	0.0000e+000	-7.8114e-003	0.0000e+000	0.0000e+000	0.0000e+000
1	1.0000e+006	6.7604e-004	-1.4123e+002	1.0000e+000	0.0000e+000
2	2.0000e+006	4.2189e-004	-1.4918e+002	6.2406e-001	-7.9482e+000
3	3.0000e+006	3.3557e-004	-1.4860e+002	4.9638e-001	-7.3686e+000
4	4.0000e+006	3.7412e-004	-1.4687e+002	5.5339e-001	-5.6427e+000
5	5.0000e+006	3.2164e-002	2.0680e+000	4.7577e+001	1.4330e+002
6	6.0000e+006	2.6148e-004	1.3776e+002	3.8678e-001	2.7899e+002
7	7.0000e+006	1.4791e-004	1.4174e+002	2.1879e-001	2.8297e+002
8	8.0000e+006	1.1892e-004	1.5462e+002	1.7590e-001	2.9584e+002
9	9.0000e+006	1.3098e-004	1.5069e+002	1.9374e-001	2.9192e+002

Total Harmonic Distortion: 4.7589e+003 percent

Fourier components of transient response v(andrpi):

No. Harmonics: 10, Gridsize: 200, Interpolation Degree: 1

Harmonic	Frequency	Magnitude	Phase	Norm. Mag	Norm. Phase
0	0.0000e+000	1.0000e+001	0.0000e+000	0.0000e+000	0.0000e+000
1	1.0000e+006	1.9055e-015	1.7034e+002	1.0000e+000	0.0000e+000
2	2.0000e+006	1.5048e-015	-1.0703e+002	7.8973e-001	-2.7737e+002
3	3.0000e+006	1.0515e-015	1.6731e+002	5.5183e-001	-3.0264e+000
4	4.0000e+006	1.2715e-015	-2.3028e+001	6.6727e-001	-1.9337e+002

Figure 2-63 Fourier Output Dialog

2.17 Run Time Screen

The **Run Time** window is displayed during an interactive simulation to provide you with simulation progress and the ability to abort the simulation, unless you have unchecked the **Show run-time screen** option on the **Simulation** page in the **Preferences** dialog:

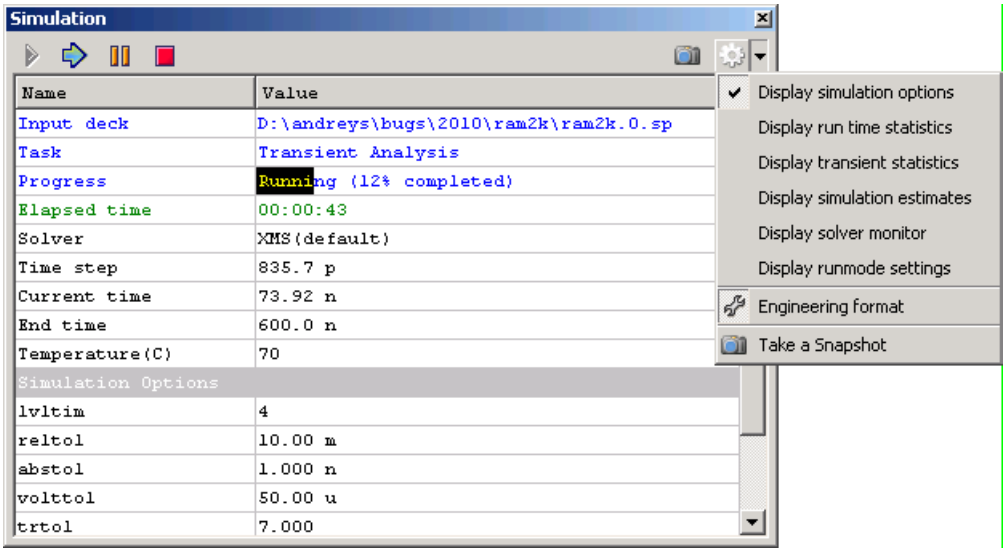


Figure 2-64 Run Time Screen

The **Run Time** screen displays simulation details, current analysis type and the progress as a percentage of the overall analysis. Simulation parameter data (e.g., Time step, Current Time, End time) for the selected vectors (see Section 2.14 Trace Vectors Dialog) is displayed to indicate detailed progress information for the current analysis.

The **Run Time** dialog supports several actions:

Stop

- **Skip:** Abort the current analysis and perform the next analysis in the simulation.

  This button is only available during a Transient analysis that is performing parametric analysis and when there are multiple analyses being performed in the simulation.

- **Pause:** Pause the current analysis. Once paused, vector data can be viewed and plots can be generated before the simulation is resumed by selecting the Run action from the Analysis menu.
- **Stop:** Stops the entire simulation and enters the IDLE state

If the shell variable `noaskstopsim` is `OFF` (`false`), you will see a confirmation dialog:

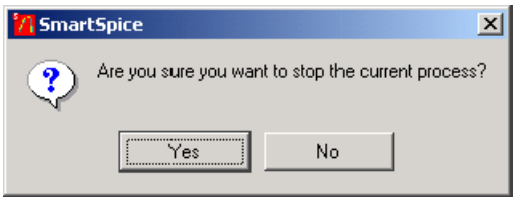


Figure 2-65 Stop Simulation - Confirmation Dialog

SmartSpice will stop the simulation if you press **YES**. To suppress the confirmation dialog, set the shell variable `NOASKSTOPSIM` to `TRUE`. This feature was created to prevent accidental stopping of the simulation process.

For more information, refer to the `skip`, `cancel`, `pause` and `cont` commands description in Chapter 5 Commands.

On the right side of the screen there are 2 additional buttons:

- **Take a Snapshot:** When pressed, the current contents of a screen will be dumped to a separate window and then saved to file if desired.
- **Options:** The drop-down menu contains a number of entries which will toggle the display of additional run-time sections if selected. These sections are shown below the primary statistics information in a view under the corresponding headers (see Figure 2-64). For details on a particular section description see Section 29.11 Run-time Statistics Display and Menu Options.

Two additional options are:

- **Engineering Format:** The format of units can be toggled between scientific and engineering representations.
- **Take a Snapshot:** Same as the previous but for a button.

2.18 Vectors Dialog

Simulation results can be accessed by selecting **Vectors** items from the **View** menu:

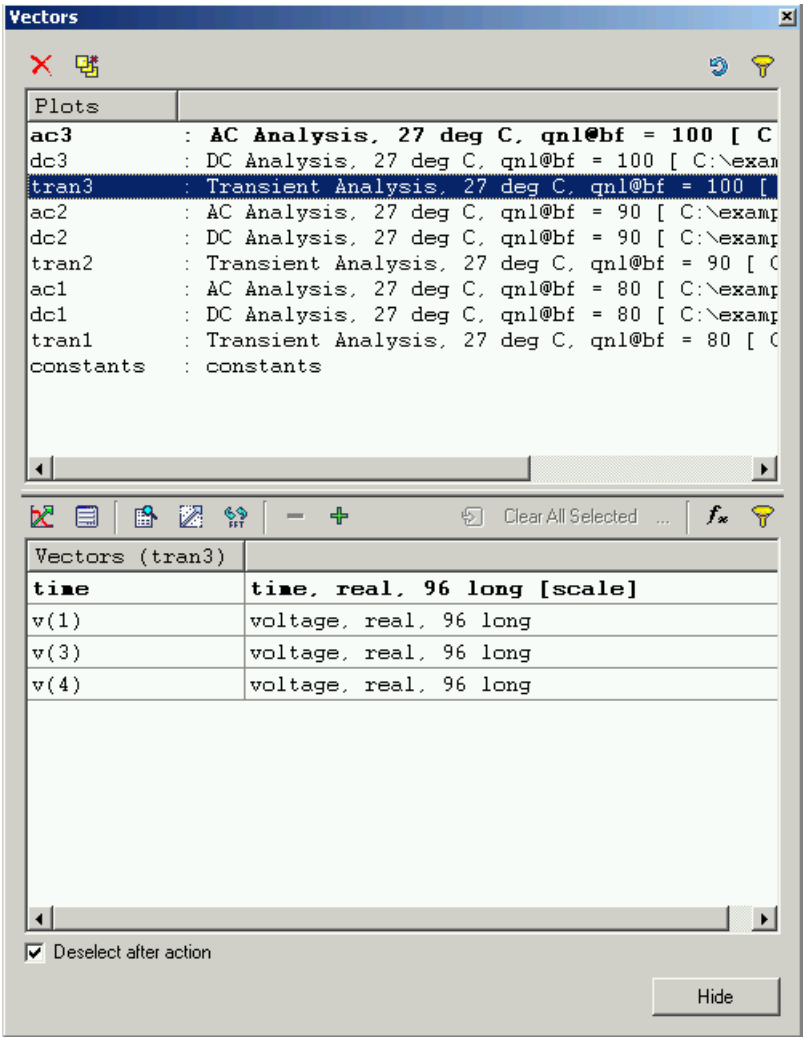


Figure 2-66 Vectors Window

The window is divided into two sections: **Plots** and **Vectors**.

2.18.1 Plots

The simulation results for each analysis are displayed in the **Plots** list view and stacked up in order of completion (most recent at the top).

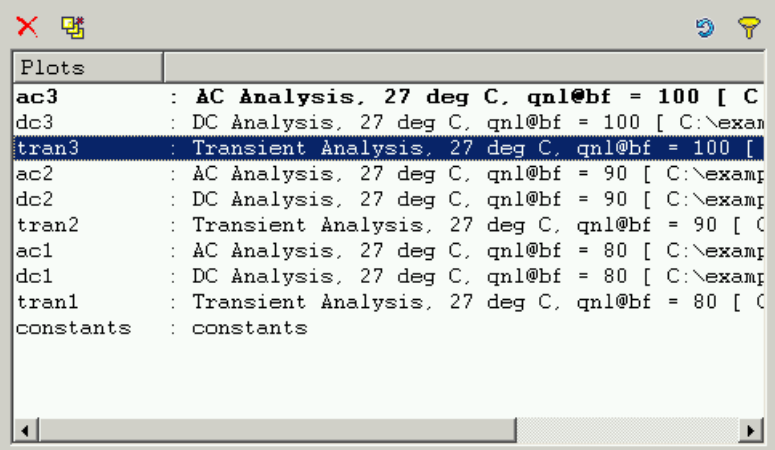


Figure 2-67 Plots View

This is a multiple selection list. Several items can be selected at a time and an action applied to all of them. There is a current plot that is highlighted in bold. Many SmartSpice post-processor commands operate on the vectors of the current plot. In addition, there is a focused item, the vectors of which are displayed in a **Vectors** section below.

You can select item by left-clicking on it and dragging for extended selection. Also you can use arrow keys (up and down) while holding **SHIFT**.

Every list item in the **Plots** viewcontains the following information:

- Short analysis type name containing the analysis type number (e.g. `tran4`).
- Full analysis type name (e.g., `Transient Analysis`).
- Parametric Information (e.g., temperature `20 °C`).
- Full input deck path.

The full list item text is displayed as a tool tip when the mouse is over each list item.

The control elements on a view (from left to right):

- **Delete:** Delete the currently selected plot(s) only.
- **Delete All:** Delete all analyses, except constants.
- **Refresh:** Reload all plots, refresh the lists.
- **Filter Plots:** If turned ON, the filter edit control will appear at the bottom of a view:

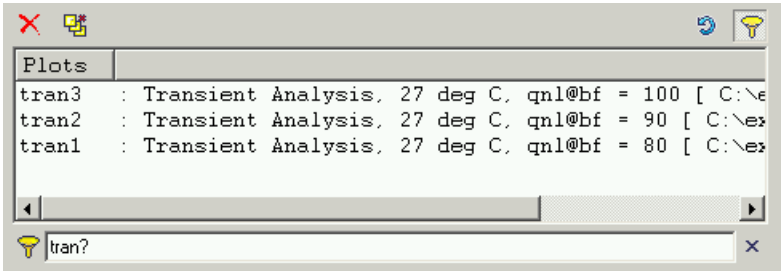


Figure 2-68 Plots View - Filter Plots

You can type in a pattern string to filter out the plots elements by name, using wildcard symbols (*|?). After it is close (by clicking **X** button on right, or typing **ESCAPE** while active), the Plots list view goes back to unfiltered state, displaying all elements.

In addition, you can right-click on a list view to popup a context menu:

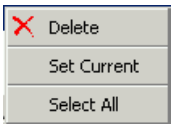


Figure 2-69 Plots View - Context Menu

- **Delete:** Delete the currently selected plot(s) only.
- **Set Current:** Toggle current plot (see setplot command).
- **Select All:** Select all items in a view.

2.18.2 Vectors

The **Vectors** section represents a multiple selection list of vectors. It always contains the vectors of the focused plot (see Section 2.18.1 Plots). The name of the focused plot is displayed on a list's view header (e.g., `Vectors(tran2)`).

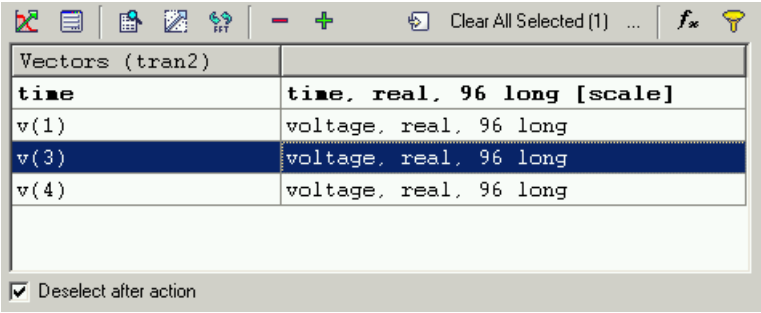


Figure 2-70 Vectors View

The items in the **Vectors** view displays the following information:

- Vector name (e.g., `i(vcc)`).
- Vector description, including unit (e.g., current), type (e.g., real), length, scale, etc.

The elements at the top of the view are:

- **Plot Vector(s):** Plot the currently selected Vectors to the waveform viewer.
- **Plot Settings:** Display the Plot Settings dialog (see Section 2.19 Plot Settings Dialog) to change the plot destination and chart options.
- **Display Values:** Prints the values of the selected vectors to the text viewer pane. It is the same as running the `print` command on a list of vectors.
- **Linearize selected Vector(s):** Force selected vectors of Transient analysis to conform to a linear scale (see the command “linearize” on page 615).
- **Perform FFT on selected Vector(s):** Perform a Fast Fourier Transformation (FFT) on selected vectors of Transient analysis (see the command “fft” on page 611).
- **Delete Vector(s):** Delete all selected vectors. The confirmation dialog will be shown to confirm deletion:

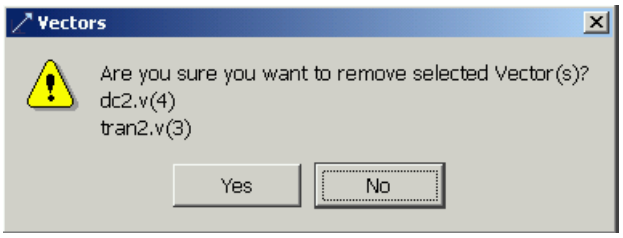


Figure 2-71 Delete Vectors Confirmation Dialog

- **Add Vector:** Click to create the post-simulation vector. This will popup an input dialog where you can specify a new vector's expression:

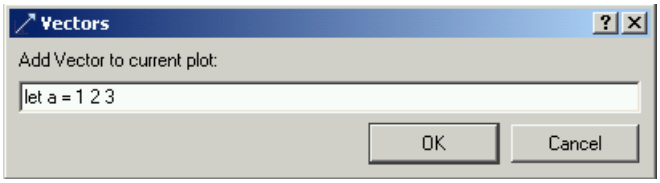


Figure 2-72 Add Vector Input Box

For more information, see the command “let” on page 615.

- **Clear Selected Vectors in Current Plot:** Deselect all selected vectors in the current plot only. This will not clear selection in plots, vectors of which, are not currently displayed.
- **Clear All Selected:** Deselect all selected vectors in all plots. This will reset to `0` the **Total selected Vectors** counter.
- **View Selected Vectors:** Opens a dialog to display all selected vectors in all plots. It is useful for reviewing selection:

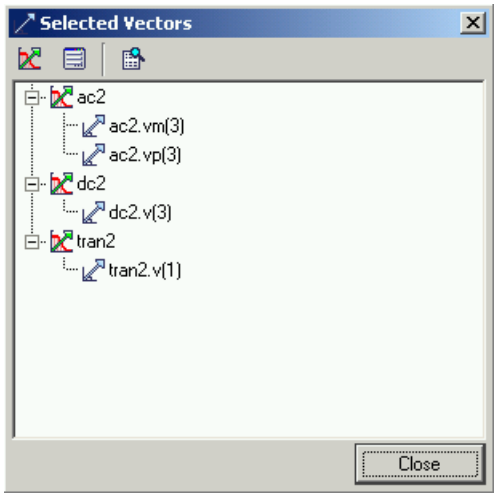


Figure 2-73 Selected Vectors Dialog

Several of most useful actions are available at the top of the list view. These are the same as described above.

- **Expression:** A vector expression can be used to create a post-simulation vector by entering an algebraic combination of output variables, constants, operations, pre-defined and user-defined macros.

Figure 2-74 Vectors Expression Box

- **Vectors Filter:** A long list of vectors can be filtered to provide quicker access to matching vectors by entering a filter pattern. Press **Enter** with the focus in the edit box to apply it:

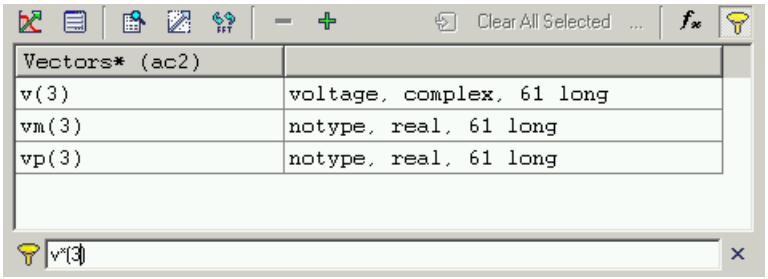


Figure 2-75 Vectors Filter

You can also make use of items of context menu, associated with the Vectors List:

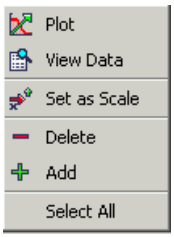


Figure 2-76 Vectors Context Menu

- **Plot:** Send selected vectors to the waveform viewer.
- **View Data:** Display contents of selected vectors in the text viewer.
- **Set as Scale:** Set the scale vector in the current plot. The scale vector is marked in the Vectors list by `[scale]` at the end of the list item. It is also highlighted in bold. The scale vector can also be toggled by double-click on an item. For more information, see the command “setscale” on page 629.
- **Delete:** Delete the selected vector(s) in the current plot.
- **Add:** Add a new vector (see above).
- **Select All:** Select all vectors in a list view.

2.19 Plot Settings Dialog

When plotting the selected vectors to the Waveform Viewer, the destination and chart options can be displayed and changed by clicking the **Settings** button on the **Vectors** dialog:

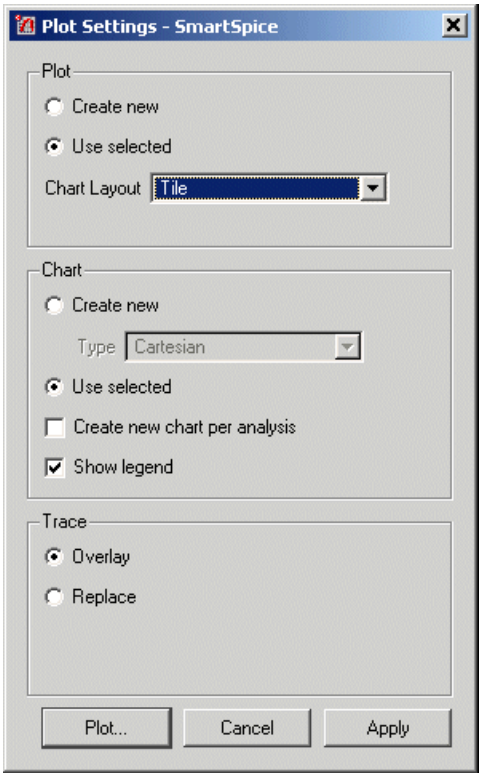


Figure 2-77 Plot Settings Dialog

This dialog comprises several sections:

2.19.1 Plot

- **Create new:** Create a new window in the waveform viewer.
- **Use selected:** Use the existing selected window in the waveform viewer.
- **Chart Layout:** Display the charts using the specified layout.

2.19.2 Chart

- **Create new:** Create a new **Chart** in the waveform viewer.
- **Use selected:** Use the existing selected chart in the Waveform Viewer. If multiple charts have been selected in the Waveform Viewer, the first selected chart reported will be used.
- **Show Legend:** Display a legend on the chart.

2.19.3 Trace

These options are only available when plotting to an existing selected chart:

- **Overlay:** Overlay the selected vector trace(s) onto the existing selected chart.
- **Replace:** Replace the existing vectors trace(s) on the existing selected chart with the selected vector trace(s).

2.20 Statistics Window

All resource usage statistics can be displayed by selecting **Statistics** from the **View→Circuit** menu:

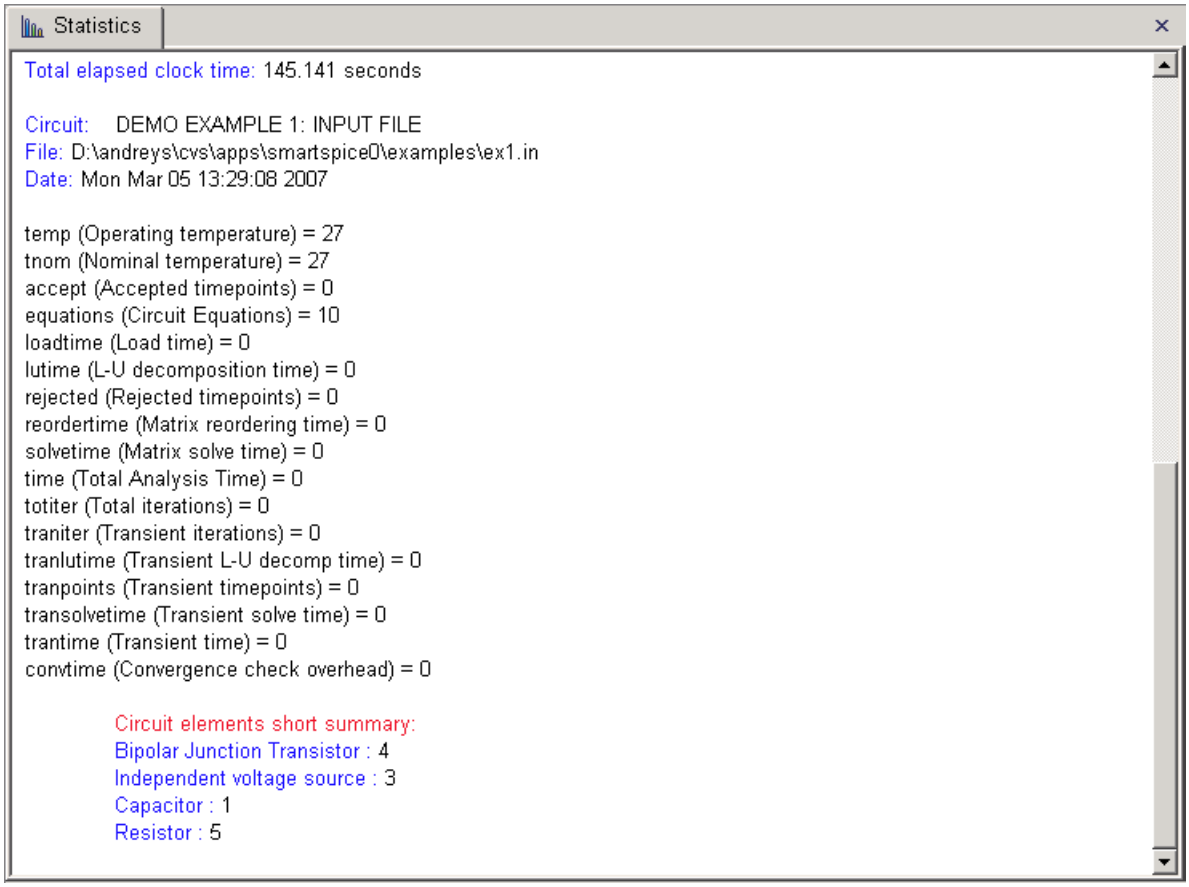


Figure 2-78 Statistics Dialog

2.21 Model and Device Views

Device and model views for the current circuit can be opened from the **View→Circuit** menu item. The left part displays the list of devices/models, and the right are parameters for the selected device/model:

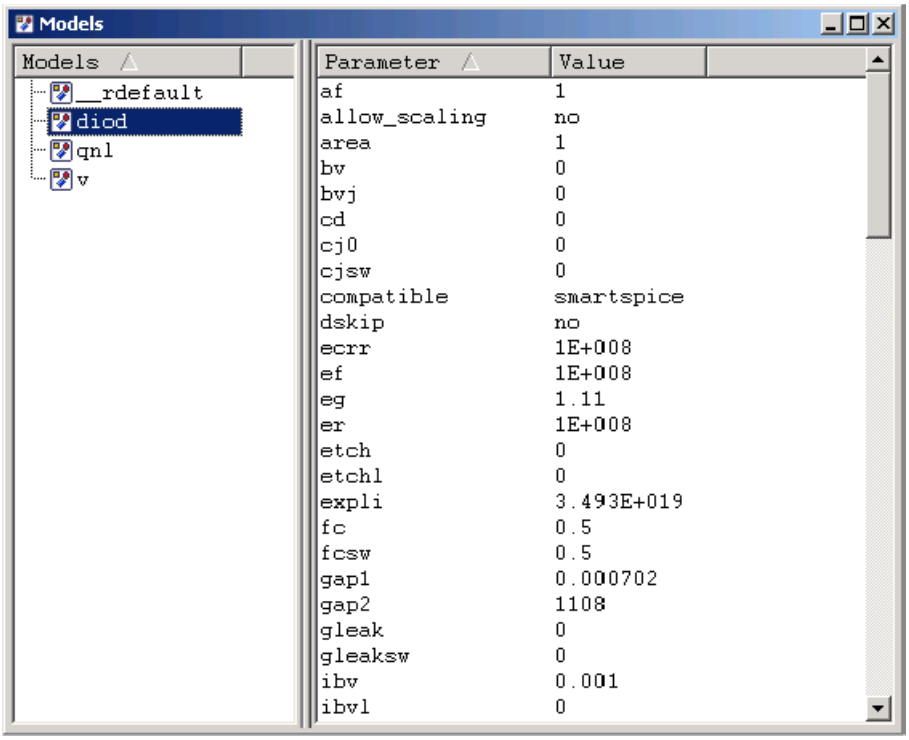


Figure 2-79 Models View

Devices View in the Flat Mode

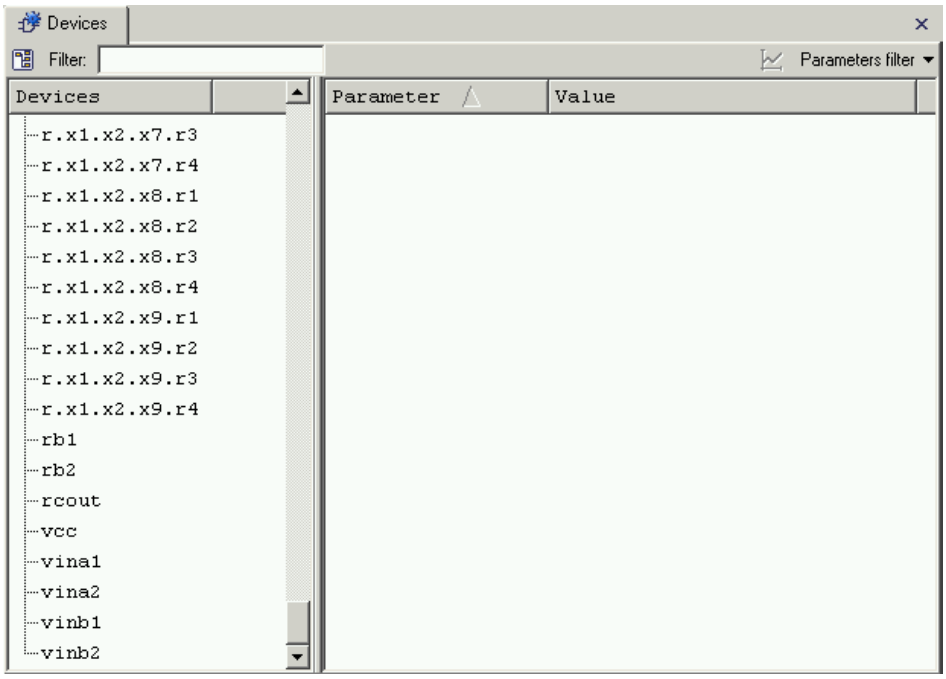


Figure 2-80 Flat Mode View

The **Devices** list contains the view mode switch button.

- **Devices:** Tree-like representation of devices list:

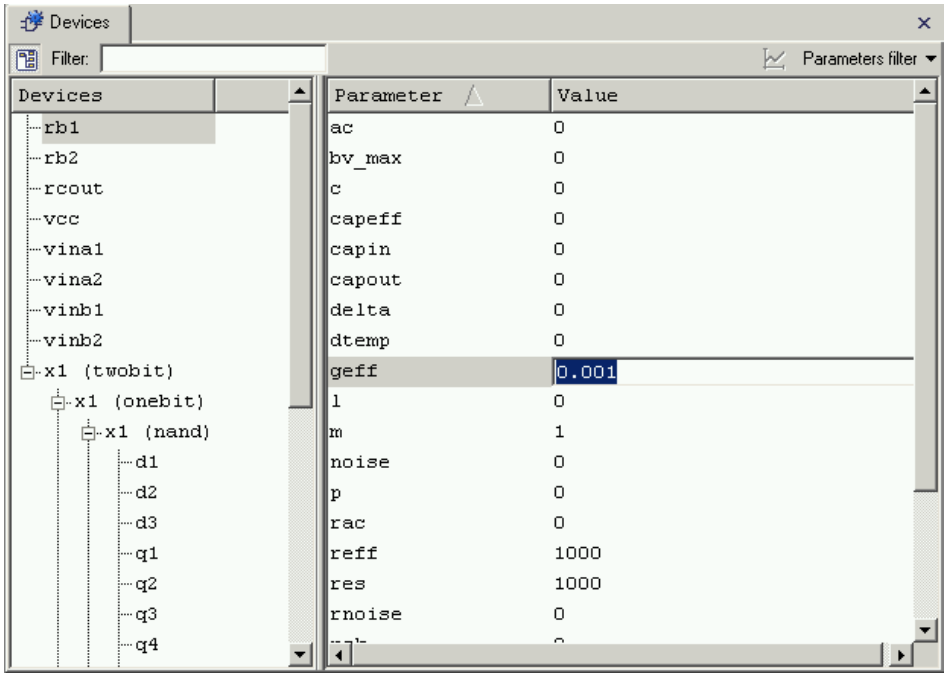


Figure 2-81 Tree Mode View

Also, the list of devices can be filtered through a user defined template typed in the **Filter** edit box at the top of the window. The regular expression capabilities can be used if the **{*|?}** button is set to ON.

2.22 Node List Window

The summary of node table connections for the current circuit can be displayed by selecting **Display→Circuit→Node List** menu:

Figure 2-82 Node List Window

2.23 Preferences

User properties that are stored between application sessions can be accessed using the **Preferences** dialog by selecting **Preferences** from the **Edit** menu:

- **OK:** Store settings and close the dialog.
- **Apply:** Store settings without closing the dialog.
- **Cancel:** Close the dialog.

2.23.1 Manage Preferences Panel

The properties can be imported/exported/reset using the **Manage Preferences** panel:

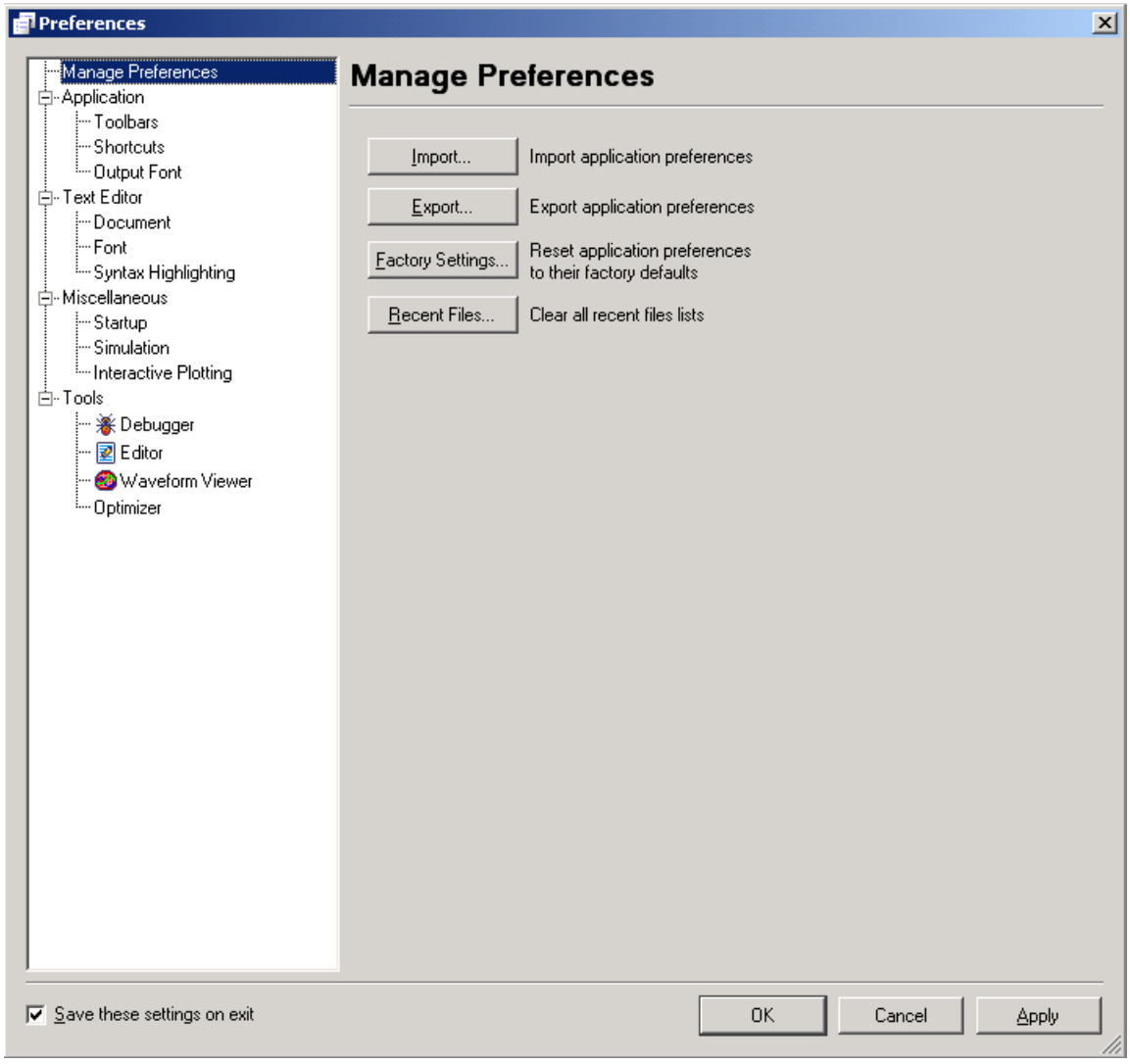


Figure 2-83 Manage Preferences Panel

- **Import:** Read the application's properties from a previously exported properties file.
- **Export:** Write the application's properties to a properties file.
- **Factory Settings:** Reset all properties to their default values.

2.23.2 Output Font Panel

The font attributes for certain text output (e.g., the Main window's output panel) can be specified:

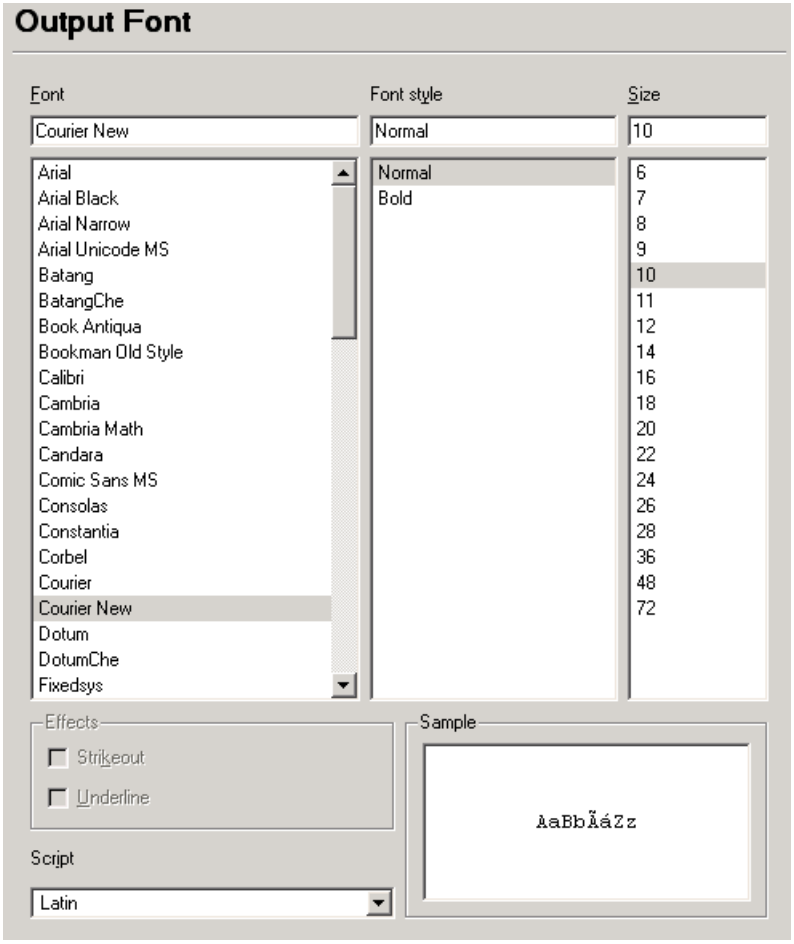


Figure 2-84 Output Font Panel

The sample text will display example text using the specified font attributes.

2.23.3 Toolbars Panel

Each toolbar can be displayed and customized to provide fast mouse access to menu actions:

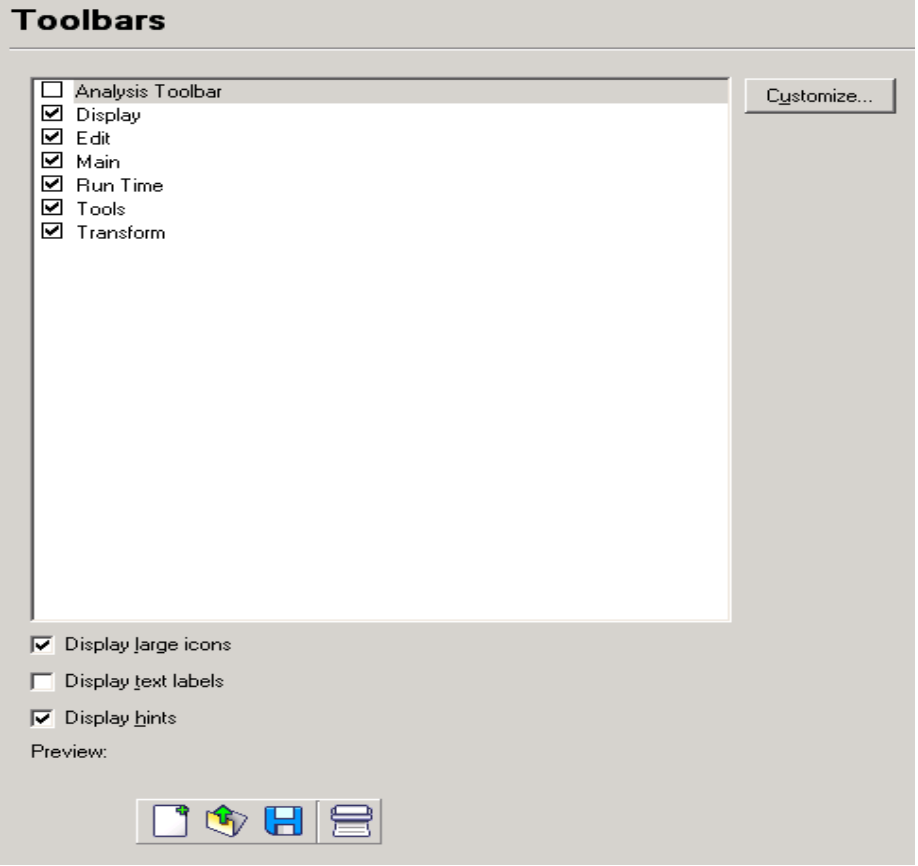


Figure 2-85 Toolbars Panel

- **Displaying a Toolbar:** Check a toolbar (e.g., Main Toolbar) to display it.
- **Display large icons:** Check to display large icons in the toolbars.
- **Display text labels:** Check to display text labels underneath each icon.
- **Display hints:** Check to briefly display the icon's tool tip label when the mouse moves over the icon.

2.23.4 Customizing a Toolbar

Click **Customize** to display the **Customize Toolbar** dialog:

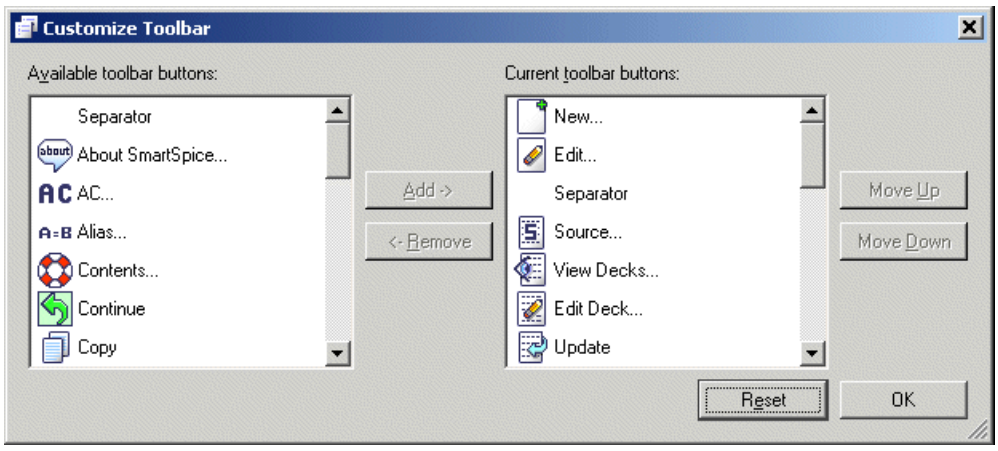


Figure 2-86 Customize Toolbar Dialog

- **Adding a Toolbar button:** Select the toolbar button from the **Available toolbar button** list and click **Add** to add it to the **Current toolbar buttons** list.
- **Ordering a Toolbar button:** Select the toolbar button from the **Current toolbar button** list and click **Move Up** or **Move Down** to move the toolbar button position up (left on a horizontal toolbar) or down (right on a horizontal toolbar) respectively.
- **Removing a Toolbar button:** Select the toolbar button from the **Current toolbar button** list and click **Remove** to remove it from the **Current toolbar buttons** list.
- **Reset:** Click to reset the toolbar to its original state prior to the recent changes.
- **OK:** Closes the dialog.

2.23.5 Shortcuts Panel

Each menu action can have shortcut keys to provide fast keyboard access:

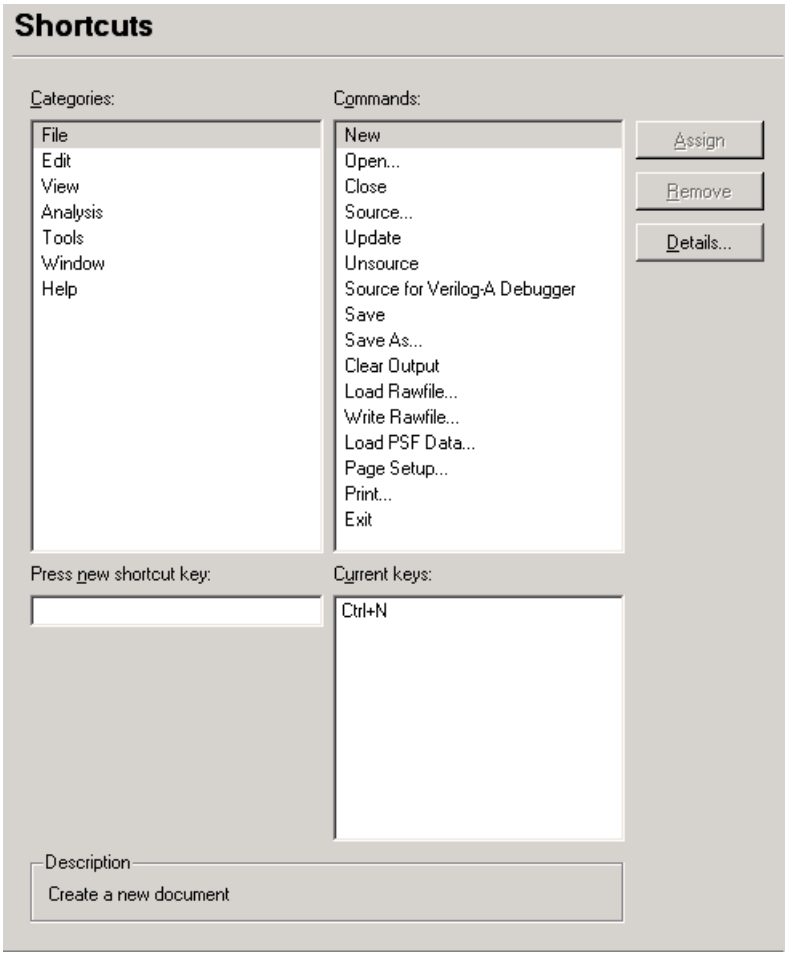


Figure 2-87 Shortcuts Panel

Select the menu action by clicking on the appropriate **Category** and **Command**, then:

- **Assigning a Shortcut:** Click on the new shortcut key control and press the shortcut key combination (e.g., **Ctrl N**), then click the **Assign** button.
- **Removing a Shortcut:** Select the existing shortcut from the **Current** keys and click the **Remove** button.

2.23.6 Startup

The following preferences are applied when SmartSpice is started:

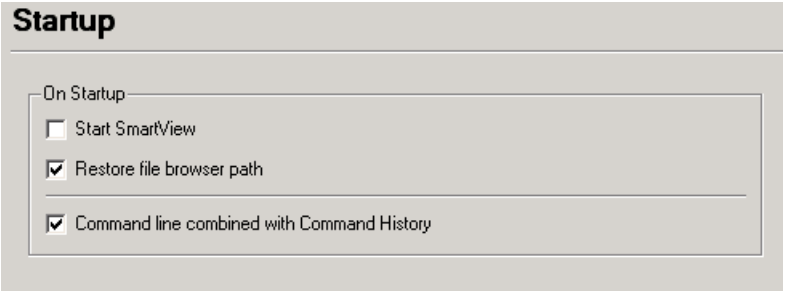


Figure 2-88 Startup Panel

- **Start SmartView:** Automatically start the Waveform Viewer once SmartSpice has started. Used to optimize the response time between SmartSpice and the Waveform Viewer during a plot/interactive plot by removing the small delay that occurs while waiting for the Waveform Viewer to start.
- **Restore file browser path:** Use the last browsed directory as the initial file browser path when SmartSpice is started. If this is unchecked, the current working directory will be used.

2.23.7 Simulation

The following preferences are applied during a simulation:

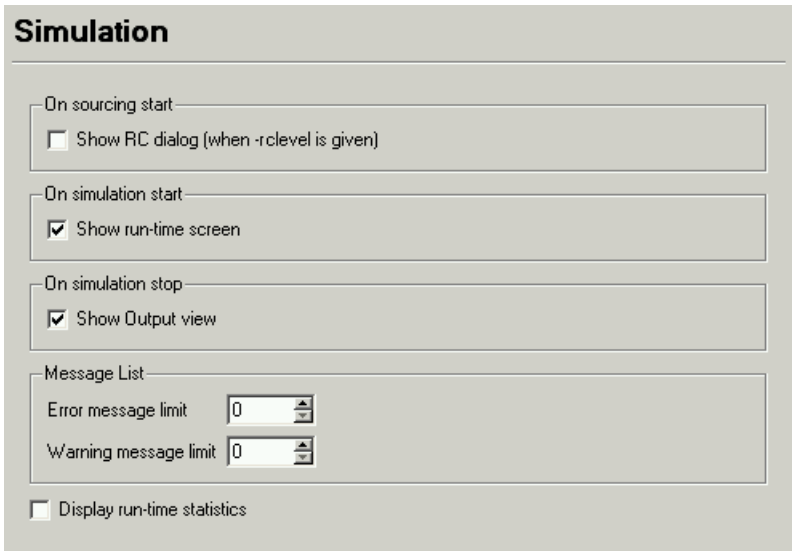


Figure 2-89 Simulation Panel

- **On simulation start:** Set ON if you wish to show Run-Time screen when denoted event occurs.
- **On simulation stop:** Set ON if you wish to show the Output view after simulation stops.
- **Message List:** Set limit on a number of displayed warnings and errors in the ‘Messages’ view during deck’s sourcing or simulation.

2.23.8 Interactive Plotting

The following preferences are applied during an interactive plot:

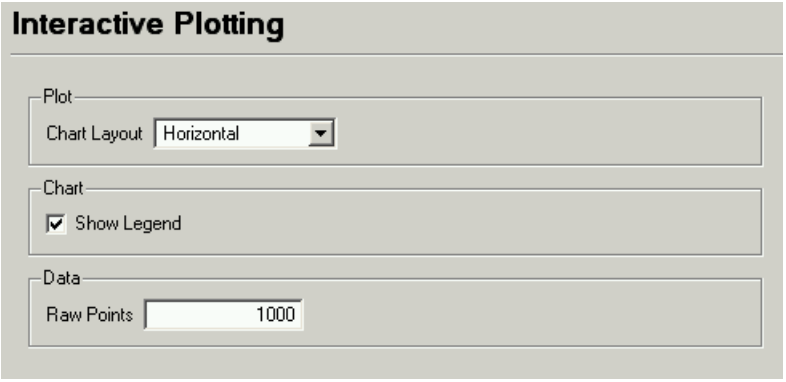


Figure 2-90 Interactive Plotting Panel

- **Chart Layout:** Display the interactive plot charts in the specified layout.
- **Show Legend:** Display the legend on each chart.
- **Raw Points:** Transfer the interactive plot data every N number of data points. Used to optimize the performance between SmartSpice and the waveform viewer during an interactive plot.

2.23.9 Editor

The following settings are used when SmartSpice starts a text editor:

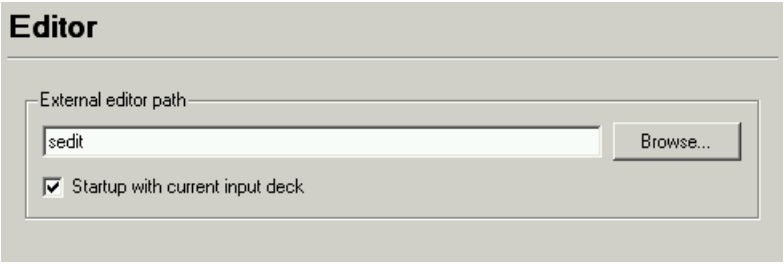


Figure 2-91 Editor Panel

- **Editor:** Path and executable name of the alternative text editor.
- **Browse:** Click to display a file browser dialog for selecting the alternative text editor.

2.23.10 Waveform Viewer

The following settings are used to initialize the SmartSpice communications with the Waveform Viewer:

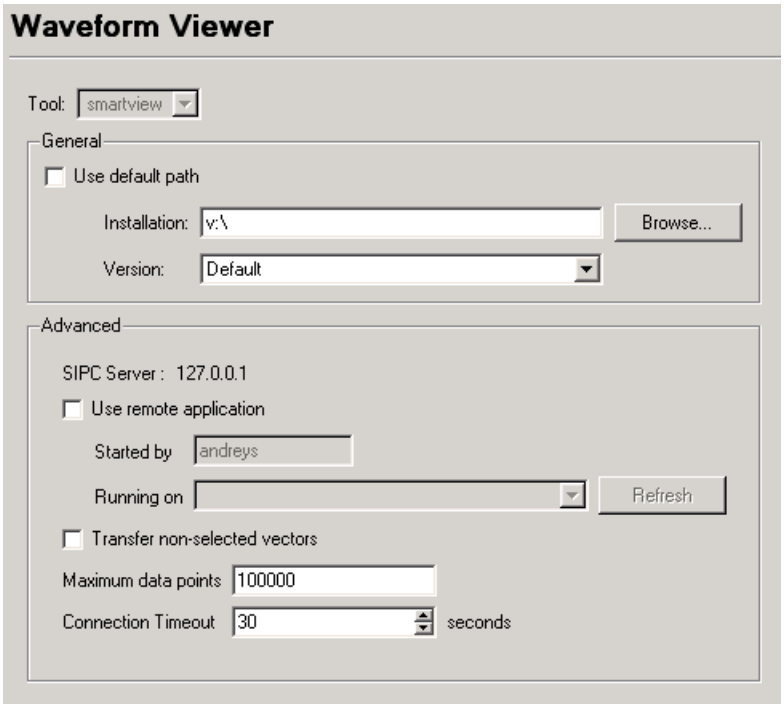


Figure 2-92 Waveform Viewer Panel

- ***Installation:** Displays the current Waveform Viewer path. A valid Waveform Viewer path will be the name of the directory that contains the Silvaco application `bin` and `lib` directories.
- ***Browse:** Opens a directory browser for selecting an alternative Waveform Viewer path.
- **Reset:** Resets the Waveform Viewer path to the default value (the `$SILVACO` environment variable value).
- **Version:** Lists compatible Waveform Viewers found in the Waveform Viewer path. Also accepts version format (`x.y.z.L`) for specifying unlisted versions (for example, when connecting to a Waveform Viewer running on a remote machine).
- **SIPC Server:** The name of the Silvaco Inter Process Communication (SIPC) Server which is being used to facilitate the SmartSpice/Waveform Viewer communications
- ****Use remote application:** Used to identify and locate a waveform viewer application running on a remote machine or using alternative credentials.
- ****Started by:** The name of the user that started the Waveform Viewer.
- ****Running on:** The name of the machine that the Waveform Viewer is running on.
- **Transfer non-selected vectors:** When ticked, pass vector data for all vectors from all analyses that have selected vectors. By default, this is unchecked so only the selected vectors' data is passed to the Waveform Viewer.
- ***Maximum data points:** Determines the maximum number of data points that can be transferred to the waveform viewer in each data segment.

* Please contact your System Administrator before changing these settings.

** These settings may not be supported by all waveform viewer versions.

2.24 Simulator Views

There are 3 views to display/modify simulator-level variables, macros and states, which you can access from **View→Simulator** main menu:

Variables View

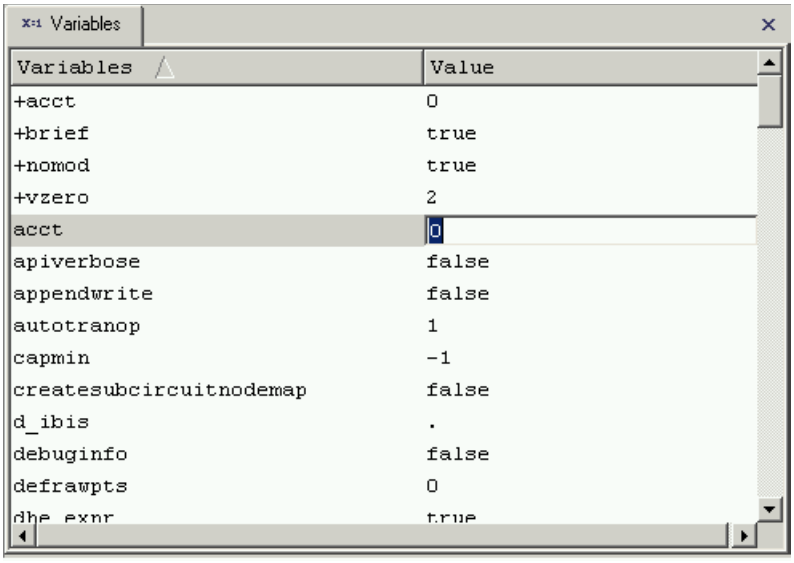


Figure 2-93 Variables View

Defines View

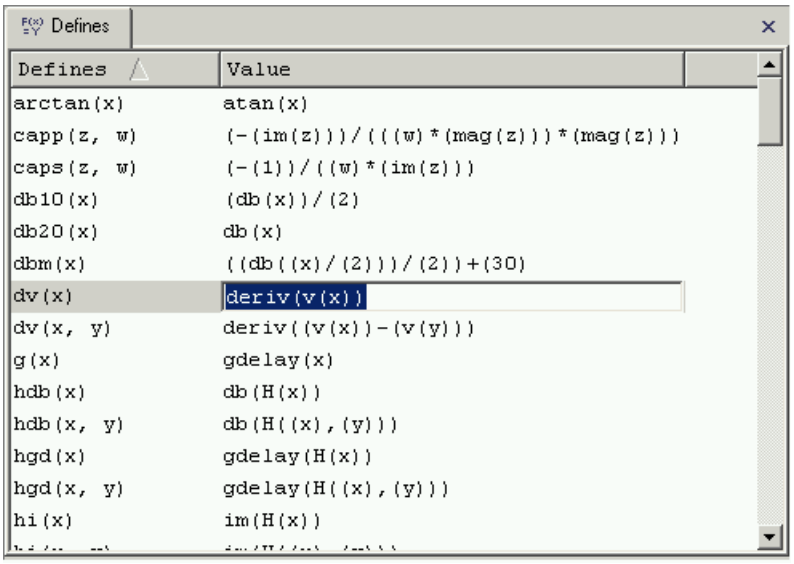


Figure 2-94 Define view

Alias View

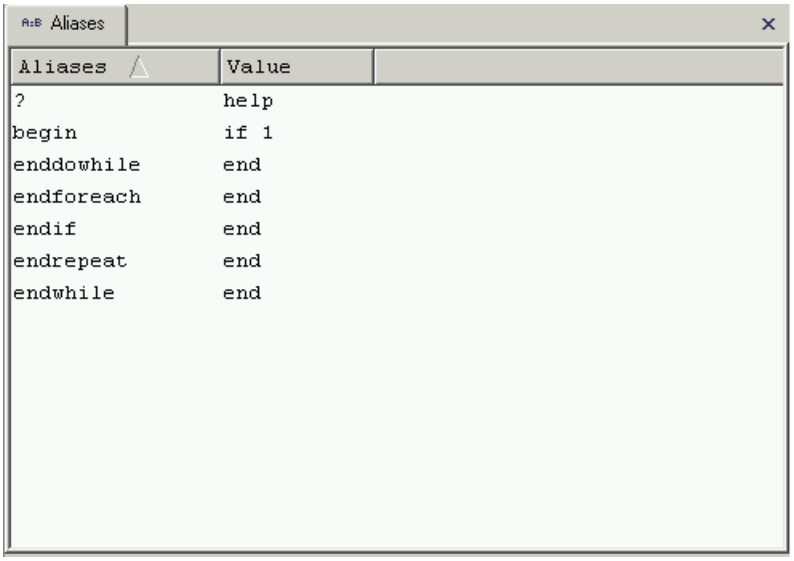


Figure 2-95 Alias View

The views are represented as two-column single selection lists. The first column displays the names of the items and the second (their values). You can modify an item's value by selecting the item (use the arrow keys) and then press **Enter**. If the item is editable, the in-place line editor will appear where you can type a new value. Press **Enter** again to make the change permanent.

For more information, refer to the `set` command description in Chapter 5 Commands.

2.25 Expression Wizard

This wizard helps to create an expression from a list of available functions and vectors.

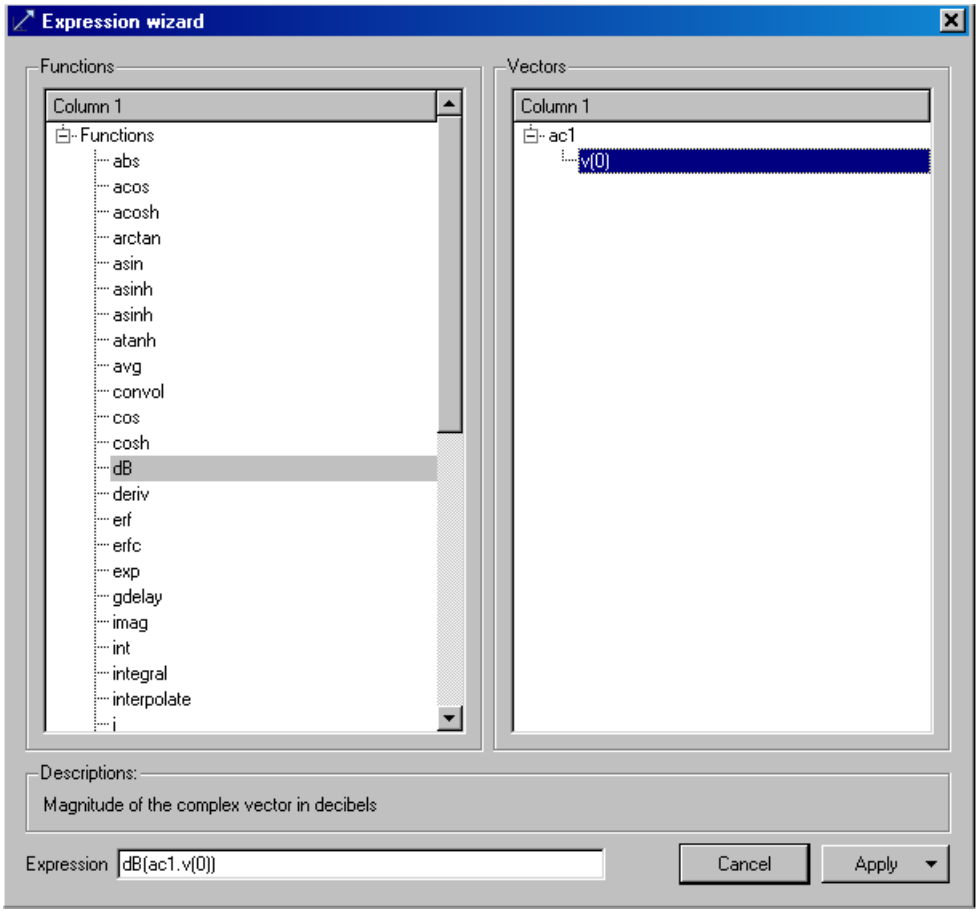
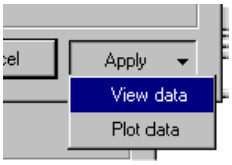


Figure 2-96 Expression Wizard

To create expressions, choose a vector at the right panel and a function at the left. Two actions are available under the expression:

- **View data:** Opens a viewer with textual table of expression vectors values.
- **Plot data:** Launches SmartView with graphical representation of the expression vectors.



The expression window is available from the vector view form.

2.26 Help

For more information, the following SmartSpice help documentation can be accessed from the Help menu:

- Contents
- SPICE MODELS MANUAL

2.27 Contacting Silvaco

Contact details for Silvaco can be found on the **About** dialog if you have any specific questions or comments not covered in this or other related documentation (see Section 2.26 Help):

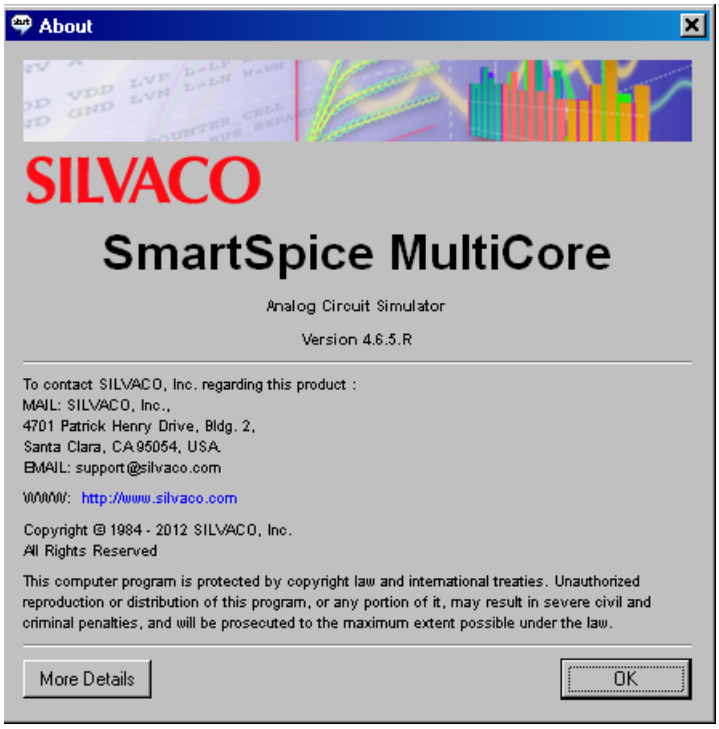


Figure 2-97 About Dialog

- **More Details:** Click to display library version information that may be requested by Technical Support.
- **OK:** Closes the dialog.

2.28 3D-plotting for Parametric Analyses

SmartSpice is able to display the simulation data, generated by parametric analysis, as a 3-dimensional surface. This feature is available from **Tasks** browser window and two tools serve this purpose - Tonyplot and Tonyplot3D.

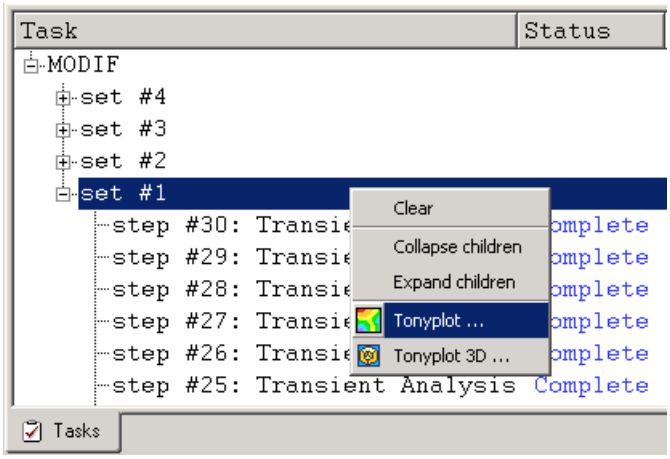


Figure 2-98 3D-plotting with Tonyplot

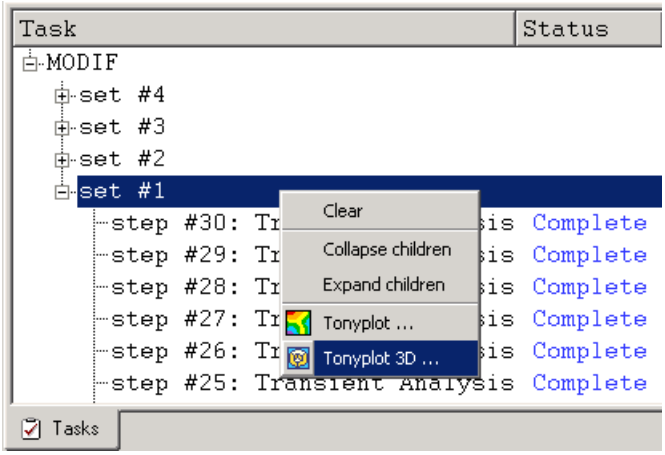


Figure 2-99 3D-plotting with Tonyplot3D

Simulate an input deck, containing the `.MODIF` statement. After simulation is done the **Tasks** view will display a hierarchy structure of plots determined by `sets` and `steps` nodes. Right-click on a particular `set` node in order to display the context menu and make it a starting point in specifying a 3D-plot configuration. Because there is no limitation on how many nested levels `.MODIF` may contain, the hierarchy with more than 1 set can be considered as having extra dimensions which must be fixed at some point. The 3D-plot configuration dialog comes next and may contain an **Extra** section (not displayed here) from where the fixed values for additional dimensions must be specified.

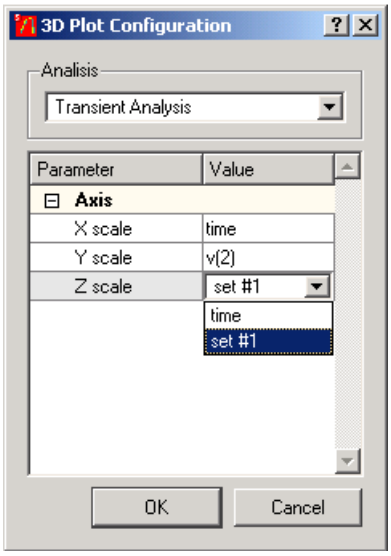


Figure 2-100 3D Plot Configuration Dialog

The dialog above allows to choose between scales for X and Z axises and the Y for the function. Also, if there are multiple analyses in the input deck, the required analysis from the combo box is at the top. Press **OK**.

Depending on what tool has been selected, SmartSpice launches Tonyplot or Tonyplot3D. By default, Tonyplot doesn't display the 3D plot at once. Therefore, go to the **Plot→Display** menu to bring forth the **Tonyplot Display** dialog:

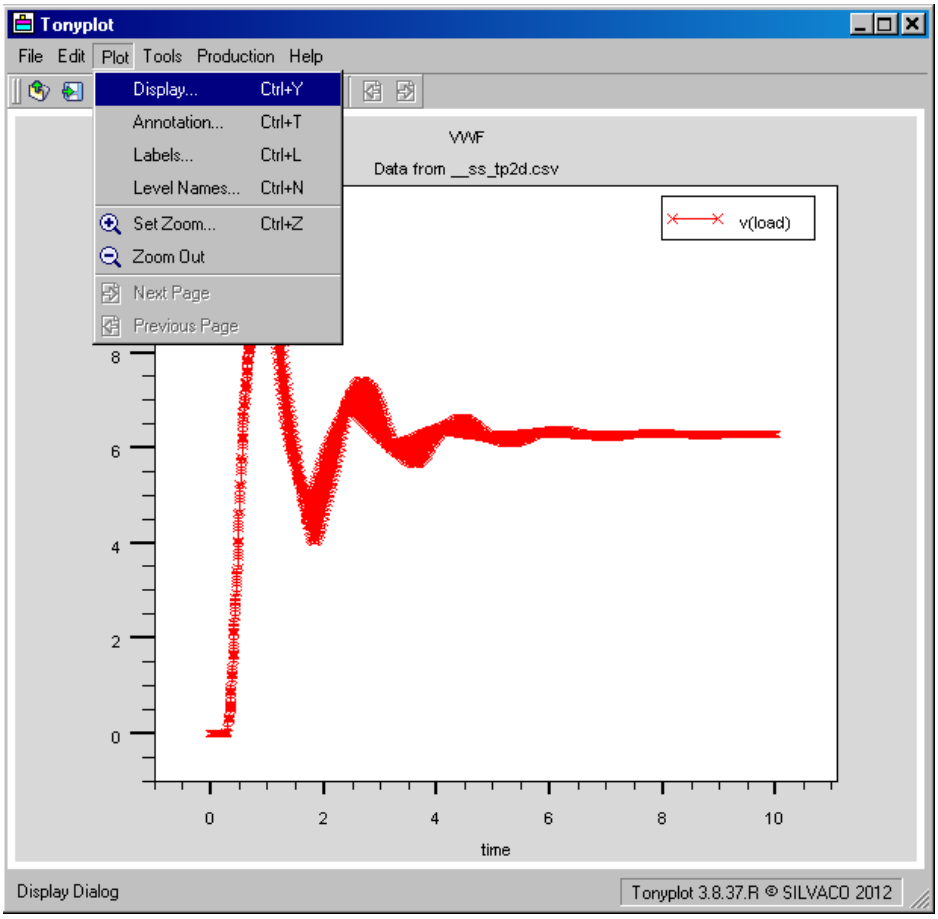


Figure 2-101 Tonyplot

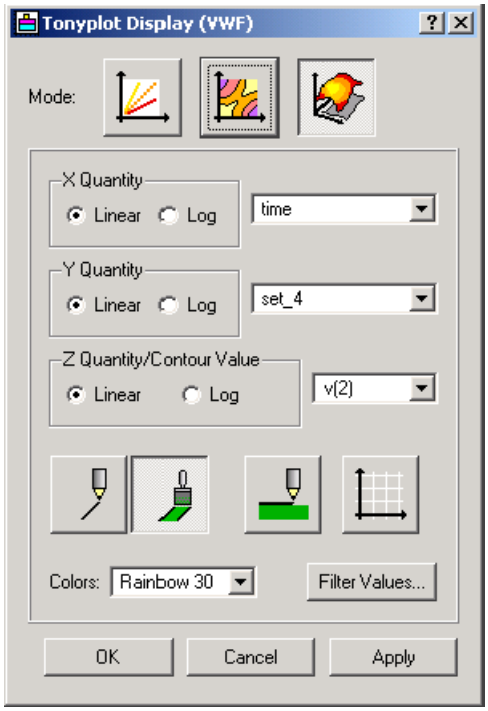


Figure 2-102 Tonyplot Display Dialog

Select the 3D-surface mode (the rightmost button at the top) and press **OK** or **Apply**.

Tonyplot will display your 3D plot:

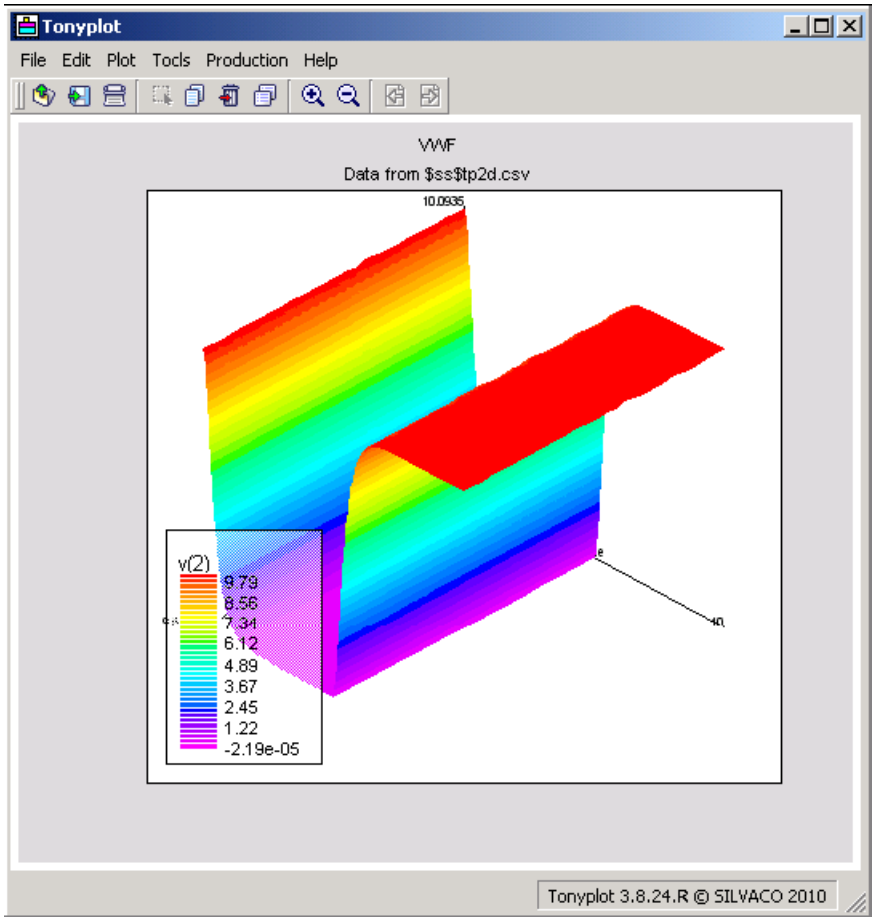


Figure 2-103 Tonyplot Displays the .MODIF Data

No additional user interaction is required for 3D-plotting with Tonyplot3D. It's displayed at startup:

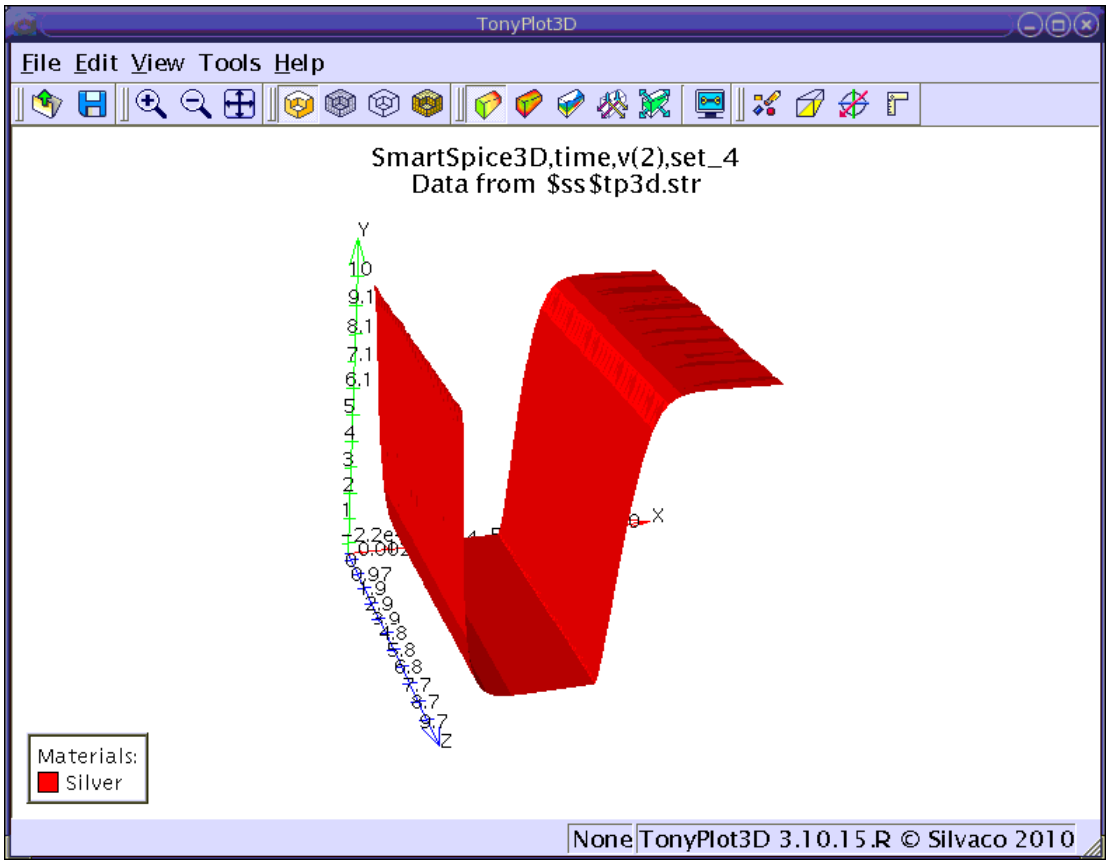


Figure 2-104 Tonyplot3D Displays the .MODIF Data

Also, Tonyplot and Tonyplot3D are available from SmartSpice's main menu **Tools** as external tools. These can be used to open the saved files of appropriate format:

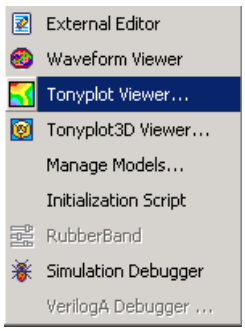


Figure 2-105 Tools Menu (SmartSpice)

Chapter 3
Statements

3.1 Introduction

SmartSpice simulates circuits based on user-created input files. These files contain the analyses to be performed, and define element values and models for devices in the circuit and all subcircuits. The input file also contains a set of analysis statements, a set of output statements, and a set of control statements. Analysis, output, and control statements define what SmartSpice will do with the described circuit.

The first line of the input file is the .TITLE statement that contains text. The last statement must be a .END statement. The remaining line order is arbitrary, with a few exceptions (such as the .ALTER and .PARAM statements).

This chapter explains circuit description, procedures on the simulation, output, and control statement specifications.

3.1.1 Circuit Description

A circuit is described in SmartSpice as an input file composed of a set of statements. These statements define circuit elements and nodes, specify models for any devices, and describe subcircuits. The input file contains the descriptions of more than one circuit. When two or more circuits are specified between the title line and the first .END statement, SmartSpice will simulate all circuits simultaneously.

3.1.2 Nodes

Node names can be arbitrary character strings consisting of alphanumeric characters and special symbols: . ! <> - + / * ^ & % # _ : @

The ground node is always called 0, GND, GND!, or GROUND. SmartSpice considers node names as characters rather than numerical values: node 1 and node 01 are treated as distinct and separate nodes.

A circuit being simulated cannot contain a loop of voltage sources, a loop of inductors, or a cluster of current sources.

If a DC analysis is specified, each node in the circuit must have a DC path to ground. Otherwise, SmartSpice connects this node to the ground through a large resistor. The value of this resistor is set using the option DCPATH.

Each node must have at least two connections, except transmission line nodes (to allow unterminated transmission lines).

SmartSpice allows the use of subcircuit external node names in the main circuit device definitions.

Example

```
.SUBCKT example Input Output
R1 Input Output R=1k
.ENDS
*** Main Circuit
X1 2 0 example
Cload X1.Input 0 C=1pF
```

An input file can also contain a .GLOBAL statement. All nodes specified in this statement are directly available in all subcircuits. The .GLOBAL statement allows all specified nodes to override local subcircuit definitions. Global nodes are generally used for power supplies and clocks, especially in larger circuits.

Note: Nodes and devices can have the same names, for example: `VEE VEE 0 DC 1` [1].

This statement specifies a voltage source `VEE` connected between node `VEE` and node `0` with a DC value of `1V`.

3.1.3 Elements

Each device in the circuit is described by an element statement. The element statement contains the element name, the circuit nodes to which the element is connected, and the values of the element parameters. The first letter of an element name specifies the type of element to be simulated. For example, a resistor name must begin with the letter `R`, and can contain one or more characters. This means that `R`, `R1`, `RSE`, `ROUT`, and `R3AC2ZY` are all valid resistor names.

Some elements, such as diodes and transistors, must always refer to a model. A set of elements can refer to the same model. For elements, such as resistors and capacitors, model referencing is optional.

Each element type has its own set of parameters. For example, a resistor statement can specify a resistance value after the optional model name. The bipolar transistor statement can specify an area parameter. All parameters have corresponding default values. Independent voltage and current sources have different specifications for transient, DC, and AC phases of simulation. DC parameters are specified by using the parameters `DC`, `DC1`, `DC2`, and `PWLDC`. Transient specifications use the parameters `EXP`, `PULSE`, `PWL`, `PWLFILE`, `PWLFILEDESC`, `SFFM`, and `SIN`. AC parameters start with the parameter `AC`.

3.1.4 Models

SmartSpice semiconductor device models require more parameter values than are listed under each statement. Often, multiple devices in a circuit are defined by the same set of device model parameters. For this reason, a set of device model parameters are defined by a separate `.MODEL` statement, and assigned a unique model name. The device element statement in SmartSpice refers to the model name, and eliminates the need to specify all model parameters on each device element statement. SmartSpice offers built-in models for many devices. For a complete description of all SmartSpice models, refer to the SPICE Models Manual.

3.1.5 Subcircuits

In most cases, a large circuit will contain many subcircuits of the same type in various locations within the circuit. It is most efficient to describe the subcircuit separately, and instantiate the subcircuit in the main circuit as many times as required. SmartSpice makes this possible by means of the `.SUBCKT` statement. A subcircuit definition begins with this statement, which specifies the name of the subcircuit and the names of the subcircuit terminals. The `.ENDS` statement indicates the end of the subcircuit description. All subcircuit elements and local model definitions are included in the subcircuit description. SmartSpice automatically inserts this group of elements whenever the subcircuit is referenced. There is no limit on the size or complexity of the subcircuits, and subcircuits can be nested.

1. Device names take precedence over node names in expressions. By convention, device names are taken to refer to the positive node of the device. Therefore, v(VEE) refers to the voltage at the positive node of the device VEE. If you specify `VEE 0 VEE DC 1`, where the name of the negative node is the same as that of the device, the expression v(VEE) will give the voltage at the positive node of the device VEE, which is node 0 (GND). This is rarely what is desired.

Note: Any model definition and any element included as part of a subcircuit definition is local and must be referenced outside the subcircuit definition using its expanded name (see Section X (Subcircuit Call)). Also, any internal nodes (if not included on the .SUBCKT statement line) are local, with the exception of global node 0 and nodes defined as global nodes in the .GLOBAL statement.

Subcircuits are instantiated within the circuit description file by using pseudo elements beginning with the letter X, followed by the actual circuit nodes used in expanding the subcircuit.

Example

```
X1 8 13 21 SUB1
.SUBCKT SUB1 1 2 3
```

The formal terminal names 1, 2, and 3 of the subcircuit definition SUB1 are replaced by the actual circuit node names 8, 13, and 21 respectively, when expanded.

3.1.6 Warnings Generated for Duplicate Subcircuits

SmartSpice runs a check on all subcircuits in a deck, and issues a warning when two subcircuits at the same hierarchical level have the same name. As before, SmartSpice will use the last definition. In addition, an error message is issued when the two definitions have different numbers of arguments. In this case, the simulation is not aborted.

Example

The following input deck fragment generates several warnings and an error message when sourced:

```
X1 mid out inv nl=1.2 pl=1.4 vbb=0.0

.SUBCKT inv input output nl=1.3 pl=1.3 vbb=-1
M1 output input GND vbias nmod W=nw L=nl
M2 output input vdd vdd pmod W=pw L=pl
Cload output 0 'cap'
Vb vbias 0 dc 'vbb'
.ENDS inv

.SUBCKT inv1 input nl=1.3 pl=1.3 vbb=-1
r1 input output 1k
.ENDS inv1
.SUBCKT inv1 input nl=1.3 pl=1.3 vbb=-1
r1 input output 2k
.ENDS inv1

.SUBCKT inv2 input output nl=1.3 pl=1.3 vbb=-1
r1 input output 1k
.ENDS inv2
.SUBCKT inv2 input nl=1.3 pl=1.3 vbb=-1
r1 input output 2k
.ENDS inv2

.SUBCKT inv3 input nl=1.3 pl=1.3 vbb=-1
r1 input output 1k
.SUBCKT inv3 input output nl=1.3 pl=1.3 vbb=-1
r1 input output 1k
```

```
.ENDS inv3
.ENDS inv3
```

The subcircuits named `inv1` will generate a warning because they are at the same hierarchical level. In this case, SmartSpice will simply use the last definition that occurred in the input deck.

The subcircuits named `inv2` will also generate a warning and an error message because they have different numbers of nodes for each instance, and therefore are not compatible.

The subcircuits named `inv3` generate neither warnings nor errors, even though they share the same name, and have different numbers of nodes. This occurs because one is nested inside the other, and therefore they occur at different levels of the hierarchy.

The messages produced by SmartSpice are shown below:

```
Warning: Two subcircuit definitions, with same name (inv2), at same
hierarchical level.
  51 : .SUBCKT inv2 input output nl=1.3 pl=1.3 vbb=-1
  55 : .SUBCKT inv2 input nl=1.3 pl=1.3 vbb=-1

Error: Incompatible definitions of subcircuit 'inv2'.
Different number of nodes in each definition.

Warning: Two subcircuit definitions, with same name (inv1), at same
hierarchical level.
  41 : .SUBCKT inv1 input nl=1.3 pl=1.3 vbb=-1
  45 : .SUBCKT inv1 input nl=1.3 pl=1.3 vbb=-1

Warning: Using most recent definition of subcircuit 'inv1'.
```

3.2 Analyses

3.2.1 Transient Analysis

Transient analysis computes circuit behavior over a specified time interval, beginning at time zero. This is the most frequently used and most universal analysis in SmartSpice. During a transient analysis, some or all of the independent voltage and current sources have time dependent values. Initial transient source values override the DC values at time zero. SmartSpice first calculates a DC operating point, and then performs the Transient analysis.

Transient analysis can bypass the DC operating point calculation if the `.TRAN` statement contains the parameter `UIC` (use initial conditions). The Transient analysis begins from the initial conditions specified in the `.IC` statements, or by the `IC` parameters on the various elements. Also, if the parameter `TRANOP` is specified, SmartSpice first computes and automatically saves the transient operating point, starting from zero initial conditions, and performs the Transient analysis. In these two cases, SmartSpice is free of any “No DC Path to Ground” problems.

If a `.TRAN` statement contains the parameters `CALLV` and/or `SAVEV`, SmartSpice calls and/or saves the previously calculated (or present) operating point. This operating point is used to help the next analysis get started. During a Transient analysis, SmartSpice uses an internal integration time step. The step size depends on the activity of the circuit. You can increase or decrease the internal time step by specifying control parameters in the `.TRAN` and `.OPTIONS` statements.

3.2.2 DC Analysis

SmartSpice DC analysis determines the DC operating point of the circuit with shorted inductors and opened capacitors. By default, the DC analysis is automatically performed prior to a Transient analysis to determine the transient initial conditions. It is also conducted prior to any small-signal analysis to determine the linearized small-signal models for nonlinear devices. If requested, the DC small-signal value of a transfer function (ratio of output variable to input source), input resistance, and output resistance are also computed as part of the DC solution. The DC analysis also generates DC transfer curves in which a specified independent voltage source, independent current source, model parameter, device parameter, or parameter label is stepped over a user-specified range.

If the `.DC` statement contains the parameters `CALLV` and/or `SAVEV`, the program calls and/or saves the previously calculated (or the present) operating point. This operating point is used to help the next analysis get started. The DC analysis options are specified on the `.DC` and `.OP` statements.

The option `SAVEV/CALLV` is the default for DC analysis. The `.OPTION AUTO_CALLVSAVEV=0` must be used to reset the default `SAVEV/CALLV` behavior.

3.2.3 AC Analysis

AC small-signal analysis computes the AC output variables as a function of frequency. By default, the program first computes the DC operating point of the circuit, and determines linear small-signal models for all nonlinear devices in the circuit. The resulting linear circuit is analyzed over a user-specified range of frequencies. The desired output of AC small-signal analysis is usually a transfer function (voltage gain, transimpedance, etc.). If the circuit has only one AC input, it can be set to unit amplitude and zero phase. Therefore, output variables have the same value as the transfer function of the output variable with respect to the input.

If a `.AC` statement contains the parameters `CALLV` and/or `SAVEV`, then the program calls and/or saves the previously calculated (or new) operating point. This operating point is used to help

the next analysis get started. An AC analysis bypasses the DC operating point calculation. If a .AC statement contains the parameters UIC and CALLV, SmartSpice linearizes the circuit around the operating point, and performs the AC analysis. In this case, the AC analysis is free from any “No Path to Ground” problem.

3.2.4 SensitivityAnalysis

The .SNS statement calculates and prints the sensitivity of basic output variables with respect to temperature model parameters, device parameters, and the parameter GMIN. A list of output variables and parameters is specified in the .SNS statement. Sensitivity analysis is applicable to the basic output variables, node voltage, currents through voltage sources and inductors.

The .SNS statement must refer to an analysis type. Currently, the program supports sensitivity calculation at any argument point of DC or Transient analysis. If a .SNS statement is specified, then SmartSpice performs sensitivity analysis for each analysis statement of the input deck.

3.2.5 Noise Analysis

When provided with an input source and an output port, noise analysis calculates the noise contributions of each device (and each noise generator within the device) to the output port voltage. It also calculates the input noise to the circuit equivalent to the output noise referred to in the specified input source. This is done for every frequency point within a specified range. The calculated noise value corresponds to the spectral density of the circuit variable viewed as a stationary Gaussian stochastic process.

After calculating spectral densities, noise analysis integrates these values over the specified frequency range to calculate the total noise. This calculated value corresponds to the variance of the circuit variable viewed as a stationary Gaussian process.

3.2.6 Distortion Analysis

Multiple tones or high power input signals at the input of electronic circuits cause distortion and appear as unwanted signals at the output. They are called Inter-modulation Products (IP).

The possible IPs for two input signals with frequencies f1 and f2 are:

1. First order - no mixing products. The output frequency list consists of input frequencies and the first, second, and third harmonics of each tone:

   ```
   f1, f2, 2f1, 2f2,  3f1, 3f2
   ```

2. Second order - the sum and positive difference of the input frequencies are added to the list:

   ```
   f1 +- f2
   ```

3. Third order - the second harmonic of each tone can mix with the first of the other:

   ```
   2f1 +- f2,    2f2 +- f1
   ```

Distortion analysis computes up to the 3rd order Inter-modulation Products in frequency domain as complex values of all circuit variables. When both input signals have the same frequency, the results are produced at the first, second, and third harmonics of input frequency. If input signals have different frequencies, the results are produced at the sum and difference of the input frequencies, and at the difference of the smaller frequency from the second harmonic of the larger frequency.

3.2.7 Small-Signal Distortion Analysis

Distortion analysis computes steady state harmonic intermodulation products for small input-signal magnitudes. If signals of a single frequency are specified as the input to the circuit, the complex values of the second and third harmonics are determined at every point in the circuit. If there are inputs containing signals of two different frequencies, the analysis finds the complex value of the circuit variables at the sum and difference of the input frequencies, and at the difference of the smaller frequency from the second harmonic of the larger frequency.

3.2.8 Pole-Zero Analysis

Pole-zero analysis computes the poles and/or zeros of the small-signal AC transfer function. The program computes the DC operating point, and determines the linearized small-signal models for all nonlinear devices in the circuit. This circuit is used to find the poles and zeros of the transfer function.

Two transfer function forms are allowed:

- (output voltage)/(input voltage)
- (output voltage)/(input current)

These two transfer function types cover all cases. For example, (output voltage)/(input current) finds the poles/zeros of functions, such as input/output impedance, and (output voltage)/(input voltage) finds the poles/zeros of voltage gain. The input and output ports are specified as two pairs of nodes.

3.2.9 Deck Runs to Completion (even when one Analysis Fails)

If a deck fails due to problems in one particular analysis in the deck, the analysis which cannot be completed will generate an error message, but the simulation will continue with the remaining analyses.

3.3 Output Statements

The following output commands are used to print, plot, and measure the results of SmartSpice simulations. Output statements refer to a particular analysis type. For example:

```
.PRINT DC V(1)
.PLOT DC V(1)
.MEASURE DC MAX_V1 MAX V(1)
```

refers to a DC analysis.

.PRINT	Prints variables or output tables.
.PLOT	Outputs variables for ASCII plotting after simulation.
.MEASURE	Measurement or circuit performance calculation over a user-specified interval.
.SAVE	Specifies variables to be saved during simulation, but is not printed/plotted.
.IPLOT	Specifies variables to be plotted during simulation.

SmartSpice executes all of the output commands after an analysis is finished, and follows the sequence of commands defined in the input file. SmartSpice saves the results of the previous measurements to use them for the next measurement. For example:

```
.MEASURE DC MAX_V1 MAX V(1)
.MEASURE TRAN CR_V1 CROSS V(1) VAL=MAX_V1
```

The first command calculates and saves under the name MAX_V1, which is the maximum value of the output variable V(1) when a DC analysis is finished. The second command calculates the crossing point of the transient curve V(1) with the constant value MAX_V1 when the Transient analysis is finished.

The .PRINT and .PLOT statements can produce large amounts of output data. The .MEASURE statement allows you to decrease the amount of output data by extracting single values. Normally, a .MEASURE statement is used to calculate a few values such as delay, rise time or error. Some analysis statements print calculated results directly without a .PRINT or .PLOT statement. These statements are .SNS, .FOUR and .OP. The .SNS and .FOUR statements depend on other commands within the input file. For example, if the .SNS statement refers to a DC analysis, and the input file does not contain a .DC statement, the sensitivity analysis is not performed and no output is generated.

The following syntax i(device_name.port_name) is supported in output statements (.print, .save, .probe, .plot) for user-defined Verilog-A elements. This syntax accesses the value of current through the specific external port. device_name is the name of a Verilog-A element. port_name is the port name.

Note: The option veriloga-args="-noexpf" (no excluded port flow variable) is required to use the i(device_name.port_name) syntax.

Example

```
x1 net1 net2 net3 net4 VLGmodel ; <- Not Xcall, Verilog-A device!
.print i(x1.d)
```

Verilog-A source file:

```
module VLGmodel (d, g, s, b);
    inout d, g, s, b;
    electrical d, g, s, b;
```

Note: The regular SmartSpice syntax is @x1[i(<d>)].

To access VLG device vector variables bus notation syntax ([]) can be used.

Example

```
// VA file for module stim
‘include "constants.vams"
‘include "disciplines.vams"
module stim(dpa_out_a, dpa_out_b, en_out);
output [25:0] dpa_out_a;
output [25:0] dpa_out_b;
output en_out;
...
// SPICE NETLIST
YVLG1  net1<25>  net1<24>  net1<23>  net1<22>  net1<21>  net1<20>
net1<19> net1<18> net1<17>
+ net1<16> net1<15> net1<14> net1<13> net1<12> net1<11> net1<10>
net1<9> net1<8> net1<7>
```

```
+ net1<6> net1<5> net1<4> net1<3> net1<2> net1<1> net1<0> net0<25>
net0<24> net0<23> net0<22> net0<21> net0<20> net0<19> net0<18>
net0<17> net0<16> net0<15> net0<14> net0<13> net0<12> net0<11>
net0<10> net0<9> net0<8> net0<7> net0<6> net0<5> net0<4> net0<3>
net0<2> net0<1> net0<0> net2 stim
.save @yvlg1[v(dpa_out_a[0:4])]
```

In this example SmartSpice saves 5bit vector `dpa_out_a` of VLG instance `yvlg1`.

3.4 Parameter Modification

The .MODIF statement is used to investigate a circuit for sets of parameters that change over a specified range. SmartSpice simultaneously modifies a specified set of parameters, and performs all analyses specified in the input file for each set of parameters. SmartSpice terminates the parameter modification process at the current set of parameters when at least one of the stop conditions specified within the set is satisfied. Two stop conditions are available:

1. A limit on the number of repetitions of an analysis.
2. A logical condition on a circuit performance measure.

A parameter set contains any number of the following parameter types:

- parameter labels defined in a global .PARAM statement;
- model parameters;
- device parameters;
- temperature parameters;
- and the parameter GMIN/DCGMIN.

The .MODIF statement is a powerful design tool. It is used to perform parametric optimization and statistical analysis. If a series of analyses require a great deal of CPU time and produce excessive output, the .MEASURE statement and parameters CALLV and SAVEV are used in the analysis statement to focus the analysis on the critical data.

The .MODIF statement cannot be run with the .ST or .TEMP statement in the same input file. .MODIF and .DC statements must not attempt to set the same parameters.

3.5 Parameter Sweep

The parameter sweep is used to make multiple runs; change a parameter label, a temperature parameter, a device or model parameter, or the parameters GMIN/DCGMIN; and to analyze these parameters during a series of user-specified changes.

The parameter sweep is supported by:

- .ST (Parametric Analysis) statement. It performs a parametric sweep for all analyses specified in the input deck. See Section .ST (Parametric Analysis).
- SWEEP parameter in the analysis statement. It performs a parametric sweep for the current analysis.
- SWEEP parameter in .MODIF (Parameter Modification) statement. It works as an external loop for the parameter modification or for Bisection optimization. See Section .MODIF (Parameter Modification).

The parameter sweep can be run in parallel by using -mp, -mps or -mpr/-mprg command keys. See Chapter 24 SmartSpice Distributive Capabilities for Monte Carlo and Alter.

The parameter sweep is useful for circuit design, but can use a great deal of CPU time and produce large amounts of output. The .MEASURE statement and the keywords CALLV and SAVEV in the analysis statement help reduce output and CPU time.

3.6 OPTIONS Statement

The `.OPTIONS` statement is used to set SmartSpice control and computational parameters. The default values of these parameters specify the approach SmartSpice uses to perform analyses. For some simulation requirements, the approach is altered by changing the parameters that define it. If a parameter is changed for an analysis, it remains changed for every analysis in the input file.

For a list of parameters available in the `.OPTIONS` statement, see Section 3.14 .OPTIONS (Option Specification). Some parameter changes by the `.OPTIONS` statement are time consuming and produce a large amount of output.

3.7 Initial Conditions

The .IC statement sets transient initial conditions, or operating point conditions for .OP and small-signal analysis. The statement has two different interpretations, depending on whether the parameter UIC is specified in the analysis statement:

- When the analysis statement (.TRAN, .AC or .NOISE) contains the parameter UIC, analysis starts from these initial conditions, and bypasses the DC operating point calculation.
- When the parameter UIC is not specified, SmartSpice uses the initial conditions during the operating point calculation. This means that the nodes specified in the .IC statement will have the specified voltage values when the operating point is found.

Analysis ignores the .IC statement if the analysis statement (.TRAN, .OP, .AC or .NOISE) contains the parameters CALLV.

The .IC statement is local to subcircuits. This means that if an .IC statement is specified in a subcircuit definition, SmartSpice expands the node names that appear in the statement during subcircuit expansion. If two .IC statements refer to the same node, the first statement is overwritten by the second statement and a warning will appear. Local .IC statements defined inside subcircuits have higher priority than top level statements for the same node.

In order to improve the DC convergence of circuits with more than one stable state, two different forms of initial condition may be specified:

1. The first form is related to the option OFF. If a device is specified to be OFF, the DC operating point is calculated with the terminal voltages for the device set to zero. After convergence is obtained, the program obtains the exact values for the terminal voltages. If a circuit has more than one DC stable state, the option OFF can force the solution to correspond to a desired state. If a device is specified OFF (when in reality the device is conducting) the program still obtains the correct solution (assuming the solutions converge). More iterations are required since the program must independently converge to two separate solutions.
2. The second form of initial condition is performed by the .NODESET statement. The .NODESET statement is easier to apply and is the recommended means to aid convergence. It finds the DC operating point by making a preliminary pass with the specified nodes held to the voltages given in the .NODESET statement. The restriction is then released, and the iteration continues to the final DC solution. In most circumstances, this statement is not necessary.

SmartSpice ignores the .NODESET statement during the operating point calculation for a Transient analysis if a .IC statement is specified (for Transient analysis, .IC statement overrides .NODESET). Any analysis ignores .NODESET if the statement for this analysis contains one of the following parameters: UIC, CALLV, and TRANOP. Like .IC, .NODESET is local to subcircuits.

3.8 Continuation Lines

SmartSpice allows several ways of continuing a single logical line over several physical lines:

- A line with a + as the first non-blank character is considered a continuation line.
- If a line has a backslash \ at the end, the next physical line will be considered part of the same logical line.
- In the interest of compatibility, a double backslash \\ may be used to continue lines; some other SPICEs require this within expressions. SmartSpice allows it everywhere, but does not require it anywhere.

If a line ends with a (single or double) backslash \, and the following line begins with a +, current behavior is to regard the backslash as a continuation indicator, and the + as a genuine addition operator.

3.9 Engineering Input Format

The input deck values can be written in engineering format. The complete list of notations and their values are shown in Table 3-1:

Table 3-1 Values, Names, and Matching Notations

Value	Name	Notation
10^{12}	tera	t
10^{9}	giga	g
10^{6}	mega	x(meg)
10^{3}	kilo	k
1	none	none
$3.048x10^{-1}$	foot	ft
$25.4x10^{-6}$	mil	mil
10^{-3}	milli	m
10^{-6}	micro	u
10^{-9}	nano	n
10^{-12}	pico	p
10^{-15}	femto	f
10^{-18}	atto	a

3.10 Option Search

When including a file through the .INCLUDE statement, or loading a library of model parameters through a .LIB statement, by default, SmartSpice will look for these files in the current working directory (./), or in your home directory. It is also used to load subcircuit definitions that have not been explicitly defined or included from a library (with .INCLUDE or .LIB statements).

In the case of undefined subcircuits, SmartSpice will make up a file name from the subcircuit name with a .inc extension. It will then try to locate such a file in the search path specified as an option (default search path is ./). If such a file is located, SmartSpice will load the subcircuit definition from it. Note that only the subcircuit being sought is loaded, and any other definition in the same file is ignored. In the case of a model definition within the subcircuit definition, the model is also loaded at the same time.

This behavior is recursive. If the newly loaded subcircuit definition uses an undefined subcircuit, SmartSpice will also try to locate its definition in a separate file. SmartSpice will not attempt to use a subcircuit definition provided by the user in the case of a subcircuit call from within an undefined subcircuit. Instead, it will always look for a definition in a separate file. This is to reduce the potential conflict (or name clash) between user defined subcircuits and library subcircuits.

The syntax for the .OPTIONS SEARCH is:

```
.{OPT | OPTION | OPTIONS} [other_options>] SEARCH [=] [+]<path |
“ [+]<path1> <path2>...”> [<other_options>]
```

If the + sign is specified at the beginning of a path, the path is appended to the current search path. Otherwise, the new path overrides the previously set search path. For the new path to take effect, it must be specified before the .INCLUDE or .LIB statements.

Example

```
.OPTIONS SEARCH = '+/CadLibs/OptCells /CadLibs/StdCells'
XtopLevel in1 in2 out1 Mplex
```

Assuming that the subcircuit Mplex has not been defined in the input deck, and is not part of any file loaded with .INCLUDE or .LIB statements, SmartSpice will search for a file called mplex.inc (in lowercase letters).

Since the path specified with .OPTIONS SEARCH starts with a + sign, it is merely appended to the existing path (./ by default). Therefore, SmartSpice will attempt to load ./mplex.inc, then /CadLibs/OptCells/mplex.inc, and finally /CadLibs/StdCells/mplex.inc. This process stops as soon as the file is found, regardless whether a proper definition of Mplex is found in the file or not.

3.11 Statement Descriptions

The following section contains SmartSpice statement descriptions. The statements are listed in alphabetical order followed by descriptions of the control statements listed in alphabetical order. The Title statement appears first since it is the first line in an input deck, although technically speaking, it is not a statement.

Title (Title Statement)

Syntax

```
any text
```

The title is the first line of each input deck. It contains any text on one line. If the input file contains more than one input deck, each deck must have its own title statement.

Example

```
POWER AMPLIFIER CIRCUIT
```

* $; (Comment Statement)

Syntax

```
* comment
$ comment
; comment
SmartSpice statement $ comment
SmartSpice statement ; comment
```

Comments can be placed anywhere in the input file.

An asterisk (*), as the first non-blank character in the input line, indicates a comment statement. SmartSpice also supports inline comments that are placed anywhere in the input line (as the first non-blank character or after a statement). By default, SmartSpice uses two inline comment characters: Dollar sign ($) and semicolon (;). All text following these characters is treated as a comment.

Inline comment characters are changed by setting the SmartSpice variable `inlinecc` to a string consisting of the desired character or characters. The default value for `inlinecc` is: `$;`. To make the semicolon character the only inline comment character, set `inlinecc=":"`.

Example

Default setup

```
*1808 MEMORY CELL: FLAT CELL
$ input vin, output v(19)
V12 1 0 5;power supply
.TEMP 0 20 50 $ run with different temperatures
```

User configured setup

```
.CONTROL
   Set inlincc=":"
.ENDC
.TEMP 0 20 50 : run with different temperatures
```

#COM, #ENDCOM (Block Comment Statement)

Syntax

```
#COM
...
#ENDCOM
```

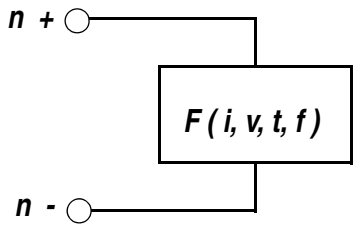
Statements allows you to comment in a multiline block in the input deck. The statements can be nested.

Example

```
.PRINT Ds(in,out)
.PRINT Kd(in,out)
.PRINT Gt(in,out) Gt(out,in)

#COM
.PRINT Sp(in,in) Sp(out,out)          comment line
.PRINT Y(in,in) Y(in,out)             comment line
.PRINT Z(in,in) Z(in,out)             comment line

#COM
.PRINT ZI(out) zi(in)                 comment line
.PRINT ZI(ALL)                        comment line
.PRINT Gms(in,out) Gms(out,in)        comment line
#ENDCOM

.PRINT GTUmax(in,out) GTUmax(out,in)  comment line
.PRINT Gamax(in,out) Gamax(out,in)    comment line
#ENDCOM
```

3.12 Device Statements

A (Analog Behavior Device)



2 connections

Syntax

```
Axxx n+ n- I/V='expression'
+ <SCALE=val> <TC1=val> <TC2=val> <M=val>
+ <MIN=val> <MAX=val> <ABS=1>
```

or

```
Axxx n+ n- DELAY=val <DELIC=val> <RES=val>
+ <SCALE=val> <TC1=val> <TC2=val>
+ <HIGH=val> <LOW=val> <COEF=val>
```

or

```
Axxx n+ n- LAST <DELIC=val> <RES=val>
+ <SCALE=val> <TC1=val> <TC2=val>
```

or

```
Axxx n+ n- I/V='v(nc1+) - v(nc1-)' LAPLACE 'H(s) expession'
+ <SCALE=val> <TC1=val> <TC2=val> <M=val>
+ <DELF|DELTA=val> <MAXF|MAX=val> <TDELTA=val>
```

Axxx	Analog behavior device name. This name must begin with the letter A.
n+, n-	Positive and negative terminal node names.
I	Current through the device (A-device is a current source).
V	Voltage across the device (A-device is a voltage source).
SCALE	Scaling multiplier for the A-device output variable (current or voltage). Default is 1.
TC1	Linear temperature coefficient for the A-device output variable. Default is 0.
TC2	Quadratic temperature coefficient for the A-device output variable. Default is 0.

M	Device multiplier, when A-device is a current source. Default is 1.
MIN	Minimum value of output variable (current or voltage). The condition I/V>=MIN is not effective unless the parameter MIN is specified.
MAX	Maximum value of output variable (current or voltage). The condition I/V<=MAX is not effective unless the parameter MAX is specified.
ABS=1	Output is absolute value of the output variable (current or voltage). Default is ABS=0.
DELAY	Indicates a delay-type device that cannot contain the expression I/V. In Transient analysis, the device propagates a voltage value from node n+ to n- with a delay of val seconds. V(n-)=V(n+) in AC and DC analyses.
DELIC	Initial condition for node voltage V(n-) at time 0.
RES	Output impedance. Default is GMIN.
HIGH	The voltage corresponding to logic state 1. Default is 1.
LOW	The voltage corresponding to logic state 0. Default is 0.
COEF	Used to calculate threshold voltage value. Default is 0.2.

The parameters HIGH, LOW and COEF are not effective unless the option LOGIC is specified in the .OPTIONS statement. The options LOGIC=1 or LOGIC=2 cause an event-driven simulation to be performed, and the A-device's voltage value V(n-) is equal to HIGH, LOW or (HIGH+LOW)/2. These values represent logic states 1, 0, and X respectively.

LAST	Indicates a delay-type device that cannot contain the expression I/V. When transient analysis is performed, the device propagates from n+ to n- the voltage calculated at the previous time point (TIME - TSTEP).
expression	Function of algebraic and logical expressions; real constants; parameter labels; circuit temperature (TEMP); integration time (TIME); integration timestep (TSTEP); frequency (HERTZ); node voltages; voltages across two nodes; branch currents - independent currents through: • independent voltage sources (V), • controlled voltage sources (E and H), • analog behavioral devices (A) with V-type expressions and inductors (L); operator DER; functions DERIVATIVE and INTEGRAL; user-defined functions.

LAPLACE	Parameter to indicate that the transfer function is described by a Laplace transform function in the form of a rational function. Should not be used as a node name.
I/V='v(nc1+)-v(nc1-)'	Indicates the type of source: I- current source, V- voltage source, and indicates the names of positive (nc1+) and negative (nc1-) voltage controlling nodes.
'H(s) expression'	Function of complex frequency variable s = jw, for example the ratio of two polynomials (functions of complex frequencies) with real positive coefficients: $H(s) = \frac{k_0 + k_1 \times s + \ldots + k_n \times s^n}{d_0 + d_1 \times s + \ldots + d_m \times s^m}$

Laplace Transform element modelling uses four mutual parameters:

MAXF(MAX)=val	Nyquist critical frequency - f_c. *2 x f_c* is the frequency window over which transfer function *H(s)* is calculated from LAPLACE, POLE or FREQ description.
DELF(DELTA)=val	Frequency resolution - Δf. *1/Δf* is the time window over which transfer characteristic h(t) is calculated from H(s) by inverse FFT.
TDELTA=val	Time resolution - Δ, which is the uniform time interval between h(t) samples.

These parameter values affect accurate transient results, and the CPU time.

Parameters f_c and Δ are tied hard by the relation *f_c = 1/(2 x Δ)*, and you can to specify only one. If f_c and Δ are both specified, the Δ has a higher priority. The following table shows parameter calculation flow (N- number of h(t) samples):

Δf	**f_c or Δ** **if f_c: Δ=1/(2 x f_c)** **if Δ: f_c=1/(2 x Δ)**	
Default	Default	Δf=1/Tstop; f_c=N x Δf; Δ=1/(2 x f_c); N=1024
Yes	Default	f_c=N x Δf; Δ=1/(2 x f_c); N=1024
Default	Yes	Δf=1/Tstop; N=(2 x f_c)/Δf; match N as power 2; Δf=(2 x f_c)/N
Yes	Yes	N=(2 x f_c)/Δf; match N as power 2; Δf=(2 x f_c)/N

In last case, when Δf and Δ are both specified, *N=(2 x f_c)/Δf* is calculated, and *Δ x N* can be smaller than Tstop or N can be larger than Nmax=8,388,608 (default value can be changed by .TRAN statement option Nmax). In such situations SmartSpice prints the warning message

```
Warning: LAPLACE: required number points N=... > Nmax=...
```

```
finalTime was DECREASED to ...
frequency resolution DELF=...
```

and decreases *Tstop=Δ x min(N,Nmax)*.

Default value of parameter Δ (TDELTA) will be set to minTimeSourceInterval (minimum changing time (rise or fall) of all controlling signals (PULSE and PWL sources) with factor val, and specified in the .TRAN statement option parameter LAPLACE_ACCURATE=val. Default is 0.

Laplace Transform

By default, when an input deck contains SIN sources, SmartSpice will adjust TDELTA and MAXF Laplace transform parameters to have at least 65536 points per period.

Examples

1. The device ADIO describes the simple model of a diode (using a **logical expression**), I(ADIO) is zero for V(1) less or equal to 0, and I(ADIO) is computed as IS*(EXP(40.*V(1))-1.0) for V(1) greater than 0.

```
.PARAM   IS = 1.E-14
ADIO 1 0  I = IF V(1) <= 0.0
+                    THEN   0.0
+                    ELSE   IS*(EXP(40.*V(1)) - 1.0)
```

2. The device A1 describes voltage source, using **logical expressions** and **user-defined functions**:
   - V(1) = - (V(2) + (3)) if V(2) is less than or equal to V(3) and V(2) is less than or equal to 0,
   - V(1) = V(2) + (3)if V(2) is less or equal than V(3) and V(2) is greater 0,
   - V(1) = - (V(2) - (3)) if V(2) is greater than V(3) and V(2) is less or equal 0,
   - V(1) = V(2) - (3) if V(2) is greater than V(3) and V(2) is greater 0.

```
.DEFINE SUM(X,Y)  X + Y
.FUNC    DIFF(X,Y)  X - Y
A1  1  0  V=    (  IF  V(2)<=V(3)    THEN    SUM(V(2),V(3))    ELSE
DIFF(V(2),V(3)) )
+               *( IF V(2)<=0.0     THEN  -1.0   ELSE  1.0)
```

3. The device ACx describes nonlinear capacitances using operator DER and function DERIVATIVE:

```
AC1 A 0  I=0.5pF * (1.0 + ABS(V(A))) * DER.V(A)
AC2 A 0  I=0.5pF * (1.0 + ABS(V(A))) * DERIV(V(A)-V(0))
```

4. The device ALx describes inductances, using the functions DERIVATIVE and INTEGRAL:

```
AL1  A  B  V  = '4nH*(1.0 + ABS(V(A))) * DERIVATIVE(I(AL1))'
AL2  A  B   I = 'INTEGRAL(V(A) - V(B)) / ( 4nH*(1.0 + ABS(V(A))))'
```

5. The device AGen connected between nodes DU and 0 generates one of two specified sinusoidal waveforms, depending on the time with respect to the threshold time point t=20ns:

```
AGen  DU  0  V=  IF  time > 20n
+                THEN
+               EXP(-(TIME - 20n)*0.5MEG) * SIN(6.28 * 1MEG * (TIME
- 20n))
+                ELSE
+               EXP(-(TIME) * 0.25MEG) * SIN(6.28 * 0.25e6 * TIME)
```

6. The device AR connected between nodes A and D describes a temperature dependent linear resistance:

```
.TEMP = 100
.PARAM RNOM = 10
+      TNOM = 25
+      TC1  = 0.02  TC2 = 0.03
+      DT   = 'TEMP - TNOM'
AR  A  D  I = (V(A) - V(D)) / (RNOM * (1. + TC1 * DT + TC2 * DT^2)
```

7. VCVS with the transfer function:

$$H(s) = s^2/(1 + 12 \times s + 22.01 \times s^2 + s^3)$$

`E1 out grnd LAPLACE in grnd 0,0,1/1.0,12.0,22.01,1.0` can be written as:

```
A1 out grnd V='v(in)' LAPLACE  'S*S/(1 + 12*S + 22.01*S*S + S*S*S)'
```

8. VCVS with the transfer function:

$$H(s) = 1.0/(1.0 \times (s + 1.001))$$

`E1out grnd POLE in grnd 1.0/1.0 1.001,0` can be written as:

```
A1 out grnd V='v(in)' LAPLACE  '1/(S+1.001)'
```

A detailed description is provided in the SPICE Models Manual.

B (IBIS Buffer, IBIS Component or JFET/MESFET)

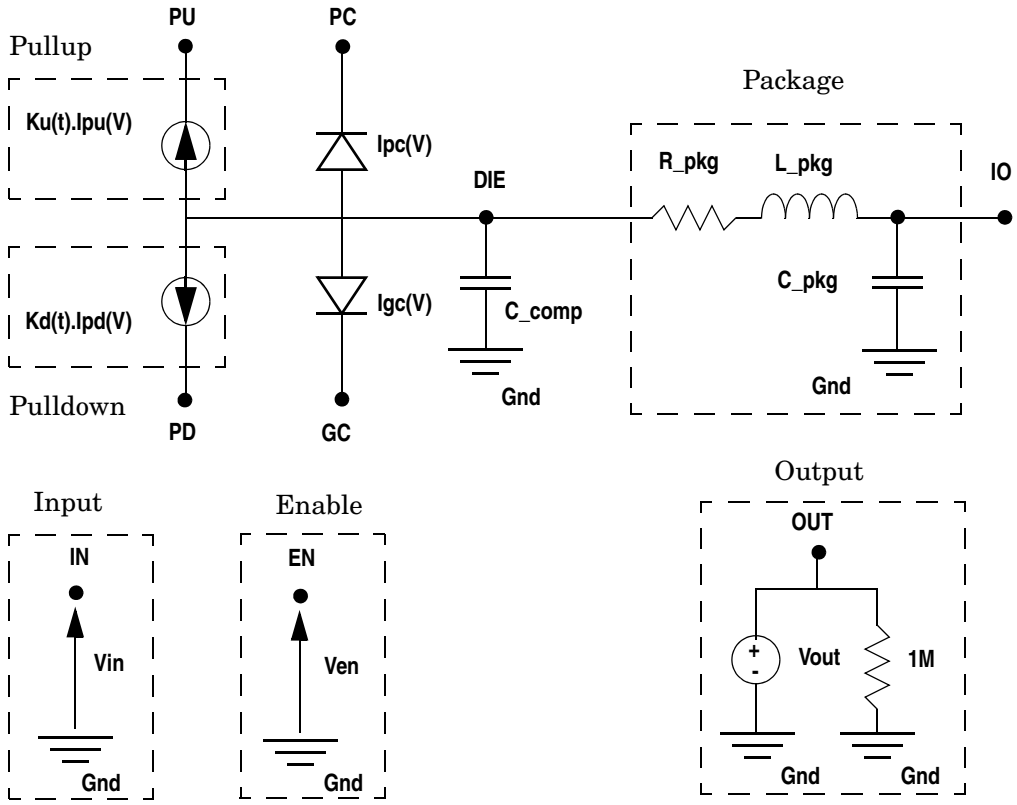


Figure 3-1 General Circuit Diagram

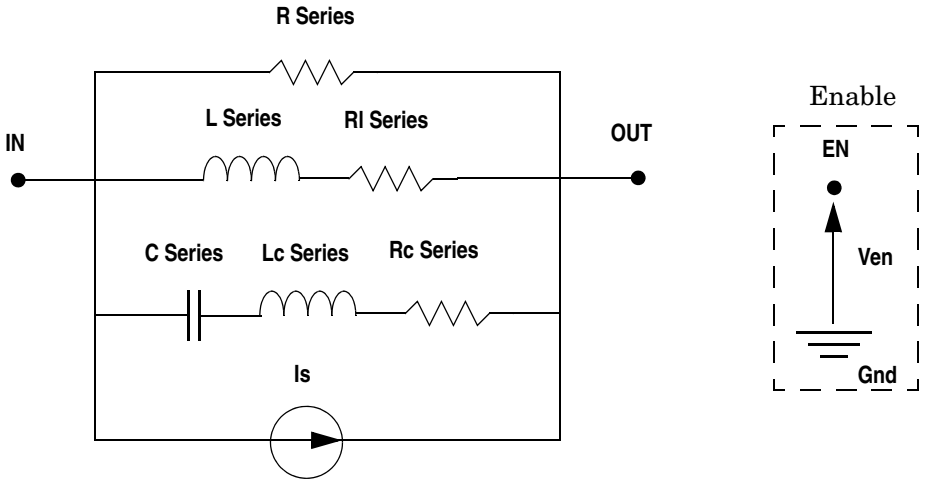


Figure 3-2 Series and Series_switch Circuit Diagram

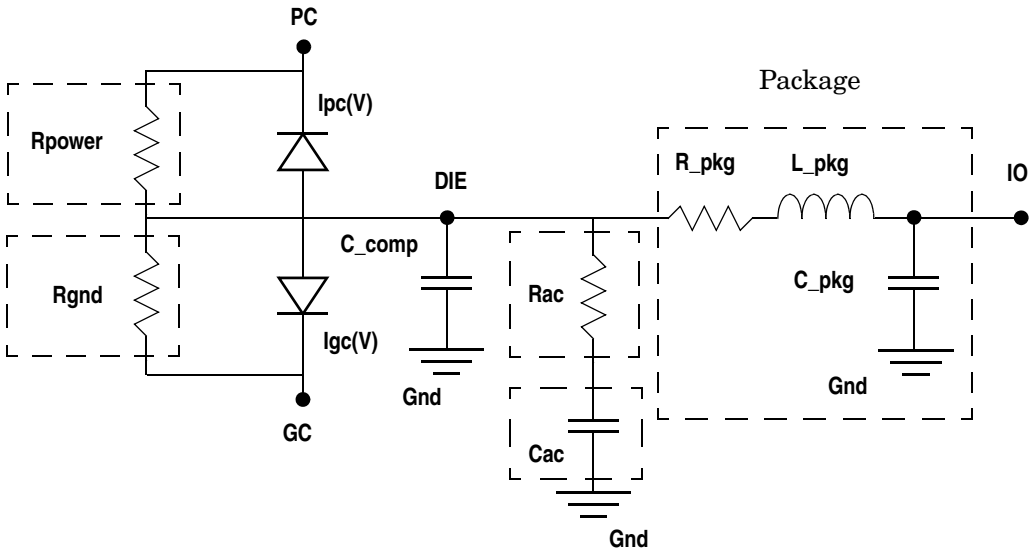


Figure 3-3 Terminator Circuit Diagram

IBIS Buffer Syntax

```
Bname term1 term2 [term3 [term4 [term5 [term6 [term7 [term8]]]]]]
+ file = ‘filename’
+ [model = ‘modelname’] | [component = ‘componentname’
+ [pin = ‘pinname’]]
+ [typ = {typ|min|max|fast|slow}] [power = {on|off}]
+ [interpol = {1|2}] [buffer = {number|type}]
+ [ramp_rwf = {0|1|2}] [ramp_fwf = {0|1|2}] [fwf_tune = value]
+ [rwf_tune = value]
+ [c_comp_pc = value] [c_comp_gc = value] [c_comp_pu = value]
+ [c_comp_pd = value]
+ [nowarn] [mac = value] [risedly = value] [falldly = value]
```

Bname	(Buffer element name) Must begin with a B statement, and followed by alphanumerical characters.
file	(IBIS file name) Required to specify the IBIS file containing the model description for this buffer. Also used to decide if a B statement corresponds to a JFET/MESFET device or a buffer.
model	(IBIS model name) Must reference a [Model] section in the IBIS file. Required if component and pin are not specified.
component	(IBIS component name) Must reference a [Component] section in the IBIS file. Required if pin is specified.
pin	(IBIS pin name) Must reference a [Pin] description in the [Component] section. Required if model is not specified.
typ	(IBIS column) Used to select the data in IBIS ranges: TYP, MIN or MAX. If SLOW or FAST is specified, MIN or MAX columns are selected depending on the IBIS parameter. Default is TYP.

power	(Internal voltage sources switch) Used to select how the buffer is powered through PC, GC, PU and PD nodes if these terminals exist for this type: ON = internal voltage sources, OFF = external voltage sources. Defaults to ON. Ignored for Series and Series_switch types.
interpol	(Interpolation method selector) Linear (1) or quadratic bi-spline (2). Default is 1.
buffer	(Buffer type) Overrides the parameter Model_type in the [Model] section (not recommended).
ramp_fwf/ramp_rwf	(Dynamic data selectors) [Ramp] data (0), one or two waveforms (1, 2) if available in the [Model] section. Ignored for Series and Series_switch types. Default is 2.
fwf_tune/rwf_tune	Control parameters for ramp_fwf=0,1/ramp_rwf=0,1 algorithms. Default is 0.1. Ignored for Series and Series_switch types.
C_comp_pc/ C_comp_gc/ C_comp_pu/ C_comp_pd	(Dimensionless die capacitance partitioning factors) Ignored if unspecified. Ignored for Series and Series_switch types.
nowarn	(Flag) Set to turn off all non-critical warnings issued when reading an IBIS file.
mac	(Selector) Used to turn on (1) and off (0) the DC/AC Mismatch Auto Correction algorithm. Default is 1. Ignored for buffer types with no pullup/pulldown stages.
risedly and **falldly**	(Internal delays) Used to delay rising and falling transitions, respectively. Default is 0.0. Ignored for buffer types with no pullup/pulldown stages.

IBIS Component Syntax

```
Bname term1 [term2 ... [termN]]
+ file = ‘filename’ component = ‘componentname’
+ [typ = {typ|min|max|fast|slow}] [interpol = {1|2}]
+ [ramp_rwf = {0|1|2}] [ramp_fwf = {0|1|2}] [fwf_tune = value]
+ [rwf_tune = value]
+ [c_comp_pc = value] [c_comp_gc = value] [c_comp_pu = value]
+ [c_comp_pd = value]
+ [nowarn] [mac = value] [risedly = value] [falldly = value]
```

Similar to the syntax of a single IBIS buffer except that model and pin must be unspecified, buffer and power are ignored if specified, and the number of terminals specified after the device name must match the number of pins listed in the [component] description.

Examples

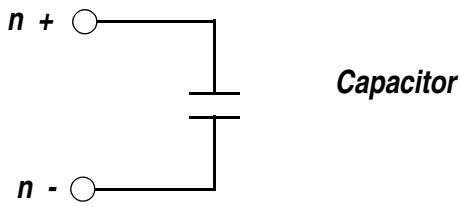
```
B_input  PC  GC  IO  OUT  file=’home/mylogin/myibisfiles/input.ibs’
model=’IN1’
B_output PU PD IO IN [PC [GC]] file=’output.ibs’ model=’OUT1’
```

```
B_three_state  PU  PD  IO  IN  EN  [PC  [GC]]  file='3state.ibs'
component='IC1' pin + = '3'
B_input_output  PU  PD  IO  IN  EN  OUT  [PC  [GC]]  file='IO.ibs'
model='IO5'
B_output_ecl PU IO IN [PC [GC]] file='ECL.ibs' model='in_out7'
+ buffer=OUTPUT_ECL
B_io_ecl PU IO IN EN OUT [PC [GC]] file='ECL.ibs' model='in_out7'
B_three_state_ecl  PU  IO  IN  EN  [PC  [GC]]  file='ECL.ibs'
model='three7'
B_series IN OUT file='passive.ibs' model='series4'
B_terminator PC GC IO file='passive.ibs' model='term10'
B_component  pin1  pin2  pin3  ...  pin48  file='3state.ibs'
component='IC1'
```

The number and the name of terminals depends on the buffer type. For further information see Chapter 7 IBIS Model Support.

Note: In SmartSpice a B statement corresponds to a JFET/MESFET device if the instance parameter `file` is not given. For this case see Section J (JFET/MESFET).

C (Capacitor)



Syntax

Linear

```
Cxxx n+ n- <mname> <<C | CAP=>capval > <M=val> <L=val> <W=val>
+ <SCALE=val> <TC1=val> <TC2= val> <TC=val1, val2> <DTEMP=val>
+ <IC=val>

Cxxx n+ n- <mname> <C | CAP=>capval <tc1val <tc2val> >
+ <M=val> <L=val> <W=val> <SCALE=val> <DTEMP=val>  <IC=val>
```

Polynomial

```
Cxxx n+ n- POLY p0 <p1 ...>
+ <M=val> <SCALE=val> <TC1=val> <TC2=val> <TC=val1, val2>
+ <DTEMP=val> <IC=val>
```

Expression

```
Cxxx n+ n- <Q='expression'
+ <M=val> <SCALE=val> <TC1=val> <TC2=val> <TC=val1, val2>
+ <DTEMP=val> <IC=val>

Cxxx n+ n- <Q='expression' <tc1val <tc2val> >
+ <M=val> <SCALE=val> <DTEMP=val> <IC=val>

Cxxx n+ n- <C=>'expression'
+ <M=val> <SCALE=val> <TC1=val> <TC2=val> <TC=val1, val2>
+ <DTEMP=val> <IC=val>

Cxxx n+ n- <C=>'expression' <tc1val <tc2val> >
+ <M=val> <SCALE=val> <DTEMP=val> <IC=val>
```

Table

```
Cxxx n+ n- Q='expression' TABLE x1,y1 <x2,y2, ... > <M=val>
+ <SCALE=val> <TC1=val> <TC2=val> <TC=val1, val2>
+ <DTEMP=val> <DELTA=val> <IC=val>

Cxxx n+ n- Q='expression' <tc1val <tc2val> > TABLE x1,y1
+ <x2,y2, ... >
+ <M=val> <SCALE=val> <DTEMP=val> <DELTA=val> <IC=val>

Cxxx n+ n- C='expression' TABLE x1,y1 <x2,y2, ... > <M=val>
+ <SCALE=val> <TC1=val> <TC2=val> <TC=val1, val2>
+  <DTEMP=val> <DELTA=val> <IC=val>
```

```
Cxxx n+ n- C='expression' <tc1val <tc2val> > TABLE x1,y1
+ <x2,y2, ... >
+ <M=val> <SCALE=val> <DTEMP=val> <DELTA=val> <IC=val>
```

When mname is specified, the capacitor value capval specification is optional.

Example

```
* Charged-based capacitor

.print ceff=lvl(c1) i(ci) v(n2) v(n3)
.tran 10n 80n
r1 n1 n2 100
r2 n3  0 100
v1 n1  0 pulse(0 7v 1ns 2ns 3ns 10ns 20ns)
c1 n2 n3 q='1e-9*v(n2,n3) +0.5*1e-10*V(n2,n3) **2'
```

Cxxx	Capacitor element name. This name must begin with the letter C.
n+, n-	Positive and negative terminal node names.
mname	Model name. Must reference a capacitor model.
capval	The value of capacitor. If capval is specified as a parameter label, and a capacitor model exists with the same name, the value is interpreted as a model name.
M	Multiplier used to simulate parallel capacitors. Default is 1.
L	Capacitor length. Default is 0.0.
W	Capacitor width. Default is 0.0.
SCALE	Scaling multiplier. Default is the capacitor model parameter SCALE.
TC1, TC2	First and second order temperature coefficients, respectively. The default TC1, TC2 are the capacitor model parameters TC1, TC2. TC1, TC2 parameters can be specified by optional syntax tc1val <tc2val> (without parameters TC1, TC2). In this case, the values tc1val, tc2val must follow immediately after the required <C|CAP=>capval or <C=>'expression' capacitance specification. The optional syntax tc1val<tc2val> can be used instead of TC1, TC2: SCALEeff = SCALE * (1 + TC1*dT + TC2*dT^2), where dT = temperature - t_nominal.
DTEMP	Difference between capacitor instance temperature and circuit temperature in degrees C.
IC	Initial voltage across the capacitor.

POLY	Polynomial function parameter, should not be used as a node name.
p0 <p1...>	Polynomial coefficients. The capacitance is described as a polynomial function of the voltage across the capacitor: `C = p0 + p1 * V(n+,n-) + p2 * V(n+,n-)^2 + p3 * V(n+,n-)^3 + ...`
expression	Function of algebraic and logical expressions; real constants; parameter labels; circuit temperature (TEMP); integration time (TIME); integration timestep (TSTEP); frequency (HERTZ); node voltages; voltages across two nodes; branch currents - independent currents through: • independent voltage sources (V), • controlled voltage sources (E and H), • analog behavioral devices (A) with V-type expressions and inductors (L); operator DER; functions DERIVATIVE and INTEGRAL; user-defined functions.
TABLE	Parameter used to indicate that the capacitance will be found by using a lookup table.

The expression immediately following the parameter C before TABLE or tc1val <tc2val> is evaluated and used to find a value in the table of (x,y) pairs. If the points are not found in the table, they will be calculated by linear interpolation. At least two pairs of values must be defined in the lookup table.

x1,y1 x2,y2...	(Pairs of values) The first value in each pair (x) is the evaluated controlling value, and the second value (y) is the corresponding output. The x values must be in increasing order.
DELTA	Smoothing distance from the corner of piecewise linear function described pairs of values x1,y1 x2,y2.... Default is DELTA=0. If DELTA is specified it is limited by 1/2 of the smallest distance between x axis points.

When capval is specified, it defines the capacitance. When mname is specified, the capacitance is calculated using the process information in the model mname, and the given values for L and W. If capval not specified, mname and L,W must be specified.

Examples

1. Capacitor connected between nodes 12 and 0, with a capacitance of 22pF:

```
CLOAD  12  0  22pf
```

2. Capacitor connected between nodes 7 and 8, with a capacitance of 0.4pF, and an initial voltage of 1.5V:

```
C2  7  8  .4pF  IC=1.5
```

3. Capacitor connected between nodes 3 and 7, with an operating temperature that is different from the circuit temperature of 73 degrees C, a length of 10 microns and width of 1 micron, and a model name cmodel:

```
C11  3   7   cmodel  L=10u  W=1u  DTEMP=73
```

4. Capacitors connected between nodes with symbolic names out and gnd. The value of C991 is 1.e-15 Farads, while the value of C992 is 1.e-30 Farads. This illustrates a common pitfall, since many people will read F in 1.e-15F as Farads, while SmartSpice reads it as femto.

```
C991  out  gnd  1.e-15
C992  out  gnd  1.e-15F
```

5. Capacitor connected between nodes 1 and 0 with capacitance varying with time:

```
C1  1  0  c='1p * (1. + 1n * time)'
```

6. Capacitor connected between nodes 1 and 0 with capacitance varying with frequency:

```
C2  1  0  C='1pF * (1. + 1e-9*HERTZ)'
```

7. Capacitor connected between nodes 1 and 0 with capacitance varying using lookup table with a variable time:

```
C3  1  0  C='time'  TABLE  0,100pF  3ns,100pF  3.1ns,1pF
```

8. Nonlinear capacitor connected between nodes 1 and 2 with capacitance described by the algebraic expression:

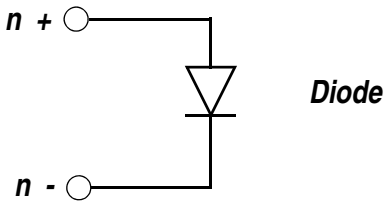
```
Cexpr  1  2  C='100p * (1. + 0.1 * V(1,2) + 0.01 * V(1,2)^2)'
```

9. Capacitor connected between nodes 1 and 0 with capacitance described by logical expression:

```
Cl  1  0  C='   (IF V(1,0) < 2  THEN 100pF  ELSE 200pF*V(1,0 )  )
+              +(IF V(1,0) > 3  THEN 300pF  ELSE 100pF*V(1,0)  )'
```

A detailed description is provided in the SPICE Models Manual.

D (Diode)



Syntax

```
Dxxx n+ n- mname <area> <L=val> <W=val> <PJ=val> <WP=val> <LP=val>
+ <WM=val> <LM=val> <OFF> <IC=val> <M=val> <TEMP=val> <DTEMP=val>
```

Dxxx	Diode element name. Must begin with a D.
n+, n-	Positive (anode) and negative (cathode) terminal node names.
mname	Model name. Must reference a diode model.
area	Area factor. Default is 1.0.
L	Length of diode in meters. Used for LEVEL 3 diode model only.
W	Width of diode in meters. Used for LEVEL 3 only.
PJ	Periphery of the diode junction. Calculated from W and L if they are specified (in the LEVEL 3 model). The ISW and CJSW model parameters are affected by the value of PJ.
WP	Width of the polysilicon capacitor in meters. Used for LEVEL 3 only. Default is 0m.
LP	Length of polysilicon capacitor in meters. Used for LEVEL 3 only. Default is 0m.
WM	Width of metal capacitor in meters. Used for LEVEL 3 only. Default is 0m.
LM	Length of metal capacitor in meters. Used for LEVEL 3 only. Default is 0m.
OFF	Sets ON/OFF startup condition for DC analysis. Default is ON.
IC	Initial voltage across the diode.
M	(Multiplier) Used to describe multiple parallel diodes.
TEMP	Device operating temperature.

DTEMP	Difference in degrees C between the device operating temperature and the circuit temperature. Default is 0.

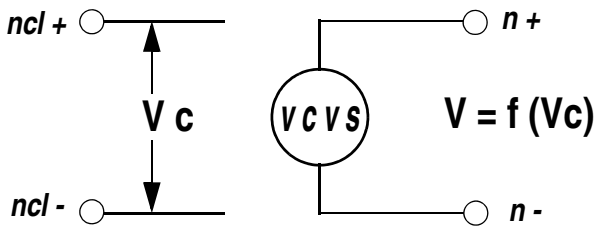
Examples

```
D12 3 4 dmod
Dclmp 3 7 Dio1 3.0 IC=0.3
```

The example defines a diode `Dclmp` connected between nodes `3` and `7`; with the model name `Dio1`, area factor `3`, and initial voltage of `0.3`.

A complete description of diode models is provided in the SPICE Models Manual.

E (Voltage-Controlled Voltage Source)



Syntax

Linear

```
Exxx n+ n- <VCVS> nc1+ nc1- gain
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <IC=val>
```

Polynomial

```
Exxx n+ n- <VCVS> POLY(ndim) nc1+ nc1- nc2+ nc2- ... ncdim+ ncdim-
+ p0  <p1 ...>
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <IC=val>
```

Piecewise Linear

```
Exxx n+ n- <VCVS> PWL(1) nc1+ nc1-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <DELTA=val> <SMOOTH=val> <IC=val>
```

Multi-Input Gates

```
Exxx n+ n- <VCVS> gatetype(k) nc1+ nc1- nc2+ nc2- ... nck+ nck-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <DELTA=val> <SMOOTH=val> <IC=val>
```

Delay Element

```
Exxx n+ n- <VCVS> DELAY nc1+ nc1- TD=val
+ <SCALE=val> <TC1=val> <TC2=val> <NPDELAY=val>
```

Behavior

```
Exxx n+ n- VOL | VALUE ='expression'
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
```

Table

```
Exxx n+ n- TABLE 'expression' = x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
```

Ideal Op-Amp

```
Exxx n+ n- OPAMP nc1+ nc1-
```

Transformer

```
Exxx n+ n- TRANSFORMER nc1+ nc1- K
```

Voltage Gate H(s) (LAPLACE)

```
Exxx n+ n- LAPLACE nc1+ nc1- k0, k1, ..., kn / d0, d1, ..., dm
+ <SCALE=val> <TC1=val> <TC2=val> <DELF | DELTA =val>
+ <MAXF | MAX=val> <TDELTA=val>
```

Voltage Gate H(s) (POLE)

```
Exxx n+ n- POLE nc1+ nc1-
+ a, az1, fz1, ... , azn, fzn  /  b, ap1, fp1, ..., apm, fpm
+ <SCALE=val> <TC1=val> <TC2=val> <DELF | DELTA =val>
+ <MAXF | MAX=val> <TDELTA=val>
```

Voltage Gate H(s) (FREQ)

```
Exxx n+ n- FREQ nc1+ nc1-
* freq mag phase
+ f1,  a1, p1,
+ f2,  a2, p2,
+ ...  ... ...
+ fn   an  pn
+ <SCALE=val> <TC1=val> <TC2=val> <DELF | DELTA =val>
+ <MAXF | MAX=val> <TDELTA=val> <RADIAN=1>
```

Voltage Gain H(z) (ZTRANS)

```
Exxx n+ n- ZTRANS nc1+ nc1- k0, k1, ..., kn / d0, d1, ..., dm
+ <SCALE=val> <TC1=val> <TC2=val> <DELF | DELTA =val>
+ <MAXF | MAX=val> <TDELTA=val>
```

Exxx	Voltage-controlled voltage source names begin with E.
n+, n-	Positive and negative terminal node names of the controlled source.
VCVS	Keyword for voltage controlled voltage source. Should not be used as a node name.
nc1+, nc1-...	Positive and negative controlling node names, in pairs, one pair for each dimension in polynomial form or one pair for each gate in multi-input gate form.
gain	Voltage gain.
MIN	Minimum value of controlled voltage. The condition Vout >= MIN is not effective unless the parameter MIN is specified.
MAX	Maximum value of controlled voltage. The condition Vout <= MAX is not effective unless the parameter MAX is specified.
ABS=1	Output is absolute value of controlled voltage. Default is ABS=0.
SCALE	Scaling factor. Default is SCALE=1.

TC1, TC2	First- and second-order temperature coefficients. Default is TC1=0, TC2=0.

SCALE, TC1 and TC2 are involved in calculation of the output voltage through:

```
Vout = V*SCALE*(1. + TC1*dT + TC2*dT*dT),
```

where dT = temperature - t_nominal.

IC	Initial condition. The initial estimate of the value(s) of controlling voltage(s). Default is IC=0.
POLY	Polynomial function parameter, should not be used as a node name.
ndim	The number of dimensions of the polynomial.
p0, p1,...	Polynomial coefficients.
PWL	Piecewise linear function parameter. Should not be used as a node name.
PPWL	Piecewise linear function parameter. Indicates that the function models the symmetrical bidirectional switch or transfer gate, PMOS. Should not be used as a node name.
NPWL	Piecewise linear function parameter. Indicates that the function models the symmetrical bidirectional switch or transfer gate, NMOS. Should not be used as a node name.
x1,y1 x2,y2...	Pairs of values. The first value in each pair (x) is the controlling voltage across nodes nc1+ and nc1-, and the second value (y) is the corresponding output. The x values must be in increasing order.
DELTA	Smoothing distance from the corner. Default is DELTA=0. If DELTA is specified, DELTA is limited by 1/2 of the smallest distance between x axis points.
SMOOTH	Selects the curve smoothing method. Unused and was introduced only for compatibility with the syntax of other SPICEs.
gatetype(k)	Parameters AND, NAND, OR and NOR set the type of multi-input gate device. The value of k is the number of inputs. AND, NAND, OR and NOR should not be used as node names. For AND and NAND cases, the controlling gate is chosen on the basis of the smallest controlling value. For OR or NOR, the largest controlling value is used. Inversion for NAND and NOR is provided by the character of the user-specified PWL function (but not by the parameters NAND or NOR).

DELAY	Delay element parameter. Should not be used as a node name. The delay element is the same as a voltage controlled voltage source: `Vout(t) = SCALE * (1. + TC1 * dT + TC2 * dT*dT) * Vin(t - TD).`

TD	Time delay.
	`.param param1 =1e-8`
	`EdelayExpTD VOUT_DELAY 0 VCVS DELAY IN 0 TD='50p+2.3*(param1/10)'`

NPDELAY	The number of time points to be saved during a transient analysis for DELAY simulation. The number must be a minimum value of 10 and a maximum value not greater than TD/TSTEP and TSTOP/TSTEP calculations The value of TSTOP and TSTEP are those specified in the .TRAN statement of the input deck. Default NPDELAY value = 10.
VOL|VALUE	Keyword used to indicate that the output voltage is defined by expression.
expression	Function of algebraic and logical expressions; real constants; parameter labels; circuit temperature (TEMP); integration time (TIME); integration timestep (TSTEP); frequency (HERTZ); node voltages; voltages across two nodes; branch currents - independent currents through: • independent voltage sources (V), • controlled voltage sources (E and H), • analog behavioral devices (A) with V-type expressions and inductors (L); operator DER; functions DERIVATIVE and INTEGRAL; user-defined functions.
TABLE	Parameter used to indicate that the output voltage will be found by using a lookup table. The expression immediately following the keyword TABLE is evaluated and used to find a value in the table of (x,y) pair. If the points are not found in the table, they will be calculated by linear interpolation. At least two pairs of values must be defined in the lookup table.
OPAMP	Ideal op-amp element parameter. Should not be used as a node name.
TRANSFORMER	Ideal transformer parameter. Should not be used as a node name.
K	Ideal transformer turn ratio: *Vin = K * Vout*.

Laplace Transform Parameters

LAPLACE	Keyword to indicate that the transfer function is described by a Laplace transform function in the form of a rational function. Should not be used as a node name.
k0, k1,..., kn	Rational function numerator coefficients. All the coefficients k0, k1,..., kn can be parameterized.
d0, d1,..., dm	Rational function denominator coefficients. All the coefficients d0, d1,..., dm can be parameterized.
POLE	Keyword to indicate that the transfer function is described by the location of the poles and zeroes. Should not be used as a node name.
a, az1,fz1,..., azn,fzn	Constant coefficient (a), real (az=Re(zero)) and imaginary (as frequency in *Hertz-fz=Im(zero)/2pi*) parts of zeroes (see the formula in SPICE Models Manual). All the coefficients a, az1,fz1, ..., azn,fzn can be parameterized.
b, ap1,fp1,..., apm,fpm	Constant coefficient (b), real (ap=Re(pole)) and imaginary (as frequency in *Hertz-fp=Im(pole)/2pi*) parts of poles (see the formula in SPICE Models Manual). All the coefficients b, ap1,fp1, ..., apm,fpm can be parameterized. Complex poles or zeroes are in conjugate pairs. In the device description, only one of them is specified.
FREQ	Keyword to indicate that the transfer function is described by a frequency-response table. Should not be used as a node name.
f1,a1,p1, f2,a2,p2,..., fn,an,pn	Triples of values. The first value (f) in each triple is the frequency point in hertz, the second value (a) is the magnitude in dB, and the third value (p) is the phase in degree (or radians). The f values must be in increasing order. All values of f,a, and p in the table can be parameterized. If the points are not found in the table, they will be calculated by interpolation. At least two triples of values must be defined in the frequency-response table.

SmartSpice simulates Laplace Transform elements with any reasonable time resolution TDELTA. However, remember that the smaller the TDELTA the more accurate the transient results and the longer the CPU time is.

Laplace Transform element modelling uses four mutual parameters:

MAXF(MAX)=val	Nyquist critical frequency - f_c. *2 x* f_c is the frequency window over which transfer function H(s) is calculated from LAPLACE, POLE or FREQ description.
DELF(DELTA)=val	Frequency resolution - Δf. *1/*Δ*f* is the time window over which transfer characteristic h(t) is calculated from H(s) by inverse FFT.

TDELTA=val	Time resolution - Δ, which is the uniform time interval between h(t) samples.

These parameter values affect accurate transient results, and the CPU time.

Parameters f_c and Δ are tied hard by relation $f_c = 1/(2 \times \Delta)$, and you can specify only one. If f_c and Δ are both specified, the Δ has a higher priority. The following table shows parameter calculation flow (N- number of h(t) samples):

Δf	f_c or Δ if f_c: Δ=1/(2 x f_c) if Δ: f_c=1/(2 x Δ)	
Default	Default	Δf=1/Tstop; f_c=N x Δf; Δ=1/(2 x f_c); N=1024
Yes	Default	f_c=N x Δf; Δ=1/(2 x f_c); N=1024
Default	Yes	Δf=1/Tstop; N=(2 x f_c)/Δf; match N as power 2; Δf=(2 x f_c)/N
Yes	Yes	N=(2 x f_c)/Δf; match N as power 2; Δf=(2 x f_c)/N

In the last case, when Δf and Δ are both specified, $N=(2 \times f_c)/\Delta f$ is calculated, $\Delta \times N$ can be smaller than Tstop, and N can be larger than Nmax=8,388,608 (default value can be changed by `.TRAN` statement option `NMAX`). In such situations, SmartSpice prints the warning message:

```
Warning: LAPLACE: required number points N=... > Nmax=...
                  finalTime was DECREASED to ...
                  frequency resolution DELF=...
```

and decreases $Tstop=\Delta \times min(N,Nmax)$.

Default value of parameter Δ (`TDELTA`) will be set to `minTimeSourceInterval` (minimum changing time (rise or fall) of all controlling signals (PULSE and PWL sources) with factor `val`, and specified in the `.TRAN` statement option parameter `LAPLACE_ACCURATE=val`. Default is `0`.

RADIAN=1	Indicates that the phase in the `FREQ` table is to be specified in radians.Default is `RADIAN=0`.

Z-Transform

A new Z-transform syntax was added to E devices, which is similar to the Laplace transform function, and allows you to specify a transfer function *H(z)* from the complex frequency variable *z=exp(s)*.

The Z-transform extension allows the frequency (*f*) response to be described in the form of a rational function, i.e., the ratio of two polynomials (function of complex frequencies) with real positive coefficients:

$$H(z) = \frac{k_0 + k_1 \times z^{-1} + k_2 \times z^{-2} + \ldots + k_n \times z^{-n}}{d_0 + d_1 \times z^{-1} + d_2 \times z^{-2} + \ldots + d_m \times z^{-m}}$$

where $z = e^{s} = e^{jw}$, $w = 2\pi \times f/f_S$, $f_S = 2 \times MAXF$ and MAXF is the Nyquist critical frequency.

ZTRANS	Keyword to indicate that the transfer function is described by a Z-transform function in the form of a rational function. This should not be used as a node name.
k0, k1, ..., kn	Rational function numerator coefficients. All the coefficients k0, k1, ..., kn can be parametrized.
d0, d1, ..., dm	Rational function denominator coefficients. All the coefficients d0, d1, ..., dm can be parametrized.

Topology Checker

Disconnected terminals of VCVS automatically are connected by 1Meg resistor to ground. Prevents singular matrix issues.

Examples

1. The device EON describes a linear VCVS connected between nodes 3 and 9, with controlling nodes 8 and 6 and voltage gain of 3.0, i.e., *V(3,9) = 3.0 * V(8,6)*:

```
EON  3  9  8  6   3.0
```

2. Linear VCVS with terminals n1 n2, controlling nodes 1 and 2, and a voltage gain of 200:

```
Eout  n1  n2  POLY(1)  1  2    200
```

3. Nonlinear VCVS whose voltage is a polynomial function of two voltages: v(3,0) and v(4,1):

```
v(va,gnd) = 1.0 *v(3,0) + 1.e-10 *v(4,1):
Ea  va  gnd  POLY(2)  3  0  4  1    0.  1.  1.e-10
```

4. Piecewise linear VCVS whose voltage is a function of the voltage between vc1 and vc2:

```
Ep1  16  17  PWL(1)  vc1  vc2   0,0  1,2u  3,5u   11,1m
```

5. Nonlinear VCVS whose voltage is given by the expression that follows the parameter VOL:

```
Eb  13  0  VOL='10. * log(i(vin)/10m)'
```

6. Tabular description of a non-linear VCVS between nodes 2 and ground whose voltage is determined from (input/output) pairs of values. First, the expression 'v(1)-v(4)' is evaluated, and the resulting value is compared to the input values in the lookup table. If the value is found, the corresponding output value is assigned to the controlled source. If the value is not found, linear interpolation is used to determine the output value.

```
Er  2  0  TABLE  'v(1) - v(4)' =  (0,-100)   (10,5)   (100,50)
```

7. VCVS with the transfer function: $H(s) = s^2/(1 + 12 \times s + 22.01 \times s^2 + s^3)$

```
E1  out  grnd  LAPLACE  in  grnd    0, 0, 1 / 1.0,  12.0,  22.01,
1.0
```

8. VCVS with the transfer function: $H(s) = 1.0/(1.0 \times (s + 1.001))$

```
E1  out  grnd  POLE  in  grnd   1.0 / 1.0  1.001,0.
```

9. Ehigh_pass out 0 ZTRANS in 0:

```
+ -0.0082 -0.1793 0.6579 -0.1793 -0.0082 / 1
```

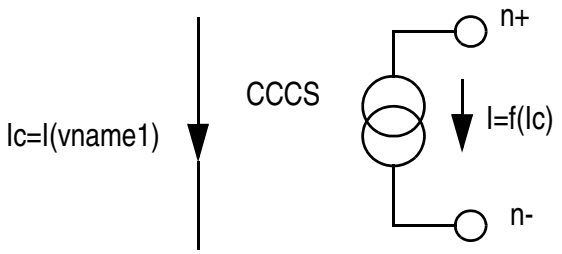
The `Ehigh_pass` element statement describes a fourth-order highpass filter, with the transfer function:

$$H(z) = (-0.0082) - 0.1793 \times z^{-1} + 0.6579 \times z^{-2} - 0.1793 \times z^{-3} - 0.0082 \times z^{-4}$$

Note: If the parameter f=0, the pole or zero is not complex and the conjugate pair: $(s + a - j2\pi \times f) \times (s + a + j2\pi \times f)$ is replaced with $(s + a)$.

A detailed description is provided in the SPICE Models Manual.

F (Current-Controlled Current Source)



Syntax

Linear

```
Fxxx n+ n- <CCCS> vname1 gain
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <IC=val>
```

Polynomial

```
Fxxx n+ n- <CCCS> POLY(ndim) vname1 vname2 ... vnamendim
+ p0 <p1 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <IC=val>
```

Piecewise Linear

```
Fxxx n+ n- <CCCS> PWL(1) vname1
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>
```

Multi-Input Gates

```
Fxxx n+ n- <CCCS> gatetype(k) vname1 vname2 ... vnamek
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>
```

Delay Element

```
Fxxx n+ n- <CCCS> DELAY vname1 TD=val
+ <SCALE=val> <TC1=val> <TC2=val> <NPDELAY=val> <M=val>
```

Fxxx	Current-controlled current source names. Must begin with F.
n+, n-	Positive and negative terminal node names of the controlled source.
CCCS	Keyword for current controlled current source. Should not be used as a node name.
vname1, ...	Names of voltage sources through which the controlling current flows. Their number is equal to ndim or k. Specify one name for each dimension or gate. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of vname.

gain	Current gain.
MIN	Minimum value of controlled current. The condition `Iout >= MIN` is not effective unless the parameter `MIN` is specified.
MAX	Maximum value of controlled current. The condition `Iout <= MAX` is not effective unless the parameter `MAX` is specified.
ABS=1	Output is absolute value of controlled current. Default is `ABS=0`.
SCALE	Scaling factor. Default is `SCALE=1`.
TC1, TC2	First- and second-order temperature coefficients. Defaults are `TC1=0, TC2=0`.
M	Device multiplier. Default is `M=1`.

`SCALE`, `TC1`, `TC2` and `M` are involved in calculation output current through:

```
Iout = I*SCALE*(1. + TC1*dT + TC2*dT*dT)*M,
```

where `dT = temperature - t_nominal`.

IC	Initial condition. The initial estimate of the value(s) of controlling current(s). Default is `IC=0`.
POLY	Polynomial function parameter. Should not be used as a node name.
ndim	The number of dimensions of the polynomial.
p0, p1, ...	Polynomial coefficients.
PWL	Piecewise linear function parameter. Should not be used as a node name.
x1, y1 x2, y2, ...	Pairs of values. The first value in each pair (`x`) is the controlling current through `vname1, vname2, ...`, and the second value (`y`) is corresponding output. The `x` values must be in increasing order.
DELTA	Smoothing distance from the corner. Default is `DELTA=0`. If `DELTA` is specified, it is limited by 1/2 of the smallest distance between x axis points.
SMOOTH	Selects the curve smoothing method. Unused and was introduced only for compatibility with the syntax of other SPICEs.

gatetype(k)	AND, NAND, OR and NOR parameters set the type of multi-input gate device. The value of k is the number of inputs. AND, NAND, OR and NOR should not be used as node names. For AND and NAND cases, the controlling gate is chosen on the basis of the smallest controlling value, while for OR or NOR, the largest controlling value is used. Inversion for NAND and NOR is provided by the character of the user-specified PWL function (but not by the parameters NAND or NOR).
DELAY	Delay element parameter. Should not be used as a node name. The delay element is the same as a current controlled current source: `Iout(t) = SCALE * (1. + TC1 * dT + TC2 * dT*dT) * Iin(t - TD) *M.`
TD	Time delay.
NPDELAY	Number of time points to be saved during Transient analysis for DELAY simulation. The value must be the larger of 10 and the smaller of TD/TSTEP and TSTOP/TSTEP: `NPDELAY = max(10, min(TD,TSTOP)/ TSTEP, NPDELAY_user).` The value of TSTOP and TSTEP are specified in the .TRAN statement. NPDELAY_user is the user-specified value of NPDELAY. Default is NPDELAY_user=10.

Output parameters are accessed by using the syntax: `@F-devicename[paramname]`.

Examples

1. Linear CCCS connected between nodes 1 and 10 with the controlling current flowing through the independent voltage source vsens and a current gain of 6.0 (i.e., `i = 6.0 * i(vsens)`):

   ```
   FA  1  10  vsens  6.0
   ```

2. Nonlinear CCCS with a current that is a polynomial function of the two currents, i(vcc)and i(vee), i = i(vcc) * i(vee):

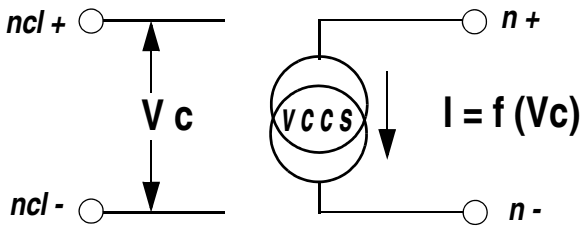
   ```
   F12  4  7  POLY(2)  vcc  vee  0  0  0  0  1
   ```

3. Piecewise linear CCCS with current that is a function of the current through the independent voltage source v1:

   ```
   Fa  3  0  PWL(1)  v1  -1,-3mA  0,-1mA  2,4mA
   ```

A detailed description is provided in the SPICE Models Manual.

G (Voltage-Controlled Current Source)



Syntax

Linear

```
Gxxx n+ n- <VCCS> nc1+ nc1- transconductance
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+  <M=val> <IC=val>
```

Polynomial

```
Gxxx n+ n- <VCCS> POLY(ndim) nc1+ nc1- nc2+ nc2- ... ncdim+ ncdim-
+ p0 <p1 ...>
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <IC=val>
```

Piecewise Linear

```
Gxxx n+ n- <VCCS> PWL(1) nc1+ nc1-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>

Gxxx n+ n- <VCCS> NPWL(1) nc1+ nc1-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>

Gxxx n+ n- <VCCS> PPWL(1) nc1+ nc1-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>
```

Multi-Input Gates

```
Gxxx n+ n- <VCCS> gatetype(k) nc1+ nc1- nc2+ nc2- ... nck+ nck-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>
```

Delay Element

```
Gxxx n+ n- <VCCS> DELAY nc1+ nc1- TD=val
+ <SCALE=val> <TC1=val> <TC2=val> <NPDELAY=val> <M=val>
```

Behavior

```
Gxxx n+ n- CUR | VALUE ='expression'
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
```

```
+ <M=val>
```

Table

```
Gxxx n+ n- TABLE ‘expression’ = x1,y1 <x2,y2  ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val>

Gxxx n+ n- TABLE ‘expression1’ ‘expression2’ = (x1,y1,z1)
+ (x2,y2,z2) (x3,y3,z3) ...
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val>
```

Transconductance H(s) (LAPLACE)

```
Gxxx n+ n- LAPLACE nc1+ nc1-  k0, k1, ..., kn / d0, d1, ..., dm
+ <SCALE=val> <TC1=val> <TC2=val> <M=val> <DELF | DELTA =val>
+ <MAXF | MAX = val> <TDELTA=val> <kaiserwindow=val>
+ <impulsesmooth=val> <impulsetrunc=val>
```

Transconductance H(s) (POLE)

```
Gxxx n+ n- POLE nc1+ nc1-
+ a, az1, fz1, ... , azn, fzn / b, ap1, fp1, ..., apm, fpm
+ <SCALE=val> <TC1=val> <TC2=val> <M=val> <DELF | DELTA =val>
+ <MAXF | MAX = val> <TDELTA=val>
```

Transconductance H(s) (FREQ)

```
Gxxx n+ n- FREQ nc1+ nc1-
* freq mag phase
+ f1,  a1, p1,
+ f2,  a2, p2,
+ ...  ... ...
+ fn   an  pn
+ <SCALE=val> <TC1=val> <TC2=val> <M=val> <DELF | DELTA =val>
+ <MAXF | MAX = val> <TDELTA=val> <RADIAN=1>
```

Transconductance H(z) (ZTRANS)

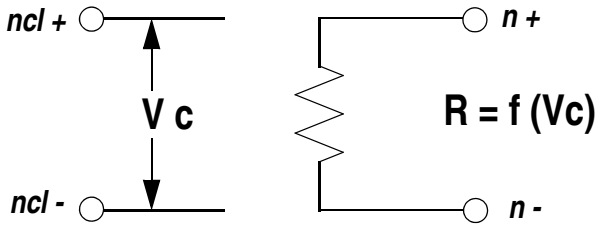
```
Gxxx n+ n- ZTRANS nc1+ nc1- k0, k1, ..., kn / d0, d1, ..., dm
+ <SCALE=val> <TC1=val> <TC2=val> <DELF | DELTA =val>
+ <MAXF | MAX = val> <TDELTA=val>
```

The value of the source can be specified as a function of branch currents.

Example

```
G x xx cur='I(R1)'
```

Voltage Controlled Resistor (VCR)



Syntax

Linear

```
Gxxx n+ n- VCR nc1+ nc1- transfactor
+ <MIN=val> <MAX=val> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <IC=val>
```

Polynomial

```
Gxxx n+ n- VCR POLY(ndim) nc1+ nc1- nc2+ nc2- ... ncdim+ ncdim-
+ p0 <p1 ...>
+ <MIN=val> <MAX=val> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <IC=val>
```

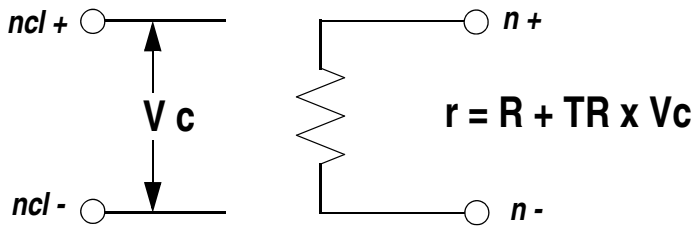
Piecewise Linear

```
Gxxx n+ n- VCR PWL(1) nc1+ nc1-
+ x1,y1 <x2,y2  ... >
+ <MIN=val> <MAX=val> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>

Gxxx n+ n- VCR NPWL(1) nc1+ nc1-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>

Gxxx n+ n- VCR PPWL(1) nc1+ nc1-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>
```

Multi-Input Gates

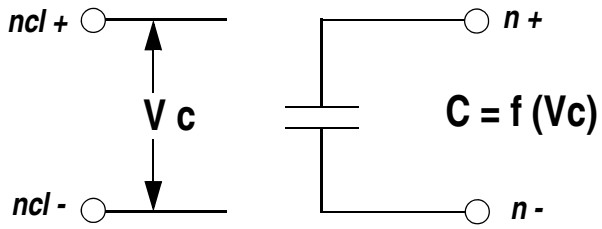
```
Gxxx n+ n- VCR gatetype(k) nc1+ nc1- nc2+ nc2- ... nck+ nck-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>
```

Voltage Controlled Resistor Switch (VCRS)




```
Gxxx n+ n- nc1+ nc1- <MIN=val> <MAX=val> <R=val> TR=val
```

Voltage Controlled Capacitor (VCCAP)



Syntax

Piecewise Linear

```
Gxxx n+ n- VCCAP PWL(1) nc1+ nc1-
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>
```

Behavior

```
Gxxx n+ n- VCCAP VALUE = 'expression'
+ <MIN=val> <MAX=val> <SCALE=val> <TC1=val> <TC2=val>
+ <M=val> <DELTA=val> <SMOOTH=val> <IC=val>=
```

Multi-Input Table

```
Gxxx n+ n- VCCAP TABLE  'expression1' 'expression2' = x1,y1,z1
+ x2,y2,z2 ...
+ <MIN=val> <MAX=val> <SCALE=val> <TC1=val> <TC2=val> <M=val>
+ <DELTA=val> <SMOOTH=val> <IC=val>=
```

Gxxx	Voltage-controlled current source names begin with G.
n+, n-	Positive and negative terminal node names of the controlled source.

VCCS	Keyword for voltage controlled current source. Should not be used as a node name.
nc1+, nc1-,...	Positive and negative controlling node names are in pairs; one pair for each dimension in polynomial form, or one pair for each gate in multi-input gate form.
transconductance	Voltage to current conversion factor.
MIN	Minimum value of controlled current (or resistance for VCR and VCRS, and capacitance for VCCAP). The condition controlled value >= MIN is not effective unless the parameter MIN is specified.
MAX	Maximum value of controlled current (or resistance for VCR and VCRS, and capacitance for VCCAP). The condition controlled value <= MAX is not effective unless the parameter MAX is specified.
ABS=1	Output is absolute value of controlled current. Default is ABS=0.
SCALE	Scaling factor. Default is SCALE=1.
TC1, TC2	First- and second-order temperature coefficients. Defaults are TC1=0, TC2=0. SCALEeff = SCALE*(1. + TC1*dT + TC2*dT*dT), where dT=temperature - t_nominal.
IC	Initial condition. The initial estimate of the value(s) of controlling voltage(s). Default is IC=0.
POLY	Polynomial function parameter. Should not be used as a node name.
ndim	The number of dimensions of the polynomial.
p0, p1,...	Polynomial coefficients.
PWL	Piecewise linear function parameter. Should not be used as a node name.
PPWL	Piecewise linear function parameter. Indicates that the function models the symmetrical bidirectional switch or transfer gate, PMOS. Should not be used as a node name.
NPWL	Piecewise linear function keyword. Indicates that the function models the symmetrical bidirectional switch or transfer gate, NMOS. Should not be used as a node name.
x1,y1 x2,y2...	Pairs of values. The first value in each pair (x) is the controlling voltage across nodes nc1+ and nc1-, and the second value (y) is the corresponding output. The x values must be in increasing order.

DELTA	Smoothing distance from the corner. Default is DELTA=0. If DELTA is specified, it is limited by 1/2 of the smallest distance between x axis points.
SMOOTH	Selects the curve smoothing method. Unused and was introduced only for compatibility with the syntax of other SPICEs.
gatetype(k)	AND, NAND, OR and NOR parameters set the type of multi-input gate device. The value of k is the number of inputs. AND, NAND, OR and NOR should not be used as node names. For AND and NAND cases, the controlling gate is chosen on the basis of the smallest controlling value, while for OR or NOR, the largest controlling value is used. Inversion for NAND and NOR is provided by the character of the user-specified PWL function (but not by the parameters NAND or NOR).
DELAY	Delay element parameter. Should not be used as a node name. The delay element is the same as a current controlled current source: `Iout(t) = SCALE * (1. + TC1 * dT + TC2 * dT*dT) * Iin(t - TD) *M.`
TD	Time delay.
NPDELAY	Number of time points to be saved during Transient analysis for DELAY simulation. The value must be the larger of 10 and the smaller of TD/TSTEP and TSTOP/TSTEP: `NPDELAY = max(10, min(TD,TSTOP)/ TSTEP, NPDELAY_user)` The value of TSTOP and TSTEP are specified in the .TRAN statement. NPDELAY_user is the user-specified value of NPDELAY. Default is NPDELAY_user=10.
CUR|VALUE	Parameter used to indicate that the output current value is defined by expression.
expression	Function of algebraic and logical expressions; real constants; parameter labels; circuit temperature (TEMP); integration time (TIME); integration timestep (TSTEP); frequency (HERTZ); node voltages; voltages across two nodes; branch currents - independent currents through: • independent voltage sources (V), • controlled voltage sources (E and H), • analog behavioral devices (A) with V-type expressions and inductors (L); operator DER; functions DERIVATIVE and INTEGRAL; user-defined functions.

TABLE expr	Parameter used to indicate that the current through the controlled source will be found by using a lookup table. The expression immediately following the parameter TABLE is evaluated and used to find a value in the table of (x,y) pair. If the points are not found in the table, they will be calculated by linear interpolation. At least two pairs of values must be defined in the lookup table. (See Example 1 below)
TABLE expr1 expr2	With two expressions, expression1 and expression2, calculates an interpolated value using Thin Plate Spline (TPS) interpolation algorithm by given scattered points (x,y,z) on a surface. X and Y correspond to expression1 and expression2 calculated values. Z represents a device state at X,Y. Minimum number of points is three. Recommended maximum number of points is 10,000. (See Example 2 below)

Example 1

```
** Gx n+ n- VCCAP VALUE = "expr"

V1 NET1 GND DC 1 PWL (0u 0 5u 0 5.001u 1) AC=1
R1 NET1 NET2  20
R2 NET1 GND  10
R3 NET2 GND  30
G1 NET1 NET2 VCCAP VALUE = '1p  + 1p'
C1 NET1 NET2 2p
G2 NET1 GND VCCAP VALUE = '1p*i(r1) + 1p'

.tran 10n 10u
.ac DEC 10 10K 10G

.probe i(r1) i(G1) i(G2) @G1[cap] @G2[cap]
.probe v(NET1) v(NET2)
.option nomod nodeck
```

Example 2

```
# G-device VCCAP TABLE "expr1" "expr2" =  x1,y1,z1 ...
v1  1  0 pwl (0n 0.0  5000n 115000n 1 20000n 0)
r1  1  2  100
g1  2  0  vccap TABLE "v(2)+0.1" "v(0)*1.1" = (0,   0,   2.8e-12)
(0,   1,   2.8e-12) (1,   0,   2.8e-12) (1,   1,   12.8e-12)  (0.5,
0.5,   1.5e-12)
.tran 1n 20u

.let v2_v0_diff 'v(2)-v(0)'
.probe v(1) v(2) @g1[cap]
.probe i(r1) i(v1)  i(g1)
```

Laplace Transform Parameters

LAPLACE	Keyword to indicate that the transfer function is described by a Laplace transform function in the form of a rational function. Should not be used as a node name.
k0, k1,..., kn	Rational function numerator coefficients. All the coefficients k0, k1,..., kn can be parameterized.
d0, d1,..., dm	Rational function denominator coefficients. All the coefficients d0, d1,..., dm can be parameterized.
POLE	Keyword to indicate that the transfer function is described by the location of the poles and zeroes. Should not be used as a node name.
a, az1,fz1,..., azn,fzn	Constant coefficient (a), real (*az=Re(zero)*) and imaginary (as frequency in *Hertz-fz=Im(zero)/2pi*) parts of zeroes (see the formula in SPICE Models Manual). All the coefficients a, az1,fz1, ..., azn,fzn can be parameterized.
b, ap1,fp1,..., apm,fpm	Constant coefficient (b), real (*ap=Re(pole)*) and imaginary (as frequency in *Hertz-fp=Im(pole)/2pi*) parts of poles (see the formula in SPICE Models Manual). All the coefficients b, ap1,fp1, ..., apm,fpm can be parameterized.

Complex poles or zeroes are in conjugate pairs. In the device description, only one of them is specified.

FREQ	Keyword to indicate that the transfer function is described by a frequency-response table. Should not be used as a node name.
f1,a1,p1, f2,a2,p2,..., fn,an,pn	Triples of values. The first value (f) in each triple is the frequency point in hertz, the second value (a) is the magnitude in dB, and the third value (p) is the phase in degree (or radians). The f values must be in increasing order. All values of f,a, and p in the table can be parameterized. If the points are not found in the table, they will be calculated by interpolation. At least two triples of values must be defined in the frequency-response table.

SmartSpice simulates Laplace Transform elements with any reasonable time resolution TDELTA. However, remember that the smaller the TDELTA the more accurate the transient results and the longer the CPU time is.

Laplace Transform element modelling uses three mutual parameters:

MAXF(MAX)=val	Nyquist critical frequency - f_c. *2 x* f_c is the frequency window over which transfer function H(s) is calculated from LAPLACE, POLE or FREQ description.

DELF(DELTA)=val	Frequency resolution - Δf. *1/Δf* is the time window over which transfer characteristic h(t) is calculated from H(s) by inverse FFT.
TDELTA=val	Time resolution - Δ, which is the uniform time interval between h(t) samples.
kaiserwindow=val	Specifies the parameter's value for Kaiser filter. If specified in the instance statement, the impulse response of the given G-element will be processed with Kaiser-Bessel window. Default is 0 (no windowing).
impulsesmooth=val>	Specifies the smooth width parameter for rectangular smoothing of the impulse response. Integer number only. Recommended value is 2. Default is 0 (no smoothing).
impulsetrunc=val>	Float number with the range [0;1]. Specifies the coefficient for truncation of the tail of impulse response. Recommended no more than 0.1. Default is 0 (no truncation).

These parameter values affect accurate transient results, and the CPU time.

Parameters f_c and Δ are tied hard by relation $f_c = 1/(2 \times \Delta)$, and you can specify only one. If f_c and Δ are both specified, the Δ has a higher priority. The following table shows parameter calculation flow (N- number of h(t) samples):

Δf	**f_c or Δ** **if f_c: Δ=1/(2 x f_c)** **if Δ: f_c=1/(2 x Δ)**	
Default	Default	Δf=1/Tstop; f_c=N x Δf; Δ=1/(2 x f_c); N=1024
Yes	Default	f_c=N x Δf; Δ=1/(2 x f_c); N=1024
Default	Yes	Δf=1/Tstop; N=(2 x f_c)/Δf; match N as power 2; Δf=(2 x f_c)/N
Yes	Yes	N=(2 x f_c)/Δf; match N as power 2; Δf=(2 x f_c)/N

In the last case, when Δf and Δ are both specified, $N=(2 \times f_c)/\Delta f$ is calculated, $\Delta \times N$ can be smaller than Tstop, and N can be larger than Nmax=8,388,608 (default value can be changed by the `.TRAN` statement option `NMAX`). In such situations, SmartSpice prints the warning message:

```
Warning: LAPLACE: required number points N=... > Nmax=...
                  finalTime was DECREASED to ...
                  frequency resolution DELF=...
```

and decreases *Tstop=*Δ *x min(N,Nmax)*.

Default value of parameter Δ (`TDELTA`) will be set to `minTimeSourceInterval` (minimum changing time (rise or fall) of all controlling signals (PULSE and PWL sources) with factor

val, and specified in the .TRAN statement option parameter LAPLACE_ACCURATE=val. Default is 0.

RADIAN=1	Indicates that the phase in the FREQ table is to be specified in radians.Default is RADIAN = 0.

VCR, VCCAP Parameters

VCR	Parameter for voltage controlled resistor. Should not be used as a node name.
transfactor	Voltage to resistance conversion factor.
R	Resistance. Default is R=0.0.
TR	Voltage to resistance conversion factor for voltage controlled resistor switch. RESistance = R + TR * (v(nc1+) - v(nc1-))
VCCAP	Parameter for voltage controlled capacitor. Should not be used as a node name.

Output parameters are accessed by using the syntax: @G-devicename[paramname].

Current through G-device can be accessed by using the syntax: i(G-devicename).

Z-Transform

A new Z-transform syntax was added to G devices, which is similar to the Laplace transform function, and allows you to specify a transfer function *H(z)* from the complex frequency variable *z=exp(s)*.

The Z-transform extension allows the frequency (*f*) response to be described in the form of a rational function, i.e., the ratio of two polynomials (function of complex frequencies) with real positive coefficients:

$$H(z) = \frac{k_0 + k_1 \times z^{-1} + k_2 \times z^{-2} + \ldots + k_n \times z^{-n}}{d_0 + d_1 \times z^{-1} + d_2 \times z^{-2} + \ldots + d_m \times z^{-m}}$$

where $z = e^{s} = e^{jw}$, $w = 2\pi \times f / f_s$, $f_s = 2 \times MAXF$ and MAXF is the Nyquist critical frequency.

ZTRANS	Keyword to indicate that the transfer function is described by a Z-transform function in the form of a rational function. This should not be used as a node name.
k0, k1,..., kn	Rational function numerator coefficients. All the coefficients k0, k1,..., kn can be parametrized.
d0, d1,..., dm	Rational function denominator coefficients. All the coefficients d0, d1,..., dm can be parametrized.

Parameters

```
<SCALE=val> <TC1=val> <TC2=val> <DELF | DELTA =val> <MAXF | MAX =
val> <TDELTA=val>
```

The same as described in G-device “Laplace Transform Parameters”.

Examples

1. Linear VCCS connected between nodes 3 and 4, with controlling nodes 1 and 0, and transconductance of 2 ohms:

```
G2  3  4  1  0   2.0
```

2. Nonlinear VCCS with a current that is a polynomial function of four voltages: v(5,0), v(6,0), v(7,0) and v(8,0). This current is given by the expression *v(5)+v(6)+v(7)+v(8)-v(6)*v(8)*:

```
Gp  11  12  POLY(4)  5 0  6 0  7 0  8 0   0  1  1  1  1  0  0  0  0
0  0  -1
```

3. Piecewise linear VCCS with a current that is a function of the voltage between nodes 2 and 0:

```
Ga  3  0  PWL(1)  2  0   0,10m   1.4,1m   2.4, 0.1m
```

4. Nonlinear VCCS with current given by the expression that follows the parameter CUR:

```
Gout  0  8  CUR='v(3)*v(4) - v(5)*v(6)'
```

5. Tabular description of nonlinear VCCS between nodes 2 and 3 whose current is determined from (input, output) pairs of values:

```
Gr 2 3 TABLE   v(1)= (-2,100) (0.3,0)  (2.7,20m)  (5,60m)
```

6. Voltage-controlled resistor switch connected between nodes 1 and 2, with controlling nodes 4 and 5. The minimum and maximum limits for the controlled resistance are 1300 and 1meg, respectively, and its value is: *1meg - 0.5meg * (v(4)-v(5))*:

```
G1  1  2  4  5  MIN=1300  MAX=1meg  R=1meg  TR=-0.5meg
```

7. VCCS connected between nodes grnd and out, with controlling nodes in and grnd, and with the transfer function:

$$H(s) = 1.0/(1 + 12 \times s + 22.01 \times s^2 + 31. \times s^3)$$

```
Gl grnd out LAPLACE in grnd   1 / 1.0, 12.0, 22.01, 31.0
```

8. VCCS connected between nodes grnd and out, with controlling nodes in and grnd, and with the transfer function:

$$H(s) = s/(1 \times (s + 1.5 - j2\pi \times 0.01) \times (s + 1.5 + j2\pi \times 0.01))$$

```
GL grnd out POLE in grnd 1  0.0,0.0  /  1.0  1.5,0.01
```

9. VCCS connected between nodes 0 and outg, with controlling nodes in and 0, and whose transfer function is described by the following table:

```
g_low_pass  0  outg  FREQ  in  0
*   freq              mag             phase
+   1.0000e+03       -6.3950e-03    -1.2570e-01
+   1.0965e+03       -6.3951e-03    -1.3785e-01
+   1.2023e+03       -6.3954e-03    -1.5117e-01
+   1.3183e+03       -6.3959e-03    -1.6579e-01
            ...                        ...                        ...
+   3.6308e+03       -6.9888e-03    -4.6028e-01
+   3.9811e+03       -7.4321e-03    -5.0567e-01
+   3.6308e+03       -8.2044e-03    -5.5579e-01
```

```
+            ...                    ...                    ...
+   7.5858e+05       -1.0069e+02     1.6128e+00
+   8.3176e+05       -1.0309e+02     1.6091e+00
+   9.1201e+05       -1.0549e+02     1.6057e+00
+   1.0000e+06       -1.0789e+02     1.6026e+00
+         tdelta=0.01   radian=1
```

10. Voltage-controlled resistor connected between nodes `d` and `d1`, and with controlling multi-input gates `v(g1,0)`, `v(g2,0)` and `v(g3,0)`:

```
gAND  d  d1   VCR   AND(3)  g1  0   g2  0   g3  0
+         delta=0.1   scale=2    m=2    tc1=1.1e-10    tc2=1.2e-10
+         0,1G   1,1G   2,1   5,1
```

11. Voltage-controlled resistor connected between nodes `d` and `0`, the resistance is the piecewise linear function of the voltage between *v(g)-v(0)*:

```
gpmos  d  0   VCR    PPWL(1)   g  0
+        -5v,2k   -3v,4k   -2v,50k   -1v,10meg   -0.4v,150g
```

12. `Glow_pass 0 out ZTRANS` in `0`

```
+ 0.0317 0.095 0.0951 0.0317 / 1.0 -1.459 0.9104 -0.1978
```

The `Glow_pass` element statement describes a third-order low-pass filter, with the transfer function:

$$H(z) = \frac{0.0317 + 0.095 \times z^{-1} + 0.0951 \times z^{-2} + 0.0317 \times z^{-3}}{1 - 1.459 \times z^{-1} + 0.9104 \times z^{-2} - 0.1978 \times z^{-3}}$$

A detailed description is provided in the SPICE Models Manual.

Voltage Controlled Voltage Source

Syntax

Linear

```
Exxx n+ n- nc1+ nc1- gain <ic>
```

ic	Treated as “initial condition” for `gain`.

The resulting value of parameter `gain` is calculated as the sum of `gain` and `ic`.

Example

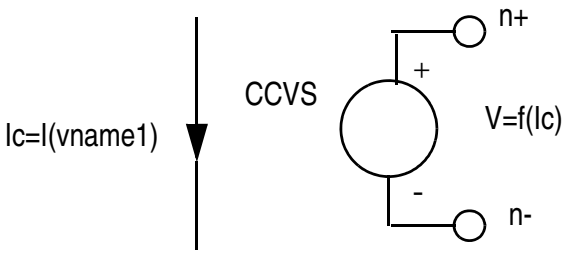
```
E1 N1 N2 VCVS NC1 NC2 0.5 0.7
```

The parameters `GAIN` and `IC` will be set to 0.5 and 0.7 correspondingly.

```
********** G-device table multi input
*** CASE:  scattered surface generated by 11 points
* min/max for epxr1 [0;1], min/max for y [0;1];  min/max for z1 [0;
0.5]
Gdd1 2 1 TABLE ‘v(3)’ ‘v(3)’  =
+ (0, 0, 0)
+ (0, 1, 0)
+ (1, 0, 0)
+ (1, 1, 0)
+ (0.260109, 0.219489, 0.124912)
+ (0.602771, 0.203772, 0.473006)
```

```
+ (0.00885037, 0.675832, 0.218009)
+ (0.389294, 0.776574, 0.263649)
+ (0.0761742, 0.910489, 0.405942)
+ (0.85345, 0.116123, 0.396146)
+ (0.991668, 0.348308, 0.49971)
r1 1 0 0.5K
r2 3 2 0.5k
vac 3 0 pwl 0 0 10n 1
.tran 1n 10n
.probe  v(3)  i(r1) i(r2) i(vac) i(gdd1)
.option nomod
.end
```

H (Current-Controlled Voltage Source)



Syntax

Linear

```
Hxxx n+ n- <CCVS> vname1 transresistance
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <IC=val>
```

Polynomial

```
Hxxx n+ n- <CCVS> POLY(ndim) vname1 vname2 ... vnamendim
+ p0 <p1 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <IC=val>
```

Piecewise Linear

```
Hxxx n+ n- <CCVS> PWL(1) vname1
+ x1,y1  <x2,y2  ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <DELTA=val> <SMOOTH=val> <IC=val>
```

Multi-Input Gates

```
Hxxx n+ n- <CCVS> gatetype(k) vname1 vname2 ... vnamek
+ x1,y1 <x2,y2 ... >
+ <MIN=val> <MAX=val> <ABS=1> <SCALE=val> <TC1=val> <TC2=val>
+ <DELTA=val> <SMOOTH=val> <IC=val>
```

Delay Element

```
Hxxx n+ n- <CCVS> DELAY vname1 TD=val
+ <SCALE=val> <TC1=val> <TC2=val> <NPDELAY=val>
```

Hxxx	Current-controlled voltage source names begin with H.
n+, n-	Positive and negative terminal node names of the controlled source.
CCVS	Parameter for current controlled voltage source. Should not be used as a node name.
vname1,...	Names of voltage sources through which the controlling current flows. Their number is equal to ndim or k. Specify one name for each dimension or gate. The direction of the positive controlling current flow is from the positive node, through the source, to the negative node of vname.

transresistance	Current to voltage conversion factor.
MIN	Minimum value of controlled voltage. The condition Vout >= MIN is not effective unless the parameter MIN is specified.
MAX	Maximum value of controlled voltage. The condition Vout <= MAX is not effective unless the parameter MAX is specified
ABS=1	Output is absolute value of controlled voltage. Default is ABS=0.
SCALE	Scaling factor. Default is SCALE=1.
TC1,TC2	First- and second-order temperature coefficients. Default is TC1=0, TC2=0.

SCALE, TC1 and TC2 are involved in calculation output voltage through:

```
Vout = V*SCALE*(1. + TC1*dT + TC2*dT*dT),
```

where *dT* = temperature - t_nominal.

IC	Initial condition. The initial estimate of the value(s) of the controlling current(s). Default is IC=0.
POLY	Polynomial function parameter. Should not be used as a node name.
ndim	The number of dimensions of the polynomial.
p0, p1,...	Polynomial coefficients.
PWL	Piecewise linear function parameter. Should not be used as a node name.
x1,y1 x2,y2...	Pairs of values. The first value in each pair (x) is the controlling current through vname1, vname2,..., and the second value (y) is corresponding output. The x values must be in increasing order.
DELTA	Smoothing distance from the corner. Default is DELTA=0. If DELTA is specified, DELTA is limited by 1/2 of the smallest distance between x axis points.
SMOOTH	Selects the curve smoothing method. Unused and was introduced only for compatibility with the syntax of other SPICEs.
gatetype(k)	AND, NAND, OR and NOR parameters set the type of multi-input gate device. The value of k is the number of inputs. AND, NAND, OR and NOR should not be used as node names. For AND and NAND cases, the controlling gate is chosen on the basis of the smallest controlling value, while for OR or NOR, the largest controlling value is used. Inversion for NAND and NOR is provided by the character of the user-specified PWL function (but not by the parameters NAND or NOR).

DELAY	Delay element parameter. Should not be used as a node name. The delay element is the same as a current controlled voltage source: `Vout(t) = SCALE * (1. + TC1 * dT + TC2 * dT*dT) * Iin(t - TD).`
TD	Time delay.
NPDELAY	Number of time points to be saved during Transient analysis for DELAY simulation. The value must be the larger of 10 and the smaller of TD/TSTEP and TSTOP/TSTEP: `NPDELAY = max(10, min(TD,TSTOP)/ TSTEP, NPDELAY_user)` The value of TSTOP and TSTEP are specified in the .TRAN statement. NPDELAY_user is the user-specified value of NPDELAY. Default is NPDELAY_user=10.

Output parameters are accessed by using the syntax: `@H-devicename[paramname]`.

Current through H-device can be accessed by using syntax: `i(H-devicename)`.

Examples

1. Linear CCVS, connected between nodes 5 and 17, with a trans resistance of 0.5k, and with the controlling current flowing through the independent voltage source vz. Its voltage is *v(5,17) = 0.5k * i(vz)*:

```
H1  5  17  vz  0.5k
```

2. Nonlinear CCVS whose voltage is a polynomial function of the currents through the independent voltage sources vq1, vq2, and vq3 such that:

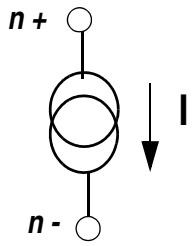
```
v(7,8) = i(vq1) + i(vq2) + i(vq3) + i(vq1)*i(vq3) + i(vq2)*i(vq3)
H02  7  8  POLY(3)  vq1  vq2  vq3  0  1  1  1  1  0  0  1  0  1
```

3. Piecewise linear CCVS whose voltage is a function of the current through the independent voltage source va:

```
H3  9  10  PWL(1)  va  -5,0  -2,2.5  1.2,5.9
```

A detailed description is provided in the SPICE Models Manual.

I (Independent Current Source)



Syntax

```
Ixxx n+ n- <<DC> dcval> <R=tval> <REP=repval> <transrc>
+ <AC> <acmag <acphase>>>
+ <DISTOF1 <f1mag <f1phase>>> <DISTOF2 <f2mag <f2phase>>>
+ <M=val> <PWLFAST=1>

Ixxx n+ n- <DC1=dc1val DC2=dc2val> <R=tval> <REP=repval> <transrc>
+ <AC <acmag <acphase>>>
+ <DISTOF1 <f1mag <f1phase>>> <DISTOF2 <f2mag <f2phase>>>
+ <M=val> <PWLFAST=1>

Ixxx n+ n- PAT <(> vhi vlo td tr tf tsample data <RB=val>
+ <R=repeat> <)>

Ixxx n+ n- PAT <(> vhi vlo td tr tf tsample
+ [component 1 ... component n]
+ <RB=val> <R=repeat> <)>

Ixxx n+ n- LFSR <(>vlow vhigh tdelay trise tfall rate seed <taps>
+ <rout=val><)>
```

Ixxx	Independent current source- names begin with `I`.
n+, n-	Positive and negative node names.
DC	Parameter for the DC source value.
dcval	DC value of the source. SmartSpice uses this value during any DC analysis phase. SmartSpice also uses `dcval` during the Transient analysis phase if `transrc` is not specified. Default is `0`.
DC1=dc1val, DC2=dc2val	Specifies the initial and pulse values of the source. SmartSpice changes the DC value of the source from `dc1val` to `dc2val` during DC transfer curve calculation. If `transrc` is not specified, then SmartSpice uses `dc1val` as a DC value during the transient analysis.

R=tval	Specifies the timepoint from which the source waveform should be repeated. The section of the waveform between `tval` and the end of its specification will be repeated until the transient analysis is completed. This parameter is used only in conjunction with the `transrc` parameter.
REP=repval	Specifies the length of the source waveform, counting backward from the end, to be repeated. For example, if a transient waveform has its endpoint at `t=3n` and `REP=0.4n`, then SmartSpice will repeat the section of the waveform from `2.6n` to `3n`, until the simulation is completed. This parameter is used only in conjunction with `transrc` parameter.
transrc	Transient analysis source specification. It can be one of the following types:

- `AM` - amplitude modulation:
  `AM <(> <sa <offset <fm <fc <td>>>>> <)>`
- `EXP` - exponential:
  `EXP (i1 i2 <td0 <tau0 <td1 <tau1>>>>)`
- `PL` - piecewise linear pairs of (current, time):
  `PL(i1 t1 <i2 t2 <i3 t3<...>>> <R=tval><TD=val>)`
- `PULSE` - pulse:
  `PULSE (i1 i2 <td <tr <tf <pw <per>>>>)`
- `PWL` - piecewise linear pairs of (time, current):
  `PWL (t1 i1 <t2 i2 t3 i3 ... tk ik> <R=tval> <TD=val>)`
- `PWLFILE` - piecewise linear from the specified file:
  `PWLFILE filename <TD=val>`
- `PWLFILEDESC` - read from specified file as series of (time, current) pairs:
  `PWLFILEDESC(1 start dt) PWLFILE filename`
- `PWLFILEDESC` - read from specified file as series of uniformly separated values:
  `PWLFILEDESC(2 start dt ilow ihigh fall_time rise_time) PWLFILE filename`
- `PWLFILEDESC` - read from specified file as series of (time, current) pairs:
  `PWLFILEDESC (3) PWLFROMRAWFILE filename PWLFROMRAWVECTOR vectorname.`
- `SFFM` - single-frequency:
  `SFFM (io ia <fc <mdi <fs>>>)`
- `SIN` - sinusoidal:
  `SIN (io ia <freq <td <theta <phase>>>>)`

SmartSpice uses the transient source values during Transient analysis. Initial transient source value at time 0 will be used for any DC analysis phase when neither `DC` nor `dcval` is specified.

The number of points in a PWL waveform are unlimited.

filename	Full name of the file contains pairs of values making up the piecewise linear waveform. filename can be the parameter that is set in the subcircuit's arguments, or in the X-call of the subcircuit.

```
xI1 NODE_1 0  5 6  CUR  PARAMS: PATHTOVALUES1 ='stimulus1.dat'
.subckt CUR 1  2  3  4  PARAMS:   PATHTOVALUES2 =' stimulus2.dat'

I1   1   2   pwlfile  PATHTOVALUES1
I2   3   4   pwlfile  PATHTOVALUES2
.ends
```

In the case of the same names of the parameter which point out on the file name path in the X-call and subcircuit definition, the name defined in the X-call will have higher priority than in the subcircuit definition.

TD=val	Time delay. Allows you to shift PWL source waveform on time equal val. Default is 0.
AC	Parameter for the AC source value.
acmag	AC magnitude. Default is 1.
acphase	AC phase. Default is 0.
DISTOF1, DISTOF2	Keywords that specify that the independent source has distortion inputs for two values.
f1mag, f2mag	Magnitudes. Defaults are 0.
f1phase, f2phase	Phases. Defaults are 0.
M=val	Multiplier. Default is 1.
PWLFAST=1	Set to 1 to suppress unnecessary reloading of PWL data between runs. Default is 0.

Examples

```
I1 1 0 DC 1MA
I2 1 0 DC=-1MA AC 2 30
I3 1 0 DC1=-1MA DC2=-2MA PULSE (-1MA -2MA 1NS 2NS 1NS 10NS 20NS)
```

The first example describes the current source connected between node 1 and node 0, with a DC current value of 1mA.

The second example describes a current source with a DC value of -1mA, an AC source with a magnitude of 2, and a phase of 30 degrees.

The third example describes an initial DC1 source with the value of -1mA, a pulse DC2 with a value of -2mA, and a pulse waveform.

Pseudo Random-bit LFSR Function

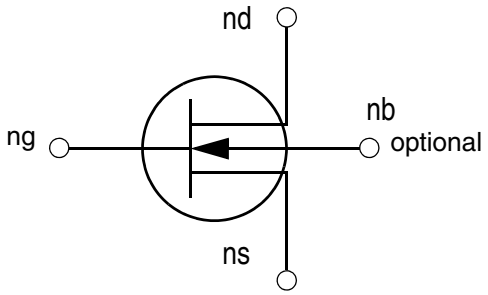
LFSR	Specifies the voltage/current source as PRBS.
vlow	The low voltage/current level.
vhigh	The high voltage/current level.
tdelay	The time delay.
trise	The duration (in seconds) from the low value to the high value.
tfall	The duration (in seconds) from the high value to the low value.
rate	The bit rate.
seed	The initial value (in integer form) of the shift register.
taps	The bits used to generate feedback. Must range from 2 to 64.
rout	The output resistance.

Example

```
I1 1 0 LFSR (0 1 1u 1n 1n 10meg 3 [2, 5] rout=5)
```

For details, see the SPICE Models Manual.

J (JFET/MESFET)



Syntax

```
Jxxx nd ng ns <nb> mname <area> <M=val> <OFF> <IC=vds,vgs>
+ <TEMP=val> <DTEMP=val> <L=val> <W=val>
```

or

```
Bxxx nd ng ns <nb> mname <area> <M=val> <OFF> <IC=vds,vgs>
+ <TEMP=val> <DTEMP=val> <L=val> <W=val>
```

or

```
Zxxx nd ng ns <nb> mname <area> <M=val> <OFF> <IC=vds,vgs>
+ <TEMP=val> <DTEMP=val> <L=val> <W=val>
```

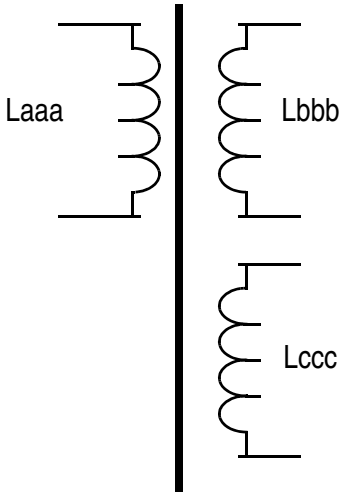
Jxxx (Bxxx, Zxxx)	JFET/MESFET element name must begin with J, B or Z.
nd, ng, ns, nb	Drain, gate, source and bulk terminal node names. The bulk node name is optional. If unspecified, the bulk node is connected to the source node.
mname	Model name. Must reference a JFET/MESFET model.
area	Area factor. Default is 1.0.
M	Multiplier used to describe multiple parallel JFETs/MESFETs. Default is 1.
OFF	Sets ON/OFF starting condition for DC analysis. Default is ON.
IC	Initial condition specification for vds and vgs.
TEMP	Device operating temperature.
DTEMP	Difference in degrees C between the device operating temperature and the circuit temperature. Default is 0.
L	Length of the gate in meters.
W	Width of the gate in meters.

Examples

```
J44 1 4 6 jmodel
```

See the SPICE Models Manual for a complete description of JFET/MESFET models.

K (Coupled Mutual Inductor)



Syntax

```
Kxxx Laaa Lbbb <Lccc ...> kvalue <mname>
```

Kxxx	Mutual inductor element, name must begin with a K.
Laaa, Lbbb, Lccc, ...	The names of two or more coupled inductors.
kvalue	The mutual coupling coefficient. The allowed range of values for kvalue is between -1.0 and +1.0. If kvalue is negative, the direction of coupling is reversed.
mname	Magnetic core model name. Indicates if the specified magnetic core model should be used instead of the simple coupled inductor model.

Examples

```
L1 1 0 50uH; dot on node 1
L2 2 0 50uH; dot on node 2
K1 L1 L2 0.999
```

In this example, the mutual inductor K1 defines the coupling between L1 and L2.

For a description of the magnetic core model, see the SPICE Models Manual.

L (Inductor)

Syntax

Linear

```
Lxxx n+ n- <mname> <L | IND=>lval <M=val> <R=val>
+ <SCALE=val> <TC1=val> <TC2= val> <TEMP=val> <DTEMP=val> <IC=val>

Lxxx n+ n- <mname> <L | IND=>lval <tc1val <tc2val> >
+ <M=val> <R=val> <SCALE=val> <TEMP=val> <DTEMP=val> <IC=val>
```

Polynomial

```
Lxxx n+ n- POLY p0 <p1 ...> <M=val> <R=val>
+ <SCALE=val> <TC1=val> <TC2=val> <TEMP=val> <DTEMP=val> <IC=val>
```

Expression

```
Lxxx n+ n- <L=>‘expression’ <M=val> <R=val>
+ <SCALE=val> <TC1=val> <TC2=val> <TEMP=val> <DTEMP=val> <IC=val>

Lxxx n+ n- <L=>‘expression’ <tc1val <tc2val> >
+  <M=val> <R=val> <SCALE=val> <TEMP=val> <DTEMP=val> <IC=val>
```

Table

```
Lxxx n+ n- L=‘expression’ TABLE x1,y1 <x2,y2, ... > <M=val>
+ <R=val> <SCALE=val> <TC1=val> <TC2=val> <TEMP=val> <DTEMP=val>
+ <DELTA=val> <IC=val>

Lxxx n+ n- L=‘expression’ <tc1val <tc2val> > TABLE x1,y1
+ <x2,y2, ... > <M=val> <R=val> <SCALE=val> <TEMP=val> <DTEMP=val>
+ <DELTA=val> <IC=val>
```

When mname is specified, the inductor value lval specification is optional. If there is a magnetic core model associated with the element, then the element specifies the transformer is winding rather than an inductor.

Lxxx	Inductor element name. This name must begin with the letter L.
n+, n-	Positive and negative terminal node names.
mname	Model name. Must reference a inductor model.
lval	The value of the inductance. Zero is a legal value corresponding to an open circuit. For transformers, this specifies the number of turns. If lval is specified as a parameter label and a inductor model exists with the same name, the value is interpreted as a model name.
M	Multiplier used to simulate parallel devices. Default is 1.

R	Resistance of inductor. Default is 0.0.
SCALE	Scaling multiplier. The default is the inductor model parameter SCALE.
TC1, TC2	First and second order temperature coefficients, respectively. The default TC1, TC2 are the inductor model parameters TC1, TC2. TC1, TC2 parameters can be specified by optional syntax tc1val <tc2val> (without the parameters TC1, TC2). In this case the values tc1val, tc2val must follow immediately after required <L|IND=>lval or <L=>'expression' inductance specification. SCALEeff = SCALE * (1 + TC1*dT + TC2*dT^2), where *dT* = temperature - t_nominal.
TEMP	Device operating temperature in degree C. The default value is set by the option TEMP of the .OPTIONS statement.
DTEMP	Difference between inductor instance temperature and circuit temperature in degrees C. Defaults to 0.0. If the device operating temperature is specified, then the default value is DTEMP=TEMP-device-TEMPcircuit.
IC	Initial current through the inductor. Default is 0.0.
POLY	Polynomial function keyword. Should not be used as a node name.
p0 <p1...>	Polynomial coefficients. The inductance is described as a polynomial function of the instantaneous current through the inductor: L = p0 + p1 * Iind + p2 * Iind^2 + p3 * Iind^3 + ...
expression	Function of algebraic and logical expressions; real constants; parameter labels; circuit temperature (TEMP); integration time (TIME); integration timestep (TSTEP); frequency (HERTZ); node voltages; voltages across two nodes; branch currents - independent currents through: • independent voltage sources (V), • controlled voltage sources (E and H), • analog behavioral devices (A) with V-type expressions and inductors (L); operator DER; functions DERIVATIVE and INTEGRAL; user-defined functions.
TABLE	Keyword used to indicate that the inductance will be found by using a lookup table.

The expression immediately following the parameter L before TABLE or tc1val <tc2val> is evaluated and used to find a value in the table of (x,y) pair. If the points are not

found in the table, they will be calculated by linear interpolation. At least two pairs of values must be defined in the lookup table.

x1,y1, x2,y2,...	Pairs of values. The first value in each pair (x) is the evaluated controlling value, and the second value (y) is the corresponding output. The x values must be in increasing order.
DELTA	Smoothing distance from the corner of piecewise linear function described pairs of values x1,y1 x2,y2,.... Default is DELTA=0. If DELTA is specified, it is limited by 1/2 of the smallest distance between x axis points.

Examples

1. Inductor connected between nodes 12 and 0, with an inductance of 220nH:

```
LSERIAL  12  0  220nH
```

2. Inductor connected between nodes 7 and 8, with inductance 400nH, and with initial current 3.2mA:

```
L2  7  8  400nH   IC=3.2mA
```

3. Inductor connected between nodes 3 and 7, with an operating temperature of 100 degrees C, with inductance of 100nH, and a model name lmodel:

```
L11  3   7   lmodel   L=100n  TEMP=100
```

4. Inductor connected between nodes 1 and 0 with inductance varying with time:

```
L1  1  0  l='200n * (1. + 1n * time)'
```

5. Inductor connected between nodes 1 and 0 with inductance varying with frequency:

```
L2  1  0  L='200nH * (1. + 1e-9*HERTZ)'
```

6. Inductor connected between nodes 1 and 0 with inductance varying using lookup table with the variable time:

```
L3  1  0  L='time'  TABLE  0,1000nH  3ns,1000nH  3.1ns,10nH
```

7. Nonlinear inductor connected between nodes 1 and 2 with inductance described by the algebraic expression:

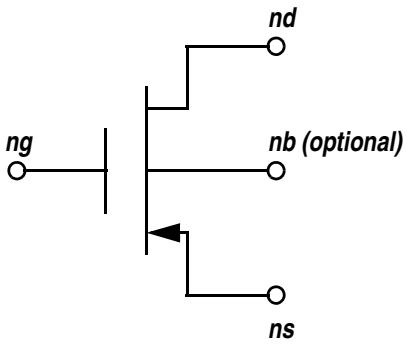
```
Lexpr  1  2  l='100n * (1. + 100. * I(lexpr) + 1000. * I(lexpr)^2)'
```

8. Inductor connected between nodes 1 and 0 with inductance described by the logical expression:

```
Llog  1  0  l='   (IF V(1,0) < 2  THEN 100nH  ELSE 200nH*V(1,0 )  )
+               +(IF V(1,0) > 3  THEN 300nH  ELSE 100nH*V(1,0)  )'
```

A detailed description is provided in the SPICE Models Manual.

M (MOSFET)



Syntax

```
Mxxx nd ng ns <nb> mname <lval|L=val> <wval|W=val> <AD=val>
+ <AS=val> <PD=val> <PS=val> <NRD=val> <NRS=val> <OFF>
+ <IC=vds,vgs,vbs> <TEMP=val> <DTEMP=val> <M=val> <GEO=val>
+ <DELVTO=val> <RTH=val> <CTH=val> <NOSELFT=val> <NS=val>
```

Mxxx	MOSFET element name, must begin with an M.
nd, ng, ns, nb	Drain, gate, source, and bulk terminal node names. Bulk node specification is optional. If the bulk node name is omitted, the name specified in the parameter BULK is used. It must be used if the bulk node name is omitted.
mname	Model name must reference a MOSFET model.
L=val, lval	Channel length in meters. Default is 100 µm.
W=val, wval	Channel width in meters. Default is 100 µm.

Note: Using W, L on the .OPTIONS statement reverses the order of lval and wval.

AD, AS	Drain and source diffusion junction area (meters^2). Defaults are 0.
PD, PS	The perimeters of the drain and source diffusion junctions in meters. Defaults are 0.
NRD, NRS	The number of squares of drain and source diffusion for resistance calculations. Defaults are 0.
OFF	Sets ON/OFF startup condition for DC analysis. Default is ON.

IC	Initial voltage condition specification for vds, vgs and vbs.
TEMP	Device operating temperature in degrees C.
M	Multiplication factor (for multiple device simulation). Default is 1.
GEO	Accounts for shared drain/source geometry. Default is 0.

Note: The parameter GEO=val is not implemented in BSIM4.

DELVTO	Threshold voltage shift. When specified on the device line, the value overrides the value of the model parameter DELVTO. If not specified, the value of the model parameter is used.
DTEMP	Difference in degrees C between the device operating temperature and the circuit temperature. Default is 0.
RTH	Thermal resistance (degrees C/W). Default value is the model parameter RTH0 (LEVEL=35, and 36 0nly).
CTH	Thermal capacitance (W.s/degrees C). The default value is the model parameter CTH0 (LEVEL=35, and 36 0nly).
NOSELFT	Selector to de-activate self-heating at the device level. It overrides the parameter SHMOD. Default is 0 (LEVEL=35, and 36 0nly).
NS	Multiple devices in series (EKV model only).

Examples

```
M1 2 4 8 9 mod1
Mout2 19 20 21 0 nmos L=5u W=2u TEMP=50
M22 3 5 7 8 mosmod1 L=10u W=5u AD=150p AS=150p
+ PD=50u PS=50u NRD=10 NRS=20
```

For a complete description of MOSFET models, see the SPICE Models Manual.

N (Independent Noise Current Source)

An independent Noise Current Source can be specified to define spectral density level of white (thermal and shot) or/and flicker (1/f) noise placed at the terminal nodes.

Noise sources are accounted for `.NOISE` and `.TRAN NOISE` analyses only, and is not used during `.DC`, `.AC`, `.OP` and transient analyses.

Noise calculations assume a 1Hz bandwidth. The total spectral density of noise source is defined as:

$$S = SW + \frac{SFL}{f^{AF}}$$

Syntax

```
Nxxx n+ n- <<SW=> swval > <SFL=val> <AF=val> <M=val> <DTEMP=val>
```

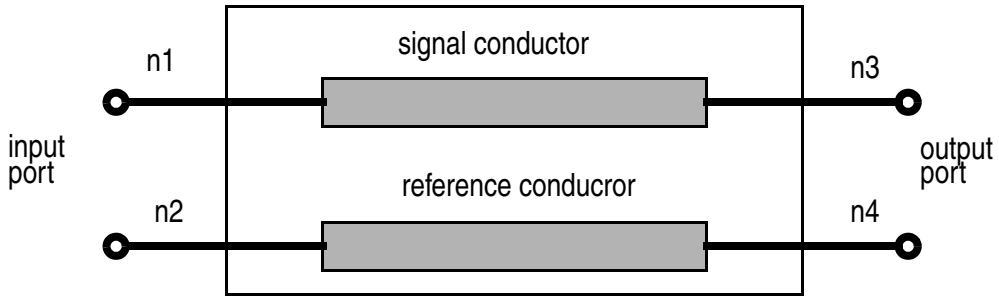
Nxxx	Noise Source element name. This name must begin with the letter `N`.
n+, n-	Positive and negative terminal node names.
SW=swval	Spectral density of white noise (in A^2/Hz). If `SW` is specified as a parameter label, and a noise source model exists with the same name, the value is interpreted as a model name.
SFL=val	Spectral density of flicker noise (in A^2/Hz). Default is `0`.
AF=val	Frequency power coefficient for Flicker noise calculation. Default is `1`.
M=val	Multiplier used to simulate parallel sources. Default is `1`.
TEMP=val	Noise source operating temperature (in degrees C).
DTEMP=val	Difference between Noise Source instance temperature and circuit temperature (in degrees C).

Noise current source produces three noise vectors: total, 1/f (`.1overf`), and white (`.wt`).

Examples

1. Noise source connected between nodes `2` and `0` with a white noise spectral density level of *5.e-12 A^2/Hz*:
   ```
   Noise1 2 0 5p
   ```
2. Noise source connected between nodes `in` and `3` with a white noise spectral density of *5e-12 A^2/Hz* and flicker noise spectral density of *2.e-12 A^2/Hz*:
   ```
   N2 in 3 SW=5p SFL=2p AF=1.2
   ```

O (Lossy Transmission Lines–Berkeley SPICE Model)



Syntax

```
Oxxx n1 n2 n3 n4 mname
+ <L | LEN | LENGTH=val> <IC=v1, i1, v2, i2> <M=val>
```

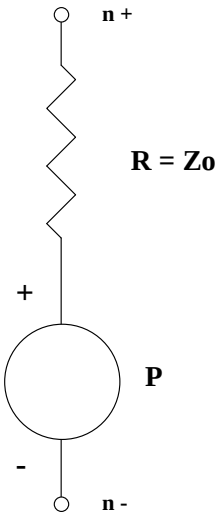
Oxxx	The lossy uniform transmission line element name begins with O.
n1, n2	The nodes at port 1.
n3, n4	The nodes at port 2.
mname	Model reference name.
L (or **LEN, LENGTH)**	Physical length of the transmission line (meters). Default is L=1m.
M	Multiplier used to simulate parallel devices. Default is M=1.
IC	Initial condition specification, which consists of the voltage and current at each of the transmission line ports. The initial conditions apply only if the option UIC is specified on the .TRAN statement.

Examples

```
O1  1 0 2 0  tlline  l=.6095
ONET  1 2 3 2  connect  length=.16
```

See the SPICE Models Manual for a description of the Lossy Transmission Line model.

P (Resistive Source Port)



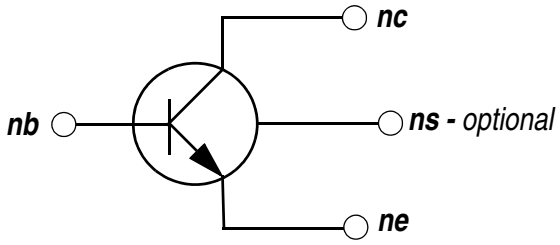
Syntax

```
Pxxx n+ n- <DC=dcval> <PORT=val> <transrc>
+ <AC <acmag <acphase>>>
+ <Z0=val> <RDC=val> <RAC=val> <RTRAN=val>
+ <POWER=[0|1|2|W|dbm]
```

Pxxx	Port names begin with P.
n+, n-	Positive and negative node names.
DC=dcval	DC value of the source. SmartSpice uses this value during any DC analysis phase. SmartSpice also uses dcval during the Transient analysis phase if `transrc` is not specified. `dcval` has a default value of 0.
PORT=val	The port number. Ports must be numbered when used for `.LIN` analyses. Numbering should go sequentially, beginning with 1, with no shared port numbers and no gaps.
transrc	Transient analysis source specification. Currently only `SIN` is supported: • `SIN`: sinusoidal; `SIN (vo va <freq <td <theta <phase>>>>)`
AC	Keyword for the AC source value.
acmag	AC magnitude. Default is 1.
acphase	AC phase. Default is 0 degrees.

Z0=val	Port resistance, inserted in series with the voltage source. Only real impedance is supported. Default is 50 ohms.
RDC=val	DC analysis series resistance (overrides Z0).
RAC=val	AC analysis series resistance (overrides Z0).
RTRAN=val	Transient analysis series resistance (overrides Z0).
POWER=[0|1|2|W|dbm]	Power switch: • 0: Element is a voltage source • 1 or W: Element is a power source with a series impedance. Source value is a RMS available power in units of Watts. • 2 or dbm: Element is a power source with a series impedance. Values are in dbms.

Q (Bipolar Junction Transistor – BJT)



Syntax

```
Qxxx nc nb ne <ns> <t1 <t2>> mname <area> <OFF> <IC=vbe,vce>
+ <TEMP=val> <M=val> <DTEMP=val>
```

or

```
Qxxx nc nb ne <ns> <t1 <t2>> mname <area=val> <areab=val>
+ <areac=val> <OFF> <IC=vbe,vce> <TEMP=val> <DTEMP=val> <NOSELFT>
+ <TNODEOUT> <DTMAX=val> <M=val>
```

Qxxx	BJT element name. Must begin with Q.
nc, nb, ne, ns	Collector, base, emitter, and substrate terminal node names. The substrate terminal node name is optional. If it is unspecified, ground is used.
t1, t2	Names of optional thermal terminals. Due to the fact that the substrate name is also optional, a new device input flag TNODEOUT has been introduced. Optional terminal names are assigned as follows:

1. If only 4 terminals are given:
   - if the flag TNODEOUT is NOT given:

     terminal #4 is assumed to be ns (t1 and t2 are unspecified).

   otherwise

     terminal #4 is assumed to be t1 (ns and t2 are unspecified).
2. If 5 terminals are given:
   - if the flag TNODEOUT is NOT given:

     terminal #4 is assumed to be ns,

     terminal #5 is assumed to be t1 (t2 is unspecified).

   otherwise

     terminal #4 is assumed to be t1,

     terminal #5 is assumed to be t2 (ns is unspecified)
3. If the 6 possible terminals are given (TNODEOUT is ignored):

     terminal #4 is assumed to be ns,

terminal #5 is assumed to be `t1`,

terminal #6 is assumed to be `t2`.

In any case, if ns is unspecified, the substrate is internally connected to the global node specified by `BULK`, or to the ground if `BULK` is not given.

`mname`	Model name. Must reference a BJT model.
`area`	Emitter area factor. Default is `1.0`.
`areab`	Base area factor. Default is `area`.
`areac`	Collector area factor. Default is `area`.
`OFF`	Sets `ON/OFF` startup condition for DC analysis. Default is `ON`.
`IC`	Initial condition specification.
`TEMP`	Device operating temperature (degrees C).
`DTEMP`	Difference in degrees C between the device operating temperature and the circuit temperature. Default is `0`.
`NOSELFT`	If specified, this flag overrides the corresponding model flag `SELFT`. It allows you to switch off self-heating calculation for a particular device (LEVEL=4, 5, 504 (level=6 vers=504), 8 and 20 only).
`TNODEOUT`	Flag to specify that the first optional terminals are thermal nodes (the default substrate is used). Ignored if all terminals are given on the device line (LEVEL=4, 5, 504 (level=6 vers=504), 8 and 20 only).
`DTMAX`	Maximum temperature rise above heatsink. If specified, its value overrides the value of the corresponding model parameter `DTMAX`. Most often, instances of a given model have different internal temperatures. If differences are important, it is necessary to adjust the temperature-rise limitation for each device to help convergence. Default is `1000` (LEVEL=4, 5, 504 (level=6 vers=504), 8 and 20 only).
`M`	Multiplication factor for parallel devices. Default is `1`.

Note: `DTMAX` is used only in self-heating calculation.

Examples

```
Q1 2 3 9 npnmod 1.5 IC=0.6,5.0
Q9 10 11 12 20 mod22 OFF TEMP=50
```

For a description of the BJT model, see the SPICE Models Manual.

R (Resistor)

Syntax

Linear

```
Rxxx n+ n- <mname> <<R | RES=>resval > <M=val> <L=val> <W=val>
+ <SCALE=val> <TC1=val> <TC2= val> <TC=val1, val2> <DTEMP=val>
+ <AC=val> <C=val> <RSK=val> <RNOISE=val> <NOISE=val>

Rxxx n+ n- <mname> <R | RES=>resval <tc1val <tc2val> > <M=val>
+ <L=val> <W=val> <SCALE=val> <DTEMP=val> <AC=val> <C=val>
+ <RSK=val> <RNOISE=val> <NOISE=val>
```

Polynomial

```
Rxxx n+ n- POLY p0 <p1 ...>
+ <M=val> <SCALE=val> <TC1=val> <TC2=val> <TC=val1, val2>
+ <DTEMP=val> <AC=val> <RNOISE=val> <NOISE=val>
```

Expression

```
Rxxx n+ n- <mname> <R=>‘expression’
+ <M=val> <SCALE=val> <TC1=val> <TC2=val> <TC=val1, val2>
+ <DTEMP=val> <AC=val> <RNOISE=val> <NOISE=val>

Rxxx n+ n- <R=>‘expression’ <tc1val <tc2val> >
+ <M=val> <SCALE=val> <DTEMP=val> <AC=val> <RNOISE=val> <NOISE=val>
```

Table

```
Rxxx n+ n- R=‘expression’ TABLE x1,y1 <x2,y2, ... > <M=val>
+ <SCALE=val> <TC1=val> <TC2=val> <TC=val1, val2> <DTEMP=val>
+ <DELTA=val> <AC=val> <RNOISE=val> <NOISE=val>

Rxxx n+ n- R=‘expression’ <tc1val <tc2val> > TABLE x1,y1
+ <x2,y2, ...> <M=val> <SCALE=val> <DTEMP=val> <DELTA=val> <AC=val>
+ <RNOISE=val> <NOISE=val>
```

When mname is specified, the resistor value resval specification is optional.

Rxxx	Resistor element name. This name must begin with the letter R.
n+, n-	Positive and negative terminal node names.
mname	Model name. Must reference a resistor model.
resval	The value of resistor. If resval is specified as a parameter label, and a resistor model exists with the same name, the value is interpreted as a model name.
M	Multiplier used to simulate parallel resistors. Default is 1.

L	Resistor length. Default is the resistor model parameter L(DEFL).
W	Resistor width. Default is the resistor model parameter W(DELW).
SCALE	Scaling multiplier. The default is the resistor model parameter SCALE.
TC1, TC2	First and second order temperature coefficients, respectively. Defaults for TC1, TC2 are the resistor model parameters TC1, TC2. TC1, TC2 parameters can be specified by the optional syntax tc1val <tc2val> (without keywords TC1, TC2). In this case the values tc1val, tc2val must follow immediately after the required <R|RES=>val or <R=>'expression' resistance specification. The optional syntax TC=val1, val2 can be used instead of TC1, TC2. SCALEeff = SCALE * (1 + TC1*dT + TC2*dT^2), where *dT* = temperature - t_nominal.
DTEMP	Difference between resistor instance temperature and circuit temperature, in degrees C.
AC	AC resistance. Default is 0.
C	Capacitance connected between node n-, and the node specified by the resistor model parameter BULK. By default, BULK is ground. Default is C=0.
RSK	Skin-effect resistance. Default is 0. If RSK is specified: $R = RES + RSK \times \sqrt{FREQ}$

For DC analysis *FREQ* = 0. For Transient analysis SmartSpice uses *FREQ = 1. / (15 * risetime)*, where *risetime* is the parameter RISETIME, which is specified by the .OPTIONS statement. If *risetime* option is used see RISETIME=val on page 547. You can control the skin-effect frequency choice with the parameter RISETIME.

RNOISE=val	The value of resistor for thermal noise calculation. Default is AC value.
NOISE=val	Resistor instance noise coefficient. Default is 1.
POLY	Polynomial function parameter. Should not be used as a node name.
p0 <p1...>	Polynomial coefficients. The conductance is described as a polynomial function of the voltage across the resistor: R = 1. / (p0 + p1 * V(n+,n-) + p2 * V(n+,n-)^2 + p3 * V(n+,n-)^3 + ...)

expression	Function of algebraic and logical expressions; real constants; parameter labels; circuit temperature (TEMP); integration time (TIME); integration timestep (TSTEP); frequency (HERTZ); node voltages; voltages across two nodes; branch currents - independent currents through: • independent voltage sources (V), • controlled voltage sources (E and H), • analog behavioral devices (A) with V-type expressions and inductors (L); operator DER; functions DERIVATIVE and INTEGRAL; user-defined functions.
TABLE	Keyword used to indicate that the resistance will be found by using a lookup table.

The expression immediately following the parameter R before TABLE or tc1val <tc2val> is evaluated, and used to find a value in the table of (x,y) pairs. If the points are not found in the table, they will be calculated by linear interpolation. At least two pairs of values must be defined in the lookup table.

x1,y1, x2,y2,...	Pairs of values. The first value in each pair (x) is the evaluated controlling value, and the second value (y) is the corresponding output. The x values must be in increasing order.
DELTA	Smoothing distance from the corner of piecewise linear function described pairs of values x1,y1, x2,y2,.... Default is DELTA=0. If DELTA is specified, it is limited by 1/2 of the smallest distance between x axis points.

When resval is specified, it defines the resistance. When mname is specified, the resistance is calculated using the process information in the model mname and the given values for L and W. If resval not specified, mname and L,W must be specified.

Examples

1. Resistor Rload connected between nodes 2 and 10 with value 10K:

```
Rload  2  10   10K
```

2. Resistor Rmod connected between nodes 3 and 7 with model name Rmodel1 and a length of 10 microns and a width of 1 micron:

```
Rmod  3  7  Rmodel1  l=10u  w=1u
```

3. Resistor Rmix connected between nodes 1 and 2 with model name RMODEL, resistance value 10K, and temperature coefficients 0.01 and 0.001:

```
Rmix  1  2  RMODEL  r=10k  TC1=0.01  TC2=0.001
```

4. Resistor Rt connected between nodes 1 and 2 with model name RMODEL, resistance value 10K, temperature coefficients 0.01 and 0.001, AC resistance 1K:

```
Rt  1  2  RMODEL  10k  0.01  0.001  AC=1K
```

5. Nonlinear resistor Rexpr connected between nodes 1 and 2 with resistance described by an algebraic expression:

```
Rexpr  1  2  R='10k * (1. + 0.1 * V(1,2) + 0.01 * V(1,2)^2)'
```

6. Resistor Rl connected between nodes 1 and 0 with resistance described by a logical expression:

```
Rl  1  0  R='   (IF V(1,0) < 2  THEN 1K  ELSE 2K*V(1,0)  )
+             +(IF V(1,0) > 3  THEN 3K  ELSE 1K*V(1,0)  )'
```

7. Nonlinear voltage dependant resistor instance as shown by this input deck:

```
.param rmodel=1
+  coef1=1
+  coef2=1

Vin 1 0 PWL (0,1 4n,4)

r1 1 11 1
r2 1 22 1
r3 1 33 1

Rvr 11 0 res_vr_model
.model res_vr_model R res=rmodel vr1=coef1 vr2=coef2

Rcoef 22 0 res_coef_model
.model res_coef_model R res=rmodel coeffs=[coef1, coef2]

Rexpr 33 0 r='rmodel/(1.+ coef1*V(33)+ coef2*v(33)^2)'

.options nomod acct accurate
.tran 0.1n 4n

.print v(11) v(22) v(33)
+     @rvr[reff]  @rcoef[reff] @rexpr[reff]
.end
```

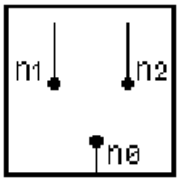
The value of the resistor can be specified as a function of branch currents.

Example

```
R x xx r='I1(M1)+ 2*I2(M1)'
```

A detailed description is provided in the SPICE Models Manual.

S (Ideal Switch, ISW)



Syntax

```
Sxxx n0 n1 ... nN idealswitch=mname
```

Sxxx	Ideal switch element name must begin with S.
n0 n1 ... nN	Node names; first node n0 is a common terminal.
idealswitch	Keyword; must be specified.
mname	Model name; must reference an ideal switch model.

The ideal switch is a device with multiple on positions and 1 off position(1-pole- multiple throw). Infinite off resistance is used for “off” position and zero for “on”. The switch helps to change circuit topology between analyses and not during the analysis. In the position 0 single-pole node is disconnected from others. Position 1 corresponds to connection between node 0 (pole) and node numbered by 1; position 2 - pole is connected to second node and disconnected from 1st node. Offset voltage is connected with minus to pole node and plus to “connected” node. The model parameter `position` defines position of the switch if `analysis_position` is not specified. The latter one overrides the `position` parameter.

Example

```
s_sw1 n_0 n_1 n_2 n_3 n_4 idealswitch=sw1_model_switch_0
.model sw1_model_switch_0 isw ( position=0
+ dc_position=3
+ ac_position=1
+ tran_position=4
+ ic_position=4 offset=7.5 m=4 )
```

Ideal Switch Model

Syntax

```
.MODEL mname ISW <parameter=val>
```

mname	Ideal Switch model name.
ISW	Specifies an ideal switch model.
parameter=val	Any ideal switch parameter.

Ideal Switch Model Parameters

Parameters	Description	Units	Default
position	Default switch position		0
ac_position	Position for AC analysis		position
dc_position	Position for DC analysis		position
tran_position	Position for TRAN analysis		position
ic_position	Position for IC analysis		position
offset	Offset voltage with common terminal	V	0
m	Multiplicity factor		1.0

S (Multi-Terminal Networks)

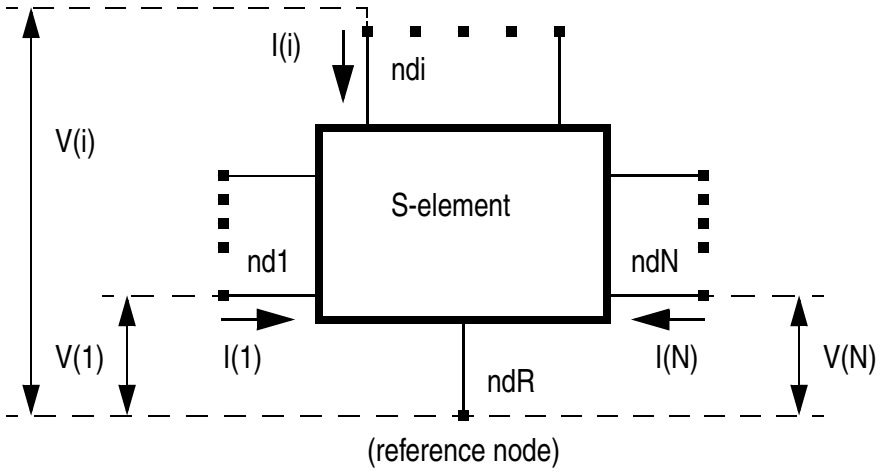
Syntax

```
Sxxx nd1 nd2 ... ndN <ndR>
+ <MNAME=mname | FQMODEL=SP_model_name>
+ <TYPE=S|Y>
+ <z0=zval|(zval1 <,zval2,...,zvalN>)>
+ <DELAYHANDLE=0|1|ON|OFF>
+ <DELAYFREQ=value>
+ <PASSIVE=pval> <PASSIVE_TOL=ptval>
```

S element, in conjunction with the frequency-domain model containing S- or Y-parameters, allows you to simulate AC behavior of multi-terminal network. The generic model equation is:

$$I = Y \cdot V$$

where I-terminal current vector, V-terminal voltage vector, and Y-characteristic admittance (Y-parameters) matrix:



When the S element model contains S-parameters, Y-parameters are calculated using the following formula:

$$Y = Z0^{-\frac{1}{2}} \cdot (I - S) \cdot (I + S)^{-1} \cdot Z0^{-\frac{1}{2}}$$

where Z0 is the characteristic impedance matrix of the reference system (which is typically assumed to be diagonal, and its diagonal values are set to the matching impedance z0), and I-unity matrix.

Note: S Element Parameter Description

nd1, nd2,...ndN	N signal nodes of an S-element.

ndR	References the node name. Default reference node (if not given) is GND.
MNAME	Model name. The model name must reference a “Multi-Terminal Network” model (SP).
FQMODEL	Defines name of a generic SP model data file, which contains frequency-dependent array of matricies.
TYPE=S|Y	Parameter type: S (scattering), Y (admittance). Default is S.
z0=zval|(zval1<,zval2,...,zvalN>)	Characteristic impedance value of the reference line (frequency-independent). For N >1 this is a diagonal characteristic impedance matrix. z0 can be the same for all terminals if only one value is specified, and different for all terminals if all N values are specified. Default is 50 Ohm.
DELAYHANDLE =0|1|ON|OFF	DELAYHANDLE activates extraction of frequency response delay before invFFT and S element equivalent model construction . Use of this parameter is recommended for transmission lines. Default is OFF|0.
DELAYFREQ=value	Sets frequency at which delays for transmission-line type parameters will be extracted The default is FMAX. If the DELAYHANDLE is set to OFF, but DELAYFREQ is nonzero, SmartSpice will activate DELAYHANDLE functionality. Default is FMAX.
PASSIVE=pval	Activates passivity checker if set to 1. Default is 0.
PASSIVE_TOL=ptval	Tolerance for passivity checker. Default is 0.01.

Note: A touchstone file of S parameters can be used with this device. For further information, see the SPICE Models Manual.

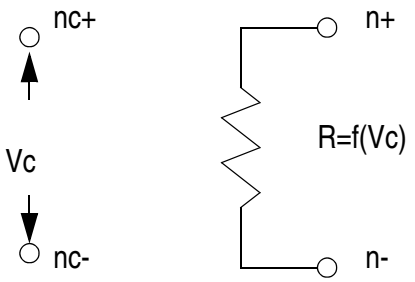
Example

```
S_w1 5 3 1 2 4 6 0   FQMODEL=w1 TYPE=Y
```

This example describes a multi-terminal network S_w1 with 6 terminals and grounded reference node. The network is defined by the model w1, which contains Y-parameter data points.

For a detailed description of “Multi-Terminal Network” model, see the SPICE Models Manual.

S (Voltage-Controlled Switch)



Syntax

```
Sxxx n+ n- nc+ nc- mname <ON|OFF> <M=val>
```

Sxxx	Voltage-controlled switch element name must begin with S.
n+, n-	Positive and negative terminal node names.
nc+, nc-	Positive and negative controlling node names.
mname	Model name. The model name must reference a voltage-controlled switch model.
ON, OFF	The initial condition of the switch. ON means that the switch is closed. OFF (default) means that the switch is open. This parameter is in effect only if the switch condition cannot be determined based on the controlling voltages.
M	Multiplier used to simulate parallel devices. Default is M=1.

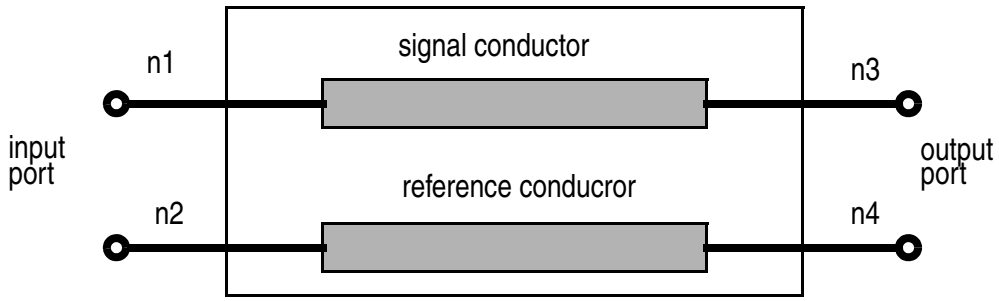
Example

```
S1 1 2 vcont 0 S1mod ON
```

This example describes a switch connected between nodes 1 and 2. The state of the switch is controlled by the voltage between nodes vcont and 0. The switch is defined by the model S1mod and is ON by default.

For a detailed description of the switch model, see the SPICE Models Manual.

T (Transmission Line, Lossless)



Syntax

```
Txxx n1 n2 n3 n4 Z0 | Zo=val TD=val <L | LEN | LENGTH=val>
+ <IC=v1, i1, v2, i2>   <M=val>

Txxx n1 n2 n3 n4 Z0 | Zo=val F=val <NL=val>
+ <IC=v1, i1, v2, i2> <M=val>

Txxx n1 n2 n3 n4 mname <L | LEN | LENGTH =val>
+ <IC=v1, i1, v2, i2> <M=val>
```

Txxx	The lossless transmission line element name begins with T, but should not be in the form of TXLxxx.
n1, n2	The nodes at input port 1.
n3, n4	The nodes at output port 2.
Z0 (or Zo)	Characteristic impedance (ohm).
TD	The transmission delay per unit length (sec/m) and is used as: • TDeff = TDxL; • TDeff = NL/F; or • TDeff = TD (computed from T model) x L.
L (or **LEN**, **LENGTH**)	Physical length of the transmission line (meters). Default is L=1m.
F	Frequency of the transmission line (Hz). Default is F=1.e9 Hz.
NL	Normalized electrical length of the transmission line with respect to the wavelength in the line at the frequency specified by F. Default is NL=0.25 (quarter wave frequency).
IC	Initial condition specification consisting of voltage and current at each transmission line port. The initial conditions apply only when the option UIC is specified in the .TRAN statement.

M	Multiplier used to parallel devices. Default is M=1.
mname	T model reference name.

Note: This element models only one propagating node. If all four nodes are distinct in the actual circuit, then two nodes may be excited. To simulate such a situation, two transmission line elements are required.

When the element name is given in the form TXLxxx, then an enhanced model of the lossy transmission line will be used.

Examples

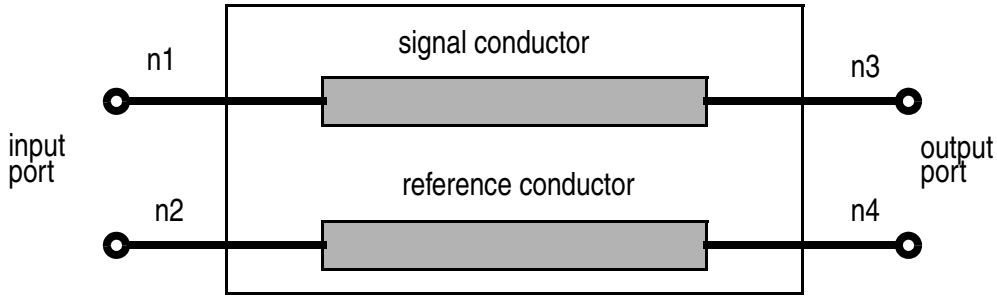
```
T1  1  0  2  0   Z0=50   TD=50nsec   L=0.2
T2  1  0  2  0   Z0=50   TD=10n
T3  1  0  2  0   Zo=50   NL=.25   F=2.5Meg
T4  1  0  2  0   wire10   LENGTH=.2
```

A more detailed description is provided in the SPICE Models Manual.

For connection node syntax to output voltage/current vectors see Chapter 6 Outputs, Table 6-1 Multi-terminal Device Description.

TXL (Transmission Line, Lossy)

TXL is used to specify that an enhanced lossy transmission line element should be used.



Syntax

```
TXLxxx n1 n2 n3 n4 mname
+ L | LEN | LENGTH=val> <M=val>
```

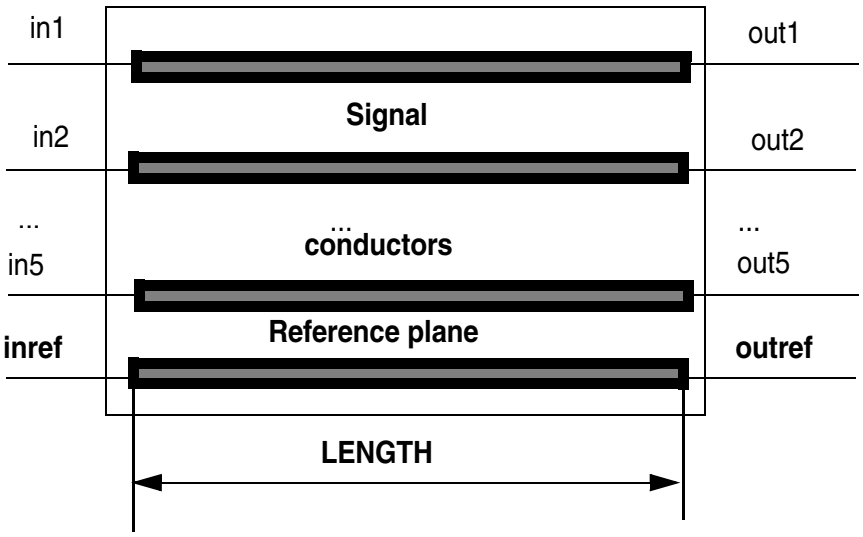
TXLxxx	The lossy uniform transmission line element name begins with TXL.
n1, n2	The nodes at port 1.
n3, n4	The nodes at port 2.
mname	Model reference name.
L (or **LEN**, **LENGTH**)	Physical length of the transmission line (meters). Default is L=1m.
M	Multiplier used to simulate parallel devices. Default is M=1.

Example

```
TXL1  1  0  2  0  tlline  l=.6095
TXLNET  1  2  3  2  connect  length=.16
```

For more details, see the SPICE Models Manual.

U (Coupled Lossy Transmission Line)



Syntax

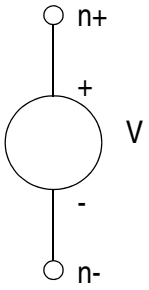
```
Uxxx in1 <in2 … in5> inref out1 <out2 … out5> outref mname
+ <L | LEN | LENGTH=val> <LUMPS=val>
```

Uxxx	The lossy uniform transmission line element name begins with U.
in1, in2, ..., in5	The input nodes.
inref	The reference node at input.
out1, out2, ..., out5	The output nodes.
outref	The reference node at output.
mname	U model reference name.
L (or **LEN**, **LENGTH**)	Physical length of the transmission line (meters). Default is L=1m.
LUMPS	The number of lumps in the equivalent network. Default is calculated.

Examples

```
U5 1 2 3 4 5 0  11 12 13 14 15 0 Microstrip L=.2  LUMPS=5
U4 in1 in2 in3 in4 0  out1 out2 out3 out4 0 USTRIP
U3 a1 a2 a3 a0  b1 b2 b3 b0  over  length=5m
U2 in1 in2 ref1  out1 out2 ref2  SEA LEN=25m lumps=10
U1 1 gnd  2 gnd  rg58  L=0.5
```

V (Independent Voltage Source)



Syntax

```
Vxxx n+ n- <<DC> dcval> <R=tval> <REP=repval> <transrc> <PWLFAST=1>
+ <AC> <acmag <acphase>>>
+ <DISTOF1 <f1mag <f1phase>>> <DISTOF2 <f2mag <f2phase>>>
+ <NORC> <NORS> <NOCG> <RS=val> <CG=val> <LS=val>

Vxxx n+ n- <DC1=dc1val DC2=dc2val>
+ <R=tval> <REP=repval> <transrc> <PWLFAST=1>
+ <AC <acmag <acphase>>>
+ <DISTOF1 <f1mag <f1phase>>> <DISTOF2 <f2mag <f2phase>>>
+ <NORC> <NORS> <NOCG> <RS=val> <CG=val> <LS=val>

Vxxx n+ n- <<DC> dcval> <pwldc> <R=tval> <REP=repval> <PWLFAST=1>
+ <AC <acmag <acphase>>>
+ <DISTOF1 <f1mag <f1phase>>> <DISTOF2 <f2mag <f2phase>>>
+ <NORC> <NORS> <NOCG> <RS=val> <CG=val> <LS=val>

Vxxx n+ n- SD=val TS=val

Vxxx n+ n- sin(vo va <freq <td < theta <phase>>>>)
+ sin2(vo va <freq <td < theta <phase>>>>)

Vxxx n+ n- PWLZ (t1 v1 < t2 v2 <... > >)

Vxxx n+ n- PAT <(> vhi vlo td tr tf tsample data <RB=val>
+ <R=repeat> <)>

Vxxx n+ n- PAT <(> vhi vlo td tr tf tsample [component 1 ...
+ component n] <RB=val> <R=repeat> <)>

Vxxx n+ n- LFSR <(>vlow vhigh tdelay trise tfall rate seed <taps>
+ <rout=val><)>
```

Vxxx	Independent voltage source- names begin with V.
n+, n-	Positive and negative node names.
DC	Keyword for the DC source value.

dcval	DC value of the source. SmartSpice uses this value during any DC analysis phase. SmartSpice also uses dcval during the Transient analysis phase if transrc is not specified. dcval has a default value of 0.
DC1=dc1val, DC2=dc2val	Specifies the initial and pulse values of the source. SmartSpice changes the DC value of the source from dc1val to dc2val during DC transfer curve calculation. If transrc is not specified, then SmartSpice uses dc1val as a DC value during the Transient analysis.
pwldc	Specifies the DC piecewise linear function to be used. Instead of dcval, dc1val and dc2val, SmartSpice uses these values during DC transfer curve calculation.
R=tval	Specifies the timepoint from which the source waveform should be repeated. The section of the waveform between tval and the end of its specification will be repeated until the transient analysis is completed. This parameter is used only in conjunction with the parameters transrc or pwldc.
REP=repval	Specifies the length of the source waveform, counting backwards from the end, to be repeated. For example, if a transient waveform specification has its endpoint at t=3n, and REP=0.4n, then SmartSpice will repeat the section of the waveform from 2.6n to 3n, until the simulation is completed. This parameter is used only in conjunction with transrc or pwldc parameters.
transrc	Transient analysis source specification. It can be one of the following types:

- AM - amplitude modulation;
  `AM <(> <sa <offset <fm <fc <td>>>>> <)>`
- EXP - exponential;
  `EXP (v1 v2 <td0 <tau0 <td1 <tau1>>>>)`
- PL - piecewise linear pairs of (voltage, time);
  `PL(v1 t1 <v2 t2 <v3 t3<...>>> <R=tval> <TD=val>)`
- PULSE - pulse;
  `PULSE (v1 v2 <td <tr <tf <pw <per>>>>)`
- PWL - piecewise linear pairs of (time, voltage);
  `PWL (t1 v1 <t2 v2 t3 v3 ... tk vk> <R=tval> <TD=val>)`
- PWL(timename, voltagename) - data driven piecewise linear;
  `PWL(TIME, VOLTAGE)`
- PWLFILE - piecewise linear from the specified file;
  `PWLFILE filename <TD=val>`
- PWLFILEDESC - read from specified file as series of (time, voltage) pairs;
  `PWLFILEDESC(1 start dt) PWLFILE filename`
- PWLFILEDESC - read from specified file as series of uniformly separated values;
  `PWLFILEDESC(2 start dt vlow vhigh fall_time rise_time) PWLFILE`

filename

- PWLFILEDESC - read specified vector from specified raw-file as series (time, voltage) pairs;

  PWLFILEDESC (3) PWLFROMRAWFILE filename PWLFROMRAWVECTOR vectorname

- SIN - sinusoidal;

  SIN (vo va <freq <td <theta <phase>>>>)

SmartSpice uses the transient source values during Transient analysis. Initial transient source value at time 0 will be used for any DC analysis phase when neither DC nor dcval is specified.

The number of points in a PWL waveform are unlimited.

filename	Full name of the file contains pairs of values making up the piecewise linear waveform. filename can be the parameter that is set in the subcircuit's arguments or in the X-call of the subcircuit.

```
xv1 NODE_1 0  5 6  VOL   PARAMS: PATHTOVALUES1 ='stimulus1.dat'
.subckt VOL 1  2  3  4  PARAMS:   PATHTOVALUES2 =' stimulus2.dat'
V1   1   2   pwlfile  PATHTOVALUES1
V2   3   4   pwlfile  PATHTOVALUES2
.ends
```

In the case of the same names of the parameter which points out on the file name path in the X-call and subcircuit definition, name is defined in the X-call will have higher priority than in subcircuit definition.

PWLFAST=1	Set to 1 to suppress unnecessary reloading of PWL data between runs. Default is 0.
TD=val	Time delay. Allows you to shift PWL source waveform on time equal val. Default is 0.
AC	Keyword for the AC source value.
acmag	AC magnitude. Default is 1.
acphase	AC phase. Default is 0 degrees.
DISTOF1, DISTOF2	Keywords specifying that the independent source has distortion inputs for two values.
f1mag, f2mag	Magnitudes. Default values are 0.
f1phase, f2phase	Phases. Default values are 0.
NORC	This flag suppresses the connection for both serial resistor RS and grounded capacitor CG for Transient Noise Analysis.
NORS	This flag suppresses the connection for serial resistor RS for Transient Noise Analysis.
NOCG	This flag suppresses the connection for grounded capacitor CG for Transient Noise Analysis.

RS=val	Specify the value of serial resistor in Ohms for Transient Noise Analysis. Default is 0.1 Ohm. For Small-Signal Multiterminal Network Analysis, RS allows you to specify the serial resistor value through DC bias supply voltage source connected to the network terminal. Default RS value is calculated so that it does not disturb circuit behavior.
CG=val	Specifies the value of grounded capacitor in Farad for Transient Noise Analysis. Default is 1.e-15 F.
LS=val	Specifies the value of the inductor in series with the DC bias supply voltage source that is connected to the network terminal for Small-Signal Multiterminal Network Analysis. Default LS value is calculated so that it does not disturb the circuit behavior in the frequency range of interest.

Examples

```
VCC 1 0 DC 5
V1 1 0 DC -1 AC 2 45
V2 1 0 DC1 -1 DC2=-2 PULSE(-1 -2 2NS 1NS 1NS 5NS 15NS)
VIN 1 0 PWLFILE ../pwldata/variable.dat
VINP 3 2 PWLFILEDESC(1 0.1ms 5ns)
+ PWLFILE ../pwldata/analog.d
```

The first example describes a voltage source connected between nodes 1 and 0 with a DC value of 5V. The second example describes a voltage source with a DC value of -1V, and an AC magnitude of 2V and phase of 45 degrees.

The third example describes a voltage source with an initial DC value of -1V, a pulse DC2 value of -2V, and a pulse waveform.

In the fourth example, the file variable.dat in the subdirectory pwldata is opened, and the pairs of values in that file are read and used in exactly the same manner as if the identical values had been entered after a PWL directive.

In the fifth example, the file analog.d in the subdirectory ../pwldata is opened and read. The values are interpreted as a series of voltages with the first value corresponds to time=0.1ms, and a uniform 5ns separation between all voltages.

```
.param tri=150ps tfi=150ps
* PWLFILEDESC format is:  start dt vlo vhi fall rise
vin  inp  0  pwlfiledesc  (2  0  3ns  0v  3.3v  'tri'  'tfi')  pwlfile
test3.dat
```

For further details on transrc specification, see the SPICE Models Manual.

Example for Damped Sine

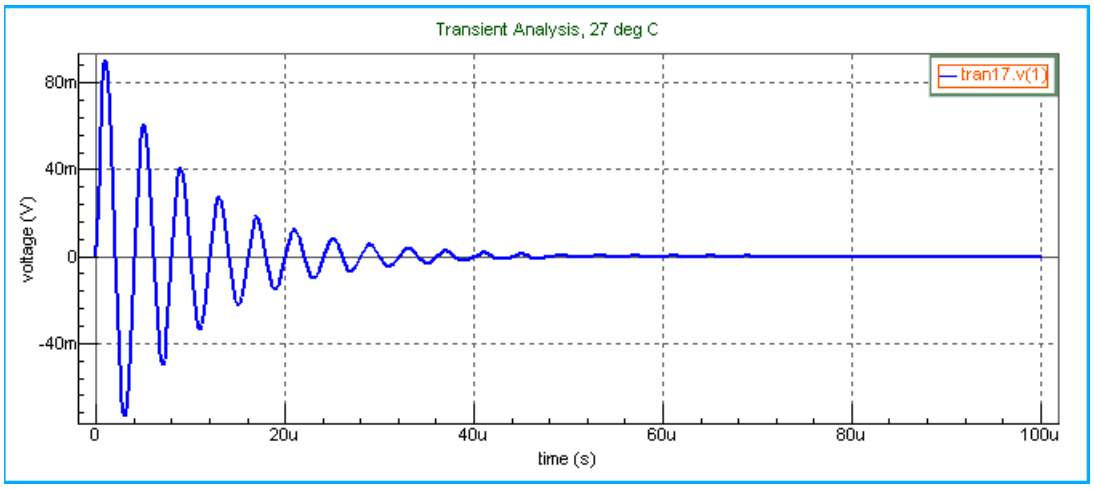
```
* Damped sinus example for VSRC
.param THETA = 1E5 ; <-- Sets damping parameter for sine waveform
Vin  1 0  DC 0  SIN(0 0.1 0.25MEG 0  THETA)
VCC  2 0  DC 1
RS1    1  2  1K
.options tmax=1u nomod
.TRAN 1u 100u
.print  v(1)
```

```
.END
```

The equation for damped sine is:

$$Y = Vo + Va * SIN((FREQ*Time + Phase/360) + 2 * Pi) * EXP(-(Time*Theta))$$

where $EXP(-(Time*Theta))$ is damping component.

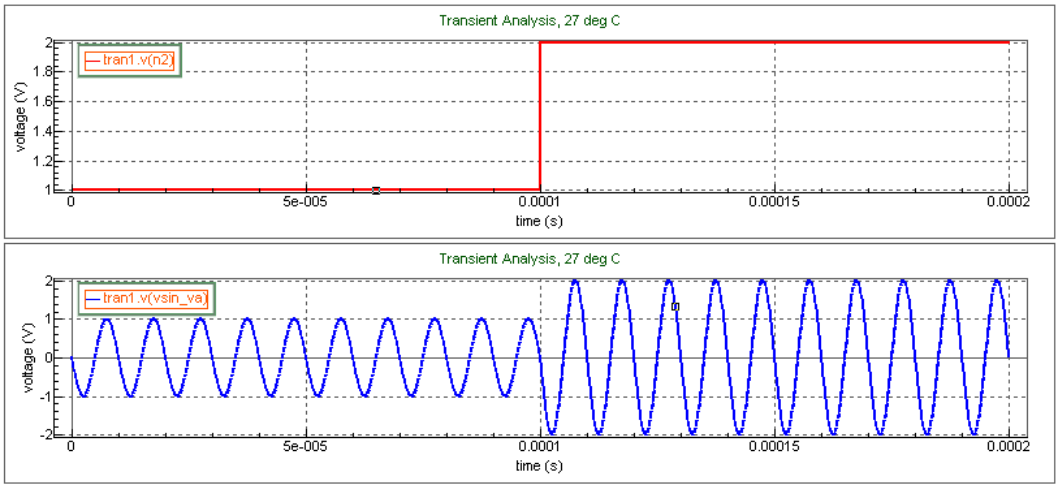


Parameters in sine function such as V0, Va, Freq, Td, Theta, and Phase can be as runtime expressions.

Example for Sine with Runtime Expressions

```
**** Runtime expressions in VSRC sine function
V2 n2 0 0 pwl (0,1  100u,1  100.01u,2 )
R0 n2 0 1
****** Va
Vsin_va vsin_va 0       DC 0 SIN (0 'v(n2)' 0.1MEG 0 0 180)
R1  vsin_va 0 1
.TRAN 10n 200u
.PROBE v(vsin_va) v(n2)
.END
```

In the example above the amplitude is defined as the expression `'v(n2)'`. `V(n2)` changes its value from `1v` to `2v` at 100E-6 sec, which in turn increases the sine amplitude on voltage source `Vsine_va` up to `2v`.



Noisy Voltage Source

SD=val, TS=val	To create a noisy voltage source element. A DC voltage source (DC value of 10V) with 3mV of standard deviation (sd) that changes to its new value every 0.025ns. A gaussian distribution (with standard deviation of 3mV) is added to the dc value of 10V.

Example

```
VCC 8 0 DC 10 sd=300u ts=0.025ns
```

Amplitude Modulation Subfunction (DBSAM)

Example

```
v0 net5 0 sin(900m 50m 1e06 0 0) sin2(900m 50m 1.025e06 0 0)
```

PWLZ Voltage Source

A PWLZ source is the same as a PWL source except it allows hi-impedance state in transient analysis. To realize hi-impedance state, SmartSpice introduces the controlled resistors internally connected to positive and negative terminals. These controlled resistors will have different values depending on operating states. In normal state internal resistors will have a small value defined by `R=1/ZRES`, so that the impact to circuit behavior should be minimum. In hi-impedance state (called z state), internal resistors will have high resistance value defined by `R=ZRES`, and `ZRES` is defined by `.OPTIONS ZRES`.

A `PWLZ` source is only supported for voltage source element, and can not be used for current source. If a `PWLZ` keyword is specified in the current source element, SmartSpice issues an error message and stops the simulation.

Example

```
v1 n1 0 pwlz (0 0 5n 1.65 10n 3.3 50n z 60n 1.65)
```

In this example, the voltage source `V1` is 0V at time 0, and changes from 0V to 1.65V in 5 ns. From 5 ns and 10 ns, the voltage source value changes from 1.65V to 3.3V. The voltage source value stays at 3.3V between 10 ns and 50 ns. Starting from 50 ns to 60 ns, V1 becomes in hi-impedance state by changing the value of internal resistors from `1/ZRES` to `ZRES` and

setting voltage source value to 0V. After 60 ns, it returns to normal mode and stays 1.65 V to the end of time.

Optional Parameters of PWLZ Voltage Source

The four instance parameters: ZTRISE, ZTFALL, RNORM and RHIZ are implemented to specify the characteristics of PWLZ voltage source.

Syntax

```
PWLZ( t1 v1 < t2 v2 < ... > > ) ZTRISE=val ZTFALL=val RNORM=val
RHIZ=val
```

PWLZ	Specifies the PWLZ source
t1, t2,...	Time points (in seconds).
v1, v2,...	Source values (in volts). The keyword z is allowed to specify hi-impedance state.
ZTRISE	transition time from Hi-Z to normal state. Default is 10 pico seconds.
ZTFALL	transition time from normal to Hi-Z state. Default is 10 pico seconds.
RNORM	value of internal resistors in normal state. Default is 1e-12(=.option GMIN).
RHIZ	value of internal resistors in Hi-Z state. Default is 1e+12(=.OPTION ZRES).

ZTRISE and ZTFALL must have positive value, otherwise the default value will be used.

If (ZTRISE+ZTFALL) >= (T[n+1] - T[n]), SmartSpice will calculates new values by the following formula;

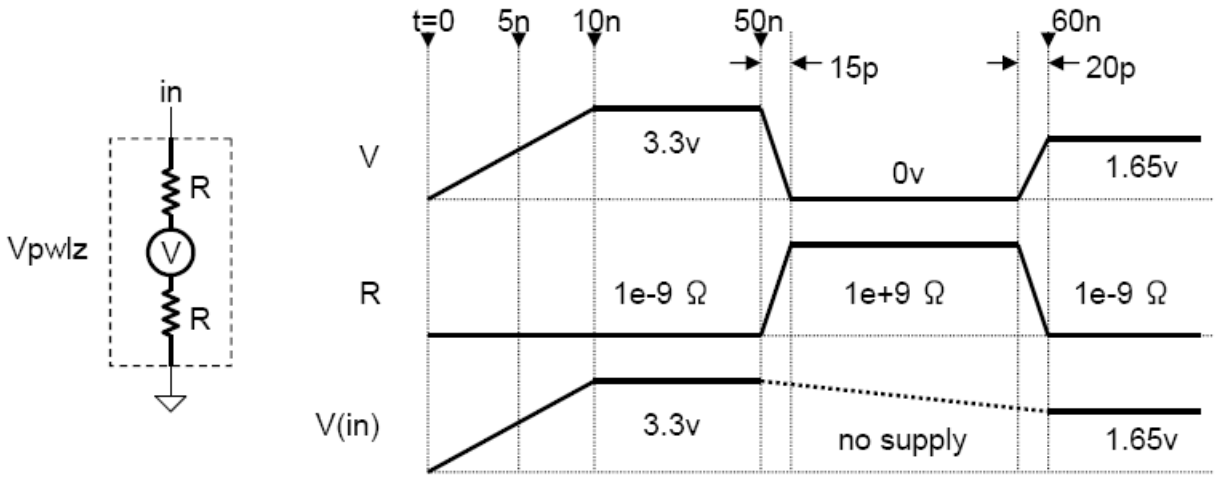
```
ratio = (T[n+1] - T[n]) / (ZTRISE+ZTFALL)
ZTRISEnew = ZTRISE*ratio
ZTFALLnew = ZTFALL*ratio
```

where T[n] is the time point where HiZ mode will start, and T[n+1] is the end time of HiZ mode.

RNORM and RHIZ are allowed to have a value in the range GMIN <= R <=ZRES, where GMIN and ZRES are .OPTION parameters. This limitation eliminates numerical problems in matrix solving and ensures convergency. If invalid values are specified to RNORM and RHIZ SmartSpice will internally set the same values to them as defined by GMIN and ZRES respectively.

Example

```
Vpwlz in 0 pwlz (0 0 5n 1.65 10n 3.3 50n z 60n 1.65) ZTRISE=20ps
ZTFALL=15ps
+ RNORM=1e-9 RHIZ=1e+9
```



Pattern Function for Voltage Sources

SmartSpice provides new pattern function for independent voltage sources.

Vxxx	Independent voltage source name.
n+, n-	Positive and negative node names.
PAT	Keyword for a pattern function.
vhi	High voltage or current value for pattern source.
vlo	Low voltage or current value for pattern source.
td	Time delay. Allows you to shift pattern source waveform on time equal val.
tr	Duration (in seconds) from the low value to the high value.
tf	Duration (in seconds) from the high value to the low value.
tsample	Time spent at the pattern value (in seconds).
data	String of 1, 0, M or Z representing a pattern source. The first letter must be B. 1 is the high voltage or current value, 0 is the low voltage or current value, M is the value which is equal to 0.5*(vhi+vlo), and Z is the high impedance state (only for voltage source).
component 1 ... component n	Component is the element that makes up nested pattern, which can be a b-string or a pattern name defined in `.PAT` statements. Brackets ([ ]) must be used. Component can also be followed by `R=repeat` and `RB=val`.
RB=val	The repeating bit. The repeat operation starts from this bit. The value `val` must be an integer and greater or equal that 1. Default is 1.

R=repeat	The repeating keyword. Shows how many times the repeating operation will be called. The value `repeat` must be an integer. `R=-1` means that the pattern will be repeated until the analysis is completed. The value `repeat` must be greater or equal that -1. Default is 0.

Example

```
v1 1 0 pat (10 0 0n 1n 2n 5n b10mz r=-1)
```

This example shows the pattern source with one b-string 10mz, which will be repeated until the analysis is completed. In this pattern:

- High voltage is `10v`
- Low voltage is `0`
- Time delay is `0n`
- Rise time is `1n`
- Fall time is `2n`
- Sample time is `5n`

Example

```
.param vhi=10 vlo=0 td=10n tr=1n tf=2n tsample=5n
v1 1 0 pat (vhi vlo td tr tf tsample b101 r=3 rb=2 b1mzm)
```

This example shows the pattern source with two b-strings: `101` and `1mzm`. The first b-string repeats three times from the second bit. Therefore, the final pattern will be:

```
101 01 01 01 1mzm
```

Nested Pattern Source Function for Voltage Sources

SmartSpice provides nested pattern source function for independent voltage sources. Nested pattern is a combination of a b-string and other nested structures defined in `.PAT` statements.

Example

```
v1 1 0 pat ( 10 0 0 2n 3n 5n [b101m r=1 rb=3 b1001] r=1 rb=2)
```

When nested pattern will be expanded, the pattern function will be `101m 1m 1001 1001`.

For more details, see Section .PAT (Pattern Definition for Voltage/Current Source).

Pseudo Random-bit LFSR Function

LFSR	Specifies the voltage/current source as PRBS.
vlow	The low voltage/current level.
vhigh	The high voltage/current level.
tdelay	The time delay.
trise	The duration (in seconds) from the low value to the high value.
tfall	The duration (in seconds) from the high value to the low value.
rate	The bit rate.
seed	The initial value (in integer form) of the shift register.
taps	The bits used to generate feedback. Must range from 2 to 64.
rout	The output resistance.

Example

```
V1 1 0 LFSR (0 1 1u 1n 1n 10meg 3 [2, 5] rout=5)
```

HSPICE Compatible Behavior in Pulse Source (V-element) with Period Parameter PER

SmartSpice supports the following syntax for pulse source specification of a V-device:

Syntax

```
PULSE (v1 v2 <td <tr <tf <pw <per>>>>>)
```

If the parameter `per` is not specified, both HSPICE and SmartSpice generate the same single pulse waveform.

The default behavior of SmartSpice is the same as HSPICE.

When the parameter `per` is set to zero, SmartSpice generates a single pulse waveform.

When the parameter `per` is set to a nonzero value, and the value is less than the full pulse width (*tr* + *pw* + *tf*), SmartSpice sets the parameter `per` = (*tr* + *pw* + *tf*), and outputs the warning: “Device name: pulse period too small, reset to `per`”.

Example

```
vpulse 1  0  pulse( 0  5  500n  1u  1u  500n  10n)
vpulse n1 n0 pulse(1 -1 0 1n 1n 7n '75n+(v(1)*25n)')
```

In the first example, `0` and `5` are the respective initial and high voltage of the pulse source. The pulse waveform has a delay of `500ns`, a rise and fall time of `1u`, a pulse width of `500ns`, and a period of `10ns`.

In the second example a non symmetric pulse is produced going from `+1v` to `-1v` in `1ns` and staying low for `1ns` before returning high in `1ns` and staying high for `75ns`. The waveform is then repeated `7ns` low and `75ns` high until the end of the simulation time.

As parameter `per` = *10ns < (tr + pw + tf)=1u+1u+0.5u=2.5u* SmartSpice sets the parameter `per` internally to `2.5u`, and outputs “Warning: vpulse: pulse period too small, reset to 2.5e-06” and forms the following waveform:

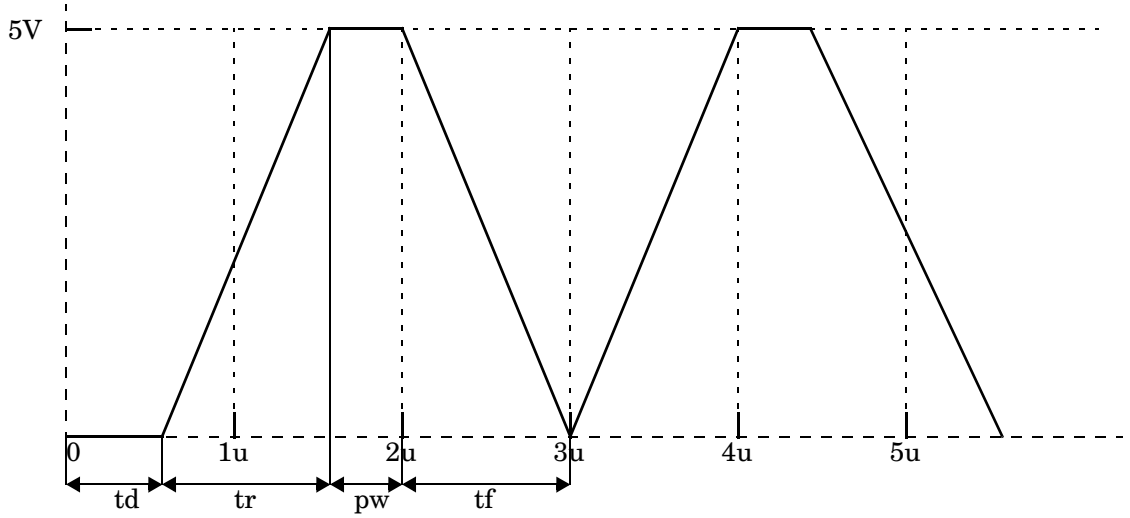
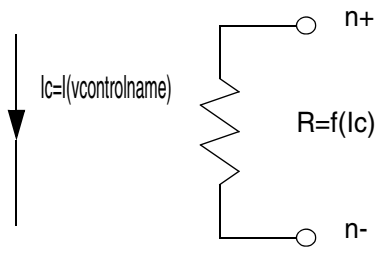


Figure 3-4 Pulse Waveform

W (Current-Controlled Switch)[1]



Syntax

```
Wxxx n+ n- vcontrolname mname <ON|OFF>
```

Wxxx	Current-controlled switch element name. Must begin with W.
n+, n-	Positive and negative terminal names.
vcontrolname	Name of the voltage source of the controlling current flow. The positive controlling current flow direction is from the positive node, through the source, to the negative node of vcontrol-name.
mname	Model name must reference a current-controlled switch model.
ON|OFF	The initial condition of the switch. ON means the switch is closed. OFF (default) means the switch is open. This parameter is in effect only if the switch condition cannot be determined based on the current through vcontrolname.

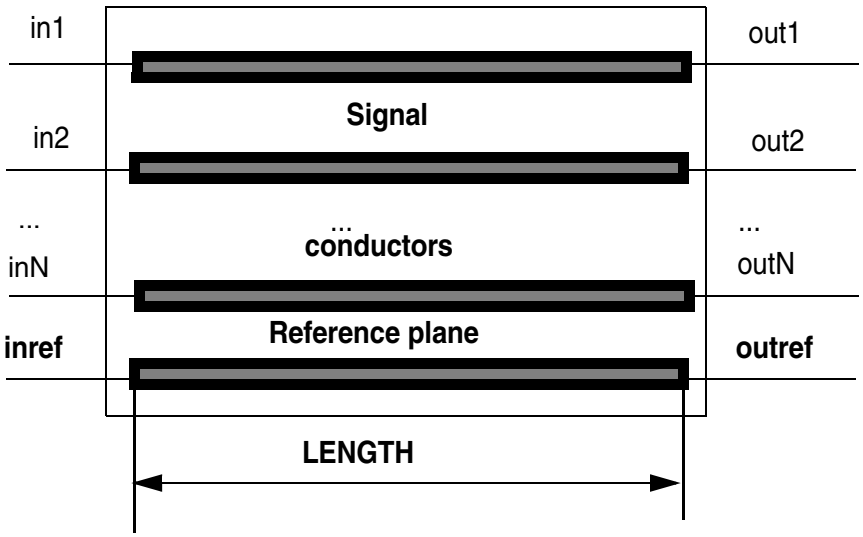
Example

```
W1 1 2 VIN WCMOD OFF
```

For a detailed description of the switch model, see the SPICE Models Manual.

1. The letter W stands for both the current controlled switch, and the coupled lossy transmission line. SmartSpice distinguishes between the two based on the arguments that follow the device name, so there is no ambiguity.

W (Coupled Lossy Transmission Line)



Syntax

```
Wxxx in1 <in2 & inN> inref out1 <out2 & outN> outref
+ <N=val>  <L | LEN | LENGTH=val>
+ <Umodel=>mname | RLGCfile=fname.rlc | RLGCmodel=mname |
+ TABLEmodel=mname | SPmodel=mname
+ <CEXPR= expression> <LEXPR= expression>
+ <FMAX=val> <SEGS=val>
+ <IMRS=0 | INCLUDERSIMAG=NO> <FGR=val>
+ <PRNTAB=val> <RLGCCHECK=val>
+ <OUTRLGCFILE=filename> <OUTMATRIX=SYMMETRIC|NONSYMMETRIC|
+ SINGLELINE>
+ <OUTNUMDGT=val>
```

Wxxx	The lossy uniform transmission line element name begins with W.
N=val	Number of signal conductors (does not include the reference conductor), and optional parameters introduced for compatibility with other SPICE W element syntax.
in1, in2,..., inN	The input nodes.
inref	The reference node at input.
out1, out2,..., outN	The output nodes.
outref	The reference node at output.
mname	W model reference name.
Umodel=mname	Keyword for Umodel type W model reference name.

RLGCfile=fname	Name of the file with RLGC parameters.
RLGCmodel=mname	Keyword for MODELTYPE=RLGC W model reference name.
TABLEmodel=mname	Keyword for MODELTYPE=TABLE W model reference name.
SPmodel=mname	Keyword for MODELTYPE=SP W model reference name.
CEXPR='expression'	Expression is a function of R, RS, L, C, G, GD and frequency (HERTZ) to enforce causality in modeling the frequency dependence of the Cij element of matrix C; as a function of the Rij, Rsij, Lij, Cij, Gij, Gdij elements of the R, Rs, L, C, G, Gd matrices.
LEXPR='expression'	Expression is the function of R, RS, L, C, G, GD and frequency (HERTZ) to enforce causality in modeling the frequency dependence of Lij element of matrix L as a function of the Rij, Rsij, Lij, Cij, Gij, Gdij elements of R, Rs, L, C, G, Gd matrices.

For an example of additional imaginary term in approximate expression of resistance matrix R with skin effect:

$$R(f) = R + \sqrt{f} \times R_S + j \times \sqrt{f} \times R_S$$

can be introduced into the model using the parameter:

LEXPR='L+RS/(2.*3.14159*sqrt(HERTZ))'

L (or **LEN**, **LENGTH**)**=val**	Physical length of the transmission line (meters). Default is L=1m.
FMAX=val	End of frequency-domain transfer function approximation interval (or maximum frequency of interest). The choice of this parameter is critical to receive more correct simulation results of the W element. FMAX represents the highest frequency considered in simulating the transmission lines behavior. It should be chosen large enough to capture all desired frequency affects, but not too large, since the algorithm becomes ill-conditioned at very high frequency. Default value of FMAX is:

1. $FMAX = min\left(\frac{2}{15 \times trise}, 20GHz\right)$, where trise is the parameter RISETIME, which is specified by the .OPTIONS statement, and can be calculated as $trise = 0.1333333 \times 1/f_c$, where *fc* is the maximum frequency of interest.
2. If RISETIME is not given:

$$FMAX = min\left(max\left(\frac{2}{15 \times trise}, 2 \times SinFreq_{max}\right), 20GHz\right)$$

where *trise* is obtained by examining the independent voltage and current PULSE source statements for minimum *tr* (rise time) and *tf* (fall time). $SinFreq_{max}$ is the maximum frequency of SIN current and voltage sources.

3. If `RISETIME` is not given, and the circuit doesn't contain PULSE and SIN sources, *trise* is obtained as MAX (TSTEP, TMAX), where TSTEP is the time interval used to control the printing and plotting of Transient analysis results (it is set with the `.TRAN` statement), TMAX is the maximum value for the timestep (it is set with the `.OPTIONS` statements), and default value FMAX is:

$$FMAX = min\left(\frac{2}{15 \times trise}, 10GHz\right)$$

4. If the RLGC matrices contain a nonzero skin-effect resistance matrix Rs or dielectric-loss conductance matrix Gd, or nonzero and frequency-dependent R or G matrices, then *FMAX=max(FMAX, 1GHz)*.

SEGS=val	Number of segments. Segmenting a lossy transmission line into `val` shorter lines allow to decrease the attenuation and improve simulation accuracy. Default is `1`. For further details, see the SPICE Models Manual.

Note: Parameter SEGS value cannot be changed by `SWEEP` or `.MODIF` statement. To change this parameter value use `.ALTER` statement.

IMRS=0	Flag to suppress the addition of skin effect resistance term to the imaginary part of the impedance: $R(f) \approx R_s \times (1 + j) \times \sqrt{f}$ Suppresses the extraction of the skin effect attenuation from propagation constant. Default is `IMRS=1`. If the `W element` statement contains `LEXPR='expression'`, the `IMRS=0`. For HSPICE compatibility, `INCLUDERSIMAG=NO` is an alias for `IMRS=0`.
FGD=val	Cut-off frequency of dielectric loss f_{gd}: $G(f) = G_0 + \frac{f}{\sqrt{1 + (f/f_{gd})^2}} \times G_d$ Default value is `FGD=0`, and the previous linear dielectric loss dependency $G(f) = G_0 + f \times G_d$ is used.
PRNTAB=val	Flag to print RLGC table, if `prntab=1`, and to additional plot the frequency-dependent R, L, G, C-matrix entries in SPmodel cases if `PRNTAB=2`. Default is `PRNTAB=0`.

RLGCCHECK=val	Flag to turn off checking of RLGC matrices on non-physical negative values and predominance of diagonal elements. Default is RLGCCHECK=0, if the parameter RLGCNOCHECK is set with the .OPTIONS statement. If the parameter RLGCNOCHECK is not specified, default is RLGCCHECK=1, and checking of the RLGC matrices is performed. If the parameter RLGCNOCHECK is given, specification of RLGCCHECK=1 allows you to turn on RLGC matrix checking for separate W elements.

For the SPmodel case default, the checking of the extracted RLGC matrices is not performed.

In the SPmodel case, SmartSpice extracts frequency-dependent RLGC matrices from the S- or Y-parameters. These extracted matrices can be saved as an external RLGCfile using the following control W device statement parameters:

OUTRLGCFILE=filename	This option allows you to save RLGC matrices in an external file with name filename. If this parameter is not specified it will not be saved.
OUTMATRIX	Matrix (data point) format:
SYMMETRIC	Symmetric matrix. Only the lower-half triangle portion of matrices is saved.
NONSYMMETRIC	Nonsymmetric matrix. The full matrices are saved.
SINGLELINE	If the matrices size equal 1 (N=1 - transmission line consists of single signal conductor), all four (L, C, R, G) matrices at any frequency can be written as four numbers on one line, after the frequency value. This saving format is called SINGLELINE.

Example

```
* Frequency-dependent RLGC matrices per unit length
* Matrix format: SINGLELINE
* N - signal conductor number
* Ns - frequency point number
* Nm - matrix number per frequency point
* ---------------------------------------
*   N     Ns    Nm
    1    102     4

*|   Frequency   |--- L[H/m] ---|--- C[F/m] ---|-- R[ohm/m] --|--
G[mho/m] --|
     0.00000e+00     2.578000e-09     8.000000e-11     3.690979e-01
1.639124e-15
     1.00000e+00     2.578000e-09     8.000000e-11     3.692955e-01
2.401876e-12
     1.00000e+08     2.578001e-09     8.000000e-11     2.323900e+00
2.400237e-04
     2.00000e+08     2.578000e-09     8.000001e-11     3.133604e+00
4.800237e-04
     3.00000e+08     2.578000e-09     8.000000e-11     3.754913e+00
```

```
7.200237e-04
     4.00000e+08     2.578001e-09     8.000000e-11     4.278700e+00
9.600237e-04
...
```

OUTNUMDGT=val	Allows specification of the number of digits (after point) in the output R, L, G, C matrix values. Default is 7.

Note: Nodes and parameters in the W-element card cannot be mixed freely. Only one RLGCfile or mname can be used in a single W element model card.

Examples

```
W5 1 2 3 4 5 0 11 12 13 14 15 0 Microstrip L=0.2
W4 in1 in2 in3 in4 0 out1 out2 out3 out4 0 USTRIP
W3 N=3 a1 a2 a3 a0 b1 b2 b3 b0 Umodel=over length=5m
W2 in1 in2 ref1 out1 out2 ref2 RLGCfile=test.rlc LEN=25m
w1 1 gnd 2 gnd rg58 1=0.5
W1 10 0 20 0 n=1 l=0.01 rlgcmodel=qgb_50
or
W1 10 0 20 0 rlgcmodel=qgb_50 n=1 l=0.01
```

A detailed description is provided in the SPICE Models Manual.

X (Subcircuit Call)

Syntax

```
Xyyy n1 <n2 n3 ...> subcktname  <<PARAMS:> parname=val ...> <M=val>
```

Xyyy	Subcircuit names must begin with X.
n1, n2,...	Node names for external reference.
subcktname	Subcircuit definition.
PARAMS	The optional parameter preceding the list of parameters and their values.
parname	The parameter name whose value is set to val, which will be local to the subcircuit. val is either a numerical value, an expression enclosed in single quotes containing previously defined parameters, or a string, which defines a model's name. If the same parameter is assigned a value on more than one statement in the input deck (.PARAM, .SUBCKT, and X statements), then: • a value assigned in a global .PARAM statement (outside subcircuits) is used if it exists, otherwise; • a value assigned in a corresponding X statement is used, if it exists, otherwise; • a value assigned in a corresponding .SUBCKT statement is used, if it exists, otherwise; • a value assigned in a local .PARAM statement (inside the subcircuit) is used, if it exists.
M	Multiplier used to simulate multiple parallel subcircuits. Default is 1.0.

This statement creates instance Xyyy of subcircuit subcktname. There must be the same number of nodes n1 ... as in the subcircuit definition (see Section .SUBCKT (Subcircuit Definition)). These nodes replace the formal node names in the .SUBCKT statement.

Subcircuits can be nested. The number of nesting levels is not limited, but nesting must not be recursive.

When the circuit is loaded, all subcircuit device names are expanded to the form devtype.subcktname.devname, where devtype is the first letter of devname. All local node names in the subcircuit are expanded to the form subcktname.nodename. Furthermore, all of the subcircuit's local model names are expanded to the form subcktname.modelname.

For nested subcircuits, expanded names have multiple dot-separated qualifiers. For example, if a circuit has a call to subcircuit subcktname1, and the subcktname1 definition contains a call to subcircuit subcircuit2, then expanded names will have the following format:

```
devtype.subcktname1.subcktname2.devname
subcktname1.subcktname2.nodename
subcktname1.subcktname2.modelname
```

To view the circuit listing after subcircuit expansion, use the command listing expand.

Note: The default subcircuit delimiter character is a period (.). To learn how to reset this character, read the description of the variable subckt_delimiter in Chapter 5 Commands.

Examples

```
X1 2 3 12 SUB1 par1= 2k
Xsss2 1 6 8 9 10 sub2 PARAMS: a=3.5 b = ‘2*a’
```

String Parameter Example

```
XP6 GND P6_ZCATH ZENNER  PARAMS: V_ZEN=2 M_DFOR=DFOR M_REV=DREV

.SUBCKT ZENNER ZAN ZCATH
DP3 ZAN ZCATH M_DFOR
DP1 ZCATH 1   M_DREV
.ENDS ZENNER

.MODEL DREV D  ( ...)
.MODEL DFOR D  ( ...)
```

Parameters M_DFOR and M_DREV are used as alieses for the model card DFOR, DREV.

The subcircuit element operating temperature can be set by specifiying the parameter TEMP in the statement. The temperature will be applied for all devices inside subcircuit instance.

Example

```
Xyyy n1 <n2 n3 ...> subcktname temp=50
```

.model Statement Inside the Subcircuit’s Definition

In both modes (the default and HSPICE compatible), SmartSpice applies the following model reference expansion scheme:

- Check local .model statements inside the subcircuit’s definition where the device instance is specified.
- Check local .model statements of all levels of hierarchy of nested X instance calls up to top level.
- Check .model statement of top level.

Example

```
.model dmod D
X1 in out ckt1

.subckt ckt1 in out
X2 in out ckt2
d1 in out dmod
.model dmod D (level=3 is=1.3E-5)
.ends
.subckt ckt2 in out
d2 in out dmod
.ends
```

The diode D1 is specified in .subckt CKT1 and refers to the local model x1.dmod defined in the same subcircuit. The diode D2 is specified in .subckt CKT2, which has no models and refers to the upper hierarchy level model x1.dmod.

Circuit Design Language (CDL) Support

SmartSpice supports the following CDL dialects.

Example

```
XA 1 2 3 4 /subcktname
```

In this example a slash (/) character distinguishes a master cell name (following the slash) from the node names (preceding the slash).

Model Reference

When a model reference can not be find for any instance SmartSpice considers such instance as a subcircuit's call and tries to find `.subckt` reference regardless on instance's name.

Example

```
c1 n3 0 1e-14
m1 n3 n4 n6 n6 pmos l=3.5e-07 w=4e-06
m2 n3 n4 n5 n5 nmos l=3.5e-07 w=4e-06
v1 n4 0 pwl(0.0ns 2.5v 4.ns 2.5v 4.125ns 0.0v 8.0ns 0.0v 8.125ns
2.5v)
v2 n5 0 dc 0.0v
v3 n6 0 dc 2.5v
.measure tran i_dyn_2_0_y AVG i(v3) FROM=4.0ns TO=7.6ns
.measure tran i_dyn_3_0_y AVG i(v3) FROM=8.0ns TO=11.6ns
.tran .01ns 12.0ns
.save v(n3) v(n4)
.subckt nmos 1 2 3 4
mos 1 2 3 4 nmos
.model nmos nmos level=49
.ends
.subckt pmos 1 2 3 4
mos 1 2 3 4 pmos
.model pmos pmos level=49
.ends
.end
```

In this example the instances `m1` and `m2` are considered as subcircuit calls.

This feature is ON by default only in -spectre mode. Use the variable `macro_model_support` to control this feature.

Example

```
.control
set macro_model_support=true | false
.endc
```

YVLG (User-defined Verilog-A Element)

This statement is used to describe a user-defined Verilog-A element.

Syntax

```
YVLGxxx n1 n2 ... mname <user_params=param-value>
```

YVLGxxx	Specifies the element name- must begin with YVLG.
n1 n2 ...	list of node instantiations.
mname	Verilog-A module name or VLG model name.
user_params	Optional list of user-defined Verilog-A parameters.

The following restrictions apply during a YVLGxxx statement:

- a .verilog statement, referring the Verilog-A file where the used Verilog-A module is, must be present in the netlist.
- The number of nodes must match the number of nodes declared in the Verilog-A module.
- For Verilog-A vector ports, a node must be associated individually to each vector element.
- Verilog-A allows parameters in expressions that define the left or right bounds of a vector port. However, in a YVLGxxx statement, the list of node instantiation must have a fixed size. Because of this, it is not possible to assign a value to a parameter that will change the size of a vector port. However, resizing the bounds of a vector port is possible in a VLG model card or in a Verilog-A module instantiation statement.

Example

```
* instantiation of the Verilog-A module 'resistor'. 'vss' and 'gnd'

* are the node names. The Verilog-A parameter 'resistance' is

* assigned the value 1k

YVLGr1 vss gnd resistor resistance=1k

* instantiation of the VLG model 'm_resistor_4k'.

YVLGr2 vss gnd m_resistor_4k
...
.model m_resistor_4k VLG MODULE=resistor resistance=4k
```

The behavior of Verilog-A modules is described in details in the VERILOG-AMS LANGUAGE REFERENCE.

Z (JFET/MESFET)

See Section J (JFET/MESFET).

3.13 Dot Statements

.AC (AC Analysis)

The .AC statement performs an AC linear small-signal analysis of the circuit. SmartSpice first creates a linearized small-signal model at the operating point of the circuit, then computes the frequency response over a user-specified range of frequencies.

Syntax

```
.AC DEC|OCT|LIN nump fstart fstop
+ <CALLV> <SAVEV> <UIC> <SKIPDC=no|yes> <IC> <PRINTOP>
+ <speedplot> <sw2_spcf> <<SWEEP> MONTE=val <PRMC<=val>>>
```

or

```
.AC START=fstart STOP=fstop STEP=fincr
+ <CALLV> <SAVEV> <UIC> <SKIPDC=no|yes> <IC> <PRINTOP>
+ <speedplot> <sw2_spcf> <<SWEEP> MONTE=val <PRMC<=val>>>
```

or

```
.AC FREQLIST|POI numsteps freq1 <freq2 <... <freqN>>>
+ <CALLV> <SAVEV> <UIC> <SKIPDC=no|yes> <IC> <PRINTOP>
+ <speedplot> <sw2_spcf> <<SWEEP> MONTE=val <PRMC<=val>>>
```

DEC	Sweep by decades.
OCT	Sweep by octaves.
LIN	Linear sweep.
nump	Number of points per decade or per octave, or number of linear sweep points.
FREQLIST|POI	Indicates the list of frequencies to be performed by AC analysis. It is possible to pass 0 (zero) as numsteps value, which has a meaning of <read all points>.
fstart	The starting frequency.
fstop	The final frequency.
CALLV	Causes SmartSpice to call a previously saved operating point before starting DC operating point calculation.
SAVEV	Causes SmartSpice to save a discovered DC operating point.
UIC	Causes SmartSpice to bypass the DC operating point calculation, and to create a linearized model at the called operating point. In this case, AC analysis is free from any “No Path to Ground” problems.
SKIPDC	To allow for syntax from a Spectre netlist.
IC	Causes SmartSpice to use .IC statements when computing the operating point.

PRINTOP	Causes SmartSpice to print OP information about computing the operating point or the called operating point (if UIC is specified).
speedplot	Allows you to speed up simulation by reusing previously created SmartSpice structures (plot). Only one plot will be created, the waveforms will not be preserved. All measure results will be printed and preserved in the .meas plot. The raw files will contain only the data from the last sweep step and all measure results, if there is no .probe statements in the netlist. Default is off.
sw2_spcf	The second (nested) sweep specification. It is specified in one of the following ways: • SWEEP swname swstart swstop swincr • SWEEP swname LIST nplist swval1 swval2 ... • SWEEP swname swtype np swstart swstop • SWEEP MODIF = dataname <PRTBL> • <SWEEP> DATA=dataname. where:
swname	The name of the parameter to be swept. It can be: • a parameter label defined in a global .PARAM statement. • a name of an independent voltage or current source. • a full-path name of a device parameter. • a full-path name of a model parameter. • the parameter TEMP (temperature sweep).
swstart	The starting value of the swept parameter.
swstop	The stop value of the swept parameter.
swstep	The increment value of the swept parameter.
swincr	The increment value for a linear sweep.
nplist	Number of parameter values swval1, swval2,... specified for a sweep of the type LIST.
swtype	The keyword that defines the type of sweep: DEC, OCT or LIN.
np	The number of points per decade or per octave, or the number of points for LIN sweep.
MODIF=dataname	This references the parametric .DATA statement.
DATA=dataname	This references the parametric .DATA statement.
PRTBL	This optional parameter flags the final table of all modified parameter labels and calculates .MEASURE results. Default is OFF.

<SWEEP> MONTE=val	Causes SmartSpice to perform Monte-Carlo statistical analysis, where `val` is the number of Monte-Carlo repetitions. Parameters of Gaussian, Uniform or Limit function distributions are set by `.PARAM` statements: • `.PARAM parname=UNIF(nomval, relvar <,mult>)` • `.PARAM parname=AUNIF(nomval, absvar <,mult>)` • `.PARAM parname=GAUSS(nomval, relvar, sigma <,mult>)` • `.PARAM parname=AGAUSS(nomval, absvar, sigma <,mult>)` • `.PARAM parname=LIMIT(nomval, absvar)` where:
numsteps	Number of points of interest (frequency points). When set to 0 all points in the list will be used.
parname	Parameter name whose value is calculated by distribution function.
UNIF	Uniform distribution function with relative variation.
AUNIF	Uniform distribution function with absolute variation.
GAUSS	Gaussian distribution function with relative variation.
AGAUSS	Gaussian distribution function with absolute variation.
LIMIT	Random limit distribution function with absolute variation, `+/- absvar` is added to `nomval` based on whether the random outcome of a -1 to 1 distribution is greater or less than 0.
nomval	Nominal value for Monte-Carlo analysis.
absvar	The absolute variation: the `AUNIF` and `AGAUSS` vary `nomval` by `+/-absvar`.
relvar	The relative variation: the `UNIF` and `GAUSS` vary `nomval` by `+/- nomval x relvar`.
sigma	The parameter for Gaussian distribution. The `absvar` or `relvar` is specified at the `sigma` level. The effective standard deviation of a random sample will be `absvar/sigma`.
mult	The multiplier repeats function calculation `mult` times, saving only the largest deviation. Default is `1`.

The name and type of measurements and names of output variables for Monte-Carlo analysis are set by .MEASURE statements.

PRMC<=val>	Causes SmartSpice to print the value of modified and measured parameters on each Monte-Carlo iteration. The PRMC=2 saves all parameters in the raw file, if the option sweepmonte is present. The feature doesn't work in HSPICE compatible mode.

The SmartSpice input deck can contain more than one .AC statement. At least one independent source of an input deck must have a .AC specification.

Examples

```
.AC DEC 10 10K 10G

.AC OCT 10 10KHZ 10GHZ

.AC LIN 101 0.01MEG 10G

.AC DEC 5 0.1MEG 10G CALLV SAVEV SWEEP VIN 2 5 1

.AC LIN 41 1MEG 10G CALLV UIC

.AC freqlist 3 100 1000 10000 sweep vc 10 20 1
```

In the first example, SmartSpice calculates the DC operating point, linearizes the circuit, and calculates a frequency sweep of 10 points per decade from 10kHz to 10GHz.

In the second example the frequency sweep has 10 points per octave.

In the third example there is a total of 101 point linearly spaced in frequency over the specified AC range.

In the fourth example, SmartSpice calls a previously saved operating point, computes and saves the DC operating point, and calculates the frequency response. The AC analysis is repeated four times. Each time the voltage source VIN is changed. The four values of VIN are 2V, 3V, 4V and 5V.

In the fifth example, SmartSpice linearizes the circuit immediately around the called operating point, and performs a linear frequency sweep.

.ACDCFACTOR (Degradation Simulation Statement)

Syntax

```
.ACDCFACTOR Vds=value Vgs=value <Vbs=value> <time=value>
```

Vds, Vgs, Vbs, time	Required keywords.
value	The parameter value.

If optional parameters Vbs and time are not specified, then Vbs is set to 0.0 and time is set to Tstop (transient final time). The third optional parameter is always Vbs.

Example

```
.ACDCFACTOR 1 2 3
```

In the example, the parameters Vds, Vgs, Vbs will be set to 1V, 2V, and 3V correspondingly.

Note: This Statement is prepared for future use.

.ACHECK (Check Node Activity)

This statement causes SmartSpice to check nodes on every timepoint during the selected window of the transient simulation, and report active and inactive nodes.

Syntax

```
.acheck <<name=> node_pattern> <exclude="exclude_node_pattern">
+ <dv=val> <start=val> <stop=val> <all> <timesort> <file=file_name>
+ <level=val> <subckt=”subckt_pattern”>
+ <report=none|inactive|active|both>
+ <type=all|gate>
```

<name=> node_pattern	Node pattern describing nodes to report. If not specified, "*" is assumed by default and all nodes are reported.
exclude	Node pattern describing nodes to exclude from report, default is empty (no excludes).
dv	Voltage difference threshold; node is considered active if absolute voltage change on that node is greater than `val`. Default is 0.1
start	Defines the start of the time window when checking is performed. Default is 0 (Transient analysis start)
stop	Defines the stop of the time window when checking is performed. Default is -1 (Transient analysis stop)
all	Report all inactive nodes in a `.achiaX` file. Default is off (do not report all inactive nodes).
timesort	Sort active nodes report by first activity time and not by name. Default is off (sort by name).
file	Specifies the output file name. Extension of a file will be `.achaX` or `.achiaX` depending on report type. Default is `netlist_name`.
level	Number specifying the hierarchy level to filter nodes. Default – all levels.
subckt	Subcircuit name pattern to filter nodes. Local nodes of subcircuit and nodes outside the subcircuit connected to the terminals will be selected for check. By default all nodes from all subcircuits will be checked.
report	Forces to report just defined nodes activity types. Available values are `none`, `inactive`, `active` and `both`. Default value is `both`.

type	Specifies type of nodes connection. • `all` - check all nodes. • `gate` - check only nodes connected to transistor's gate. Default is "all".

All parameters are optional. If not given, defaults are assumed.

Flow

1. When the Transient analysis reaches the start time of the activity check window, all checked nodes store the reference voltage value (at the time of the window start).
2. On subsequent timepoints, if the node voltage change becomes greater than the reference by plus or minus the voltage difference threshold, the node is considered active and removed from further checking of inactive nodes.
3. Checking stops when time window stop is reached or `tran` finishes.
4. When `tran` finishes, collected data is dumped into the files:

   `netlist_name.achX - active nodes, netlist_name.achiX - inactive nodes`

If `all` was used, then `netlist_name.achiaX` will contain all inactive nodes in the netlist.

Inactive nodes are always sorted by name and have their reference voltage reported. Active nodes are sorted by name or by default, and have their first activity time reported. If `timesort` is supplied, then they will be sorted by the time of first activity. Multiple `.acheck` statements are allowed. Produced files have a suffix and are not overwritten (e.g., `netlist_name.ach0`, `netlist_name.ach1`, etc.).

Example

```
.acheck n1* exclude="*_in" start=2.1u stop=2.9u all timesort
```

In this example, all nodes beginning with `n1` but not ending with `"_in"` will be checked during transient time window from 2.1u to 2.9u. All inactive nodes will be reported. Active nodes report will be sorted by the time of first activity.

.AGE (Degradation Simulation Statement)

Syntax

```
.AGE time
```

`time` is a number which defaults to the units of seconds. The number can also have a one-letter unit specifier following the number (no spaces between number and letter) to indicate a different unit, such as hours, days, or years:

s	seconds (the default without unit specifier)
m	minutes
h	hours
d	days
y	years

Aged transient analysis will be disabled if there is no instance to degrade within an input deck.

Example

```
.AGE 10y
```

Note: This Statement is prepared for future use.

.AGETHRESH (Degradation Simulation Statement)

Syntax

```
.AGETHRESH number
```

Note: This Statement is prepared for future use.

.ALTER (Alter Input Deck)

Syntax

```
.ALTER
```

This statement is used to load the input deck several times. At each input, a different configuration is identified with a special title. In batch mode, each configuration will be simulated as they are read. In interactive mode, each configuration is stored as a new circuit, and can be executed one circuit at a time using the command `run`, or together using the command `run alters` (See the note below for behavior on multi-threaded platforms).

An input deck that contains one or more `.ALTER` statements is processed several times:

- the first time, the program reads the deck from the beginning up to the first `.ALTER` statement;
- the second time the program reads the input deck from the beginning up to second `.ALTER` statement (or `.END` statement);
- the third time, the program reads the input deck from the beginning up to the third `.ALTER` statement (or `.END` statement), and so on, until the `.END` statement is read.

The new statements read after the `.ALTER` statement is encountered are appended to the input deck, or they modify the existing input deck as follows:

1. When the new statement defines an element, model, or subcircuit with a name identical to one that is already defined, the old one is commented out and the new statement replaces it.
2. If the new statement is a simulation statement (`.TRAN`, `.DC`, `.AC`, `.OP`, `.SNS`, `.TF`, `.NET`, `.NOISE`, `.DISTO`, `.PZ`, `.FOUR`), it is added to the list of simulations to be run.
3. If the new statement is any output statement (i.e., `.PRINT`, `.PLOT`, `.GRAPH`), it will be added to the list of output statements.
4. If the new statement is any control statement except `.TEMP` and `.MODIF` (i.e., `.IC`, `.INCLUDE`, `.LIB`, `.NODESET`, `.ST`), it will be added to the list of control statements. New `.TEMP` and `.MODIF` statements replace old ones. If the new `.IC` or `.NODESET` statement specifies a node voltage that was already assigned a value, the new values will replace the old ones.
5. If a parameter name is defined through a new `.PARAM` statement, and is identical to an existing parameter name, the new assigned value will replace the old one.
6. If an option name is defined through a new `.OPTIONS` statement, and is identical to the existing one, the new assigned value replaces the old one.

altermode

The current implementation of `.ALTER` statement functionality has one disadvantage - SmartSpice reloads the input deck several times. To avoid this overhead, a new algorithm has been developed. The following are steps of new algorithm:

1. SmartSpice reads the input file, up to the first `.ALTER` statement and creates basic circuit for simulation. In batch mode SmartSpice performs simulation or commands from `.control` block. In GUI mode SmartSpice performs commands from `.control` block only.
2. After that SmartSpice reads the input between `.ALTER` statement, and either the next `.ALTER` statement or the `.END` statement. New block of statements will be used to modify basic circuit.
3. SmartSpice modifies the circuit and then re-simulates.

4. SmartSpice repeats steps 2 and 3 until all netlist will be processed.

Note: SmartSpice performs all analysis statements prior to the first .ALTER statement and repeats these for each .ALTER stage.

Only .param, .func and .define statements are supported in the altermode algorithm.

The altermode feature is ignored when parallel .ALTER is used or the variable read_first is specified.

The above functionality may be turned ON using new boolean variable altermode. The default value of altermode is false.

Example

```
set altermode = true
```

Note: A new subcircuit definition completely replaces an older one. If the new statement defines a new element, model or subcircuit, it is added to the input data.

There is no limit on the number of .ALTER statements that can be placed in an input deck. In batch mode, when the Boolean variable readfirst is not set, each successive circuit is simulated immediately after it is parsed. When readfirst is specified, all circuits will be parsed before any are simulated.

In window mode, only the last portion of the deck (identified with the title "WITH .ALTER, WHOLE") will be simulated unless the following block is added at the beginning of the file:

```
.CONTROL
run
.ENDC
```

SmartSpice comments out replaced lines. When the Boolean variable comment is set, all replaced lines appear in the input deck with the prefix (*) "altered:" to identify them as altered lines. When the variable comment is not set, the lines are removed and do not appear in the input deck.

When an input deck contains a .ALTER statement, SmartSpice generates an extended title for each subsequent simulation, for example:

```
ALTER TITLE TEST (WITH .ALTER, 1)
ALTER TITLE TEST (WITH .ALTER, 2)
ALTER TITLE TEST (WITH .ALTER, WHOLE)
```

The added part of the title can be customized by setting the variable alter_add_title. This variable can be set interactively and in the SmartSpice.ini start up file. For example, the command:

```
set alter_add_title = '(WITH .ALTER, NUMBER %D)'
```

generates the following titles for each run:

```
ALTER TITLE TEST (WITH .ALTER, NUMBER 1)
ALTER TITLE TEST (WITH .ALTER, NUMBER 2)
ALTER TITLE TEST (WITH .ALTER, NUMBER -1)
```

Parallel .ALTER and the Command-line Flag -nodist[ribute] (Unix platforms only)

When a deck contains one or more .ALTER statements, its behavior depends on whether it is running in batch or interactive mode. In interactive mode, SmartSpice pops up a window asking whether to run the selected deck, or all decks in the sequence.

However, in batch mode, if the command-line option -P n_cpu is specified on UNIX (where n_cpu is the number of CPUs to be used), SmartSpice will create separate decks, one for each .ALTER, and will simultaneously run the n_cpu of them. One deck will be run in parallel on each processor. When the CPU number is less than the .ALTER number, as each deck finishes, another will be started until all decks have been simulated. The results and output information will be saved in separate .raw and .out files; separate files for each .ALTER with the name deckname-alternumber.

If the additional flag -nodist[ribute] (letters in brackets are optional) is specified, SmartSpice will behave as in previous releases. The .ALTER statements will run one at a time, but in parallel on n_cpu processors, and the results will be saved in single .raw and .out files.

Examples

Example using the .ALTER statement:

```
.CONTROL
run
.ENDC
VDS 1 0
vgs 2 0
vbs 4 0 -1
.dc vds 0 5 0.5 vgs 1 5 1
.iprint i(vds)
m1 1 2 0 4 NMOS1 l=8u w=25u
.MODEL NMOS1 nmos
*
.alter
m1 1 2 0 4 NMOS1 l=16u w=25u; change channel length
.model NMOS1 nmos (vt0=1.2 kp=2e-5 phi=0.6
+ nsub=5e15); change model parameters
.iplot i(vds)
.END
```

In this example, the input deck is loaded twice and two circuits are created. The first time the input deck is loaded, SmartSpice reads the input deck from the beginning up to the first .ALTER statement. The second time, SmartSpice loads the whole input deck with an altered device m1, an altered model NMOS1, and adds an additional .IPLOT statement. In batch mode, both circuits are simulated automatically. In Windows mode, the circuit variations can be simulated individually.

.APPENDMODEL

The .APPENDMODEL statement appends the parameter values from the source model card to the destination model card.

Syntax

```
.APPENDMODEL SOURCE_MODEL DESTINATION_MODEL
```

or

```
.APPENDMODEL SOURCE_MODEL SOURCE_MODEL_TYPE
+ DESTINATION_MODEL DESTINATION_MODEL_TYPE
```

SOURCE_MODEL	Source model name.
SOURCE_MODEL_TYPE	Source model type.
DESTINATION_MODEL	Destination model name.
DESTINATION_MODEL_TYPE	Destination model type.

Example

This example appends the parameters from the flash model card to the mos1 model card.

```
.model mos1 nmos level=49
.model flash flashcell flashlevel=1
.appendmodel flash mos1
```

For more information about the flash memory modeling, refer to Section 4.8 Flash Memory Core Cell Modeling in the SPICE Models Manual.

.BIASCHK (Check Voltage Bias)

If the voltage bias between some terminals of the device is too large, a breakdown can occur. The `.BIASCHK` statement is intended for tracking and monitoring the voltage bias between some terminals of the particular device. Setting up the limit and the noise in the `.BIASCHK` statement targets a condition for monitoring voltage bias across appointed device. `.BIASCHK` reports the following information during the Transient analysis:

- type of the device
- terminals
- time at which overreach had happened
- voltage bias overreaching the limit
- model name
- device name
- number of times the bias surpassed the limit

This information is sent to a file instead of `stderr` if the option `biasfile=filename` is specified in the netlist. Warning messages are issued during or after Transient analysis depending on the value of the option `biaswarn`.

`.BIASCHK` is intended to work with the devices from the table below.

Device	Types	Device's Name	Terminals	Comment
R	R,RES	R	n1, n2	
C	C,CAP	C	n1, n2	
L	IND	L	n1, n2	
V		VSRC	n1, n2	no type
I		ISRC	n1, n2	no type
A		ASRC	n1, n2	no type
F		CCCS	n1, n2	no type
H		CCVS	n1, n2	no type
E		VCVS	nplus, nminus, ncplus, ncminus	no type
G		VCCS	nplus, nminus, ncplus, ncminus	no type
W	CSW, ISWITCH	CSW	n1, n2	
S	SW, VSWITCH	VSW	nplus, nminus, ncplus, ncminus	
O	LTRA	LTRA	n1, n2, n3, n4	
T	T	TRA	n1, n2, n3, n4	
TXL	TXL	TXL	n1, n2, n3, n4	

Device	Types	Device's Name	Terminals	Comment
U	U	U	in1, ..,in5,inref out1,..,out5,outref	
W	WTRA	WTRA	in1, ..,inN,inref out1,..,outN,outref	
M	NMOS, PMOS NTFT,PTFT	MOSFET	nd, ng, ns, nb, nd, ng, ns, np nd, ng, ns, ne, np nd, ng, ns	MOSFET SOI without body contact SOI with body contact TFT
J	NJF, PJF	JFET	nd, ng, ns, nb	
Z	NMF, PMF	MESFET	nd, ng, ns	no bulk terminal
Q	PNP,NPN, LPNP	BJT	nc, nb, ne, ns	BJT/HBT
D	D	DIO	nplus, nminus	

The device and model names obey the wildcard rules (? is for single symbol, * is for 0 or more symbols).

Syntax

```
.BIASCHK type terminal1=termname1 <terminal2=termname2 >
+ limit=val <noise=val> <monitor=v>
+ <name=devname1> <name=devname2> ...
+ <mname=modelname1> <mname=modelname2>
+ <stoptime=val>
+ <tstart=val> <tstop=val>
+ <condition="expression">
+ <duration=val>
+ <detailed>
```

where:

type	Device type.
terminal1=termname1, <terminal2=termname2>	Terminals to check voltage between bias (see table above). If `terminal2` omitted then "0" node will be used by default.
limit=val	User specified limit. Prints a message if the bias voltage between appointed terminals of appointed devices and models is bigger than the limit.
noise=val	User specified level of noise. Default is `0.1V`.
monitor=v	This parameter specifies the type of value to observe. "`v`" means voltage. Can also have value "`i`" (current) - see description of .BIASCHK (Check Current Bias).
name=devname	Device name for which voltage bias is verified.

mname=modelname	All devices of this model are verified.
stoptime=val	If violation duration exceeds specified time limit, simulation is aborted.
tstart=val	Start of time window when BIASCHECK will be performed
tstop=val	End of time window when BIASCHECK will be performed
condition= "expression"	Conditional expression; report violation only if condition is satisfied.
duration	Minimum violation duration to report. Violations with a duration less than a given value will be filtered out. Applies only for BIASCHECKMODE=2.
detailed	Flag to turn on detailed report. In addition to regular output, voltages on device terminals are also printed. Time point index is reported only in case of BIASCHKMODE=1.

If name and mname are not specified in the .BIASCHK statement, all devices of the type are verified for bias voltage. The field type is not required in the .BIASCHK statement if name/mname are specified.

Internal Node Voltage Monitoring

.BIASCHK also has the capability to monitor the voltages between internal nodes of devices (and not only between terminals). The following "terminal" names can be specified on .BIASCHK statements to access internal node voltages:

- For MOSFET devices: ndp (internal drain), nsp (internal source).
- For BSIM4 devices: ngp (internal gate), ngm (second internal gate), nbp (internal bulk), ndb (drain side internal bulk) and nsb (source side internal bulk).
- For SOI devices: nbp (internal body).
- For diodes: nplusp (internal anode).
- For BJT devices: nci/nbi/nei/nsi (intrinsic collector/base/emitter/substrate), ncx/nbx/nex (extrinsic collector/base/emitter), and nbpnp (parasitic PNP base, VBIC only).

Description of Algorithms

.BIASCHK supports three algorithms in SmartSpice. The new option biaschkmode allows you to choose between "basic" algorithm (compatible with HSPICE) and "extra" algorithms (specific to SmartSpice). The "basic" algorithm reports only the peak points where the bias exceeds the limit, is called "local maximum" and corresponds to biaschkmode=0.

The second algorithm reports all time-points at which voltage bias exceeds the specified limit. Both specifications below are valid and turn on the "extra" algorithm:

```
.OPTIONS BIASCHKMODE
.OPTIONS BIASCHKMODE=1
```

The third algorithm reports all periods (time and duration) during which the monitored bias is out of range and is invoked by specifying:

```
.OPTIONS BIASCHKMODE=2
```

This algorithm is based on a simple criterion: the “out-of-range” condition is defined by `|Vbias|>|limit|`. The value of `limit` must be specified on the `.BIASCHK` statement. Here, the value of “noise” is ignored even if specified.

If the option `biaschkmode` is not specified (default), the “basic” algorithm is on.

Below is an example of output from the “extra” algorithm. The decision maker in SmartSpice identifies and prints a bias point accordingly with the rule: if `|bias| > limit+ noise`.

Example

```
.biaschk NMOS terminal1=ng terminal2=nb limit=4.51 noise=0.01
+ name=m1 name=m.x1.x2.m3  mname=NCH
```

Typical output from `.biaschk` is shown below:

```
.biaschk terminal1=n1 terminal2=n2 limit=4.51 noise=0.01
```

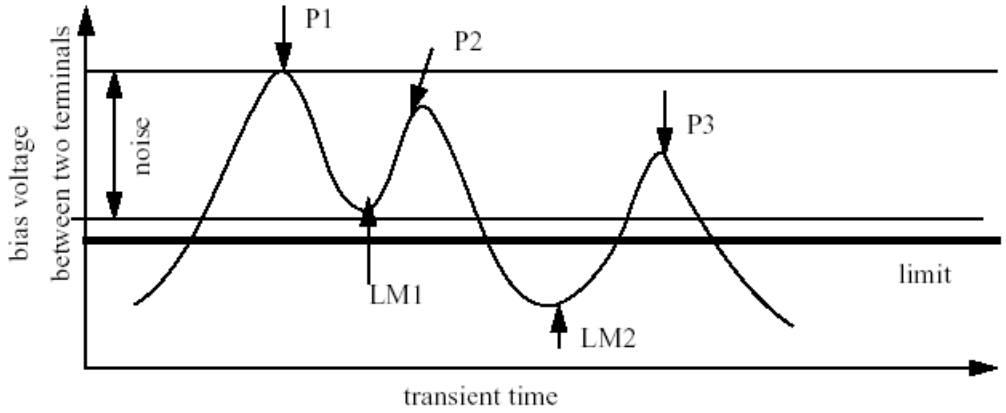
`.BIASCHK` output during Transient analysis:

```
type   parameters   time          bias          model-name element-name
c      n1 - n2      5.0300e-10    4.5202e+00    mod1       c.x1.x1.x4.c1
c      n1 - n2      5.1550e-10    4.5213e+00    mod1       c.x1.x1.x4.c1
c      n1 - n2      5.1550e-10    4.5205e+00    mod1       c.x1.x2.x4.c1
c      n1 - n2      5.4050e-10    4.5234e+00    mod1       c.x1.x1.x4.c1
c      n1 - n2      5.4050e-10    4.5226e+00    mod1       c.x1.x2.x4.c1
c      n1 - n2      5.9050e-10    4.5279e+00    mod1       c.x1.x1.x4.c1
c      n1 - n2      5.9050e-10    4.5269e+00    mod1       c.x1.x2.x4.c1
c      n1 - n2      6.9050e-10    4.5347e+00    mod1       c.x1.x1.x4.c1
c      n1 - n2      6.9050e-10    4.5335e+00    mod1       c.x1.x2.x4.c1
c      n1 - n2      7.9050e-10    4.5347e+00    mod1       c.x1.x1.x4.c1
c      n1 - n2      7.9050e-10    4.5332e+00    mod1       c.x1.x2.x4.c1
c      n1 - n2      8.9050e-10    4.5229e+00    mod1       c.x1.x1.x4.c1
c      n1 - n2      8.9050e-10    4.5210e+00    mod1       c.x1.x2.x4.c1
c      n1 - n2      2.3250e-09    4.5261e+00    mod1       c.x1.x2.x8.c1
c      n1 - n2      2.4250e-09    4.5233e+00    mod1       c.x1.x2.x8.c1
```

Elements that have `.BIASCHK` out of limit during the transient simulation:

```
type     parameters   Number Count     model-name     element-name
c        n1 - n2      7               mod1           c.x1.x1.x4.c1
c        n1 - n2      6               mod1           c.x1.x2.x4.c1
c        n1 - n2      2               mod1           c.x1.x2.x8.c1
```

The “basic” algorithm is much more complicated. If in the “extra” algorithm noise works as tolerance in the “basic” algorithm, noise is intended to eliminate unexpected points from the report. For example, if noise is set to proper value shown below, the peak point P2 will be treated as “noise”, and only P1 and P3 will be reported in the result of bias check.



All peak points are defined as local minimum or local maximum, depending on polarity.

If a local minimum V(LM1) is inside the noise regime that is defined as the region between previously reported local maximum V(P1) and V(P1)-noise, the following local maximum V(P2) will be treated as a noise and will not be reported.

Since second local minimum V(LM2) is out of the noise regime, the following local maximum V(P3) will be detected as a violation against the limit and will be reported.

If setting noise to zero, all local maximums will be reported. On the other hand, if setting noise to a large value, only remarkable peaks will be reported.

Once a local maximum is reported, the noise regime will be re-calculated from that point. In the example above, when P3 was reported, the next noise regime will result from V(P3) to V(P3)-noise, and will be used for evaluating the next local minimum. As a result of such algorithm, `.BIASCHK` cannot detect the bias exceeding the limit if the bias is always the same value during Transient analysis. In this case, the algorithm will detect only the first bias point.

`.BIASCHK` statement always accepts the first local maximum if that maximum is bigger than a user specified limit. Algorithm disregards the polarity and operates only with the absolute values of the bias.

Typical output from `.BIASCHK` “basic” algorithm is shown below.

Example

```
.biaschk   terminal1=nb   terminal2=ns   limit=0.500   noise=0.280
name=mn00
```

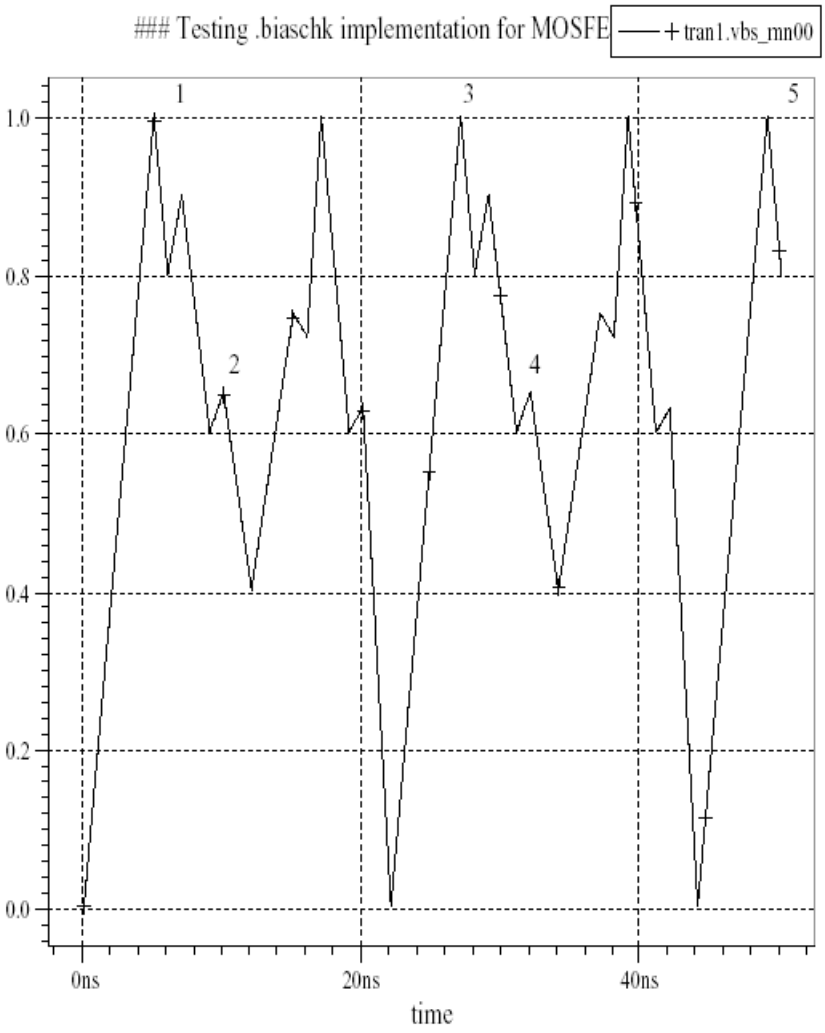
`.BIASCHK` output during Transient analysis (basic algorithm):

```
type    parameters  time          bias          model-name   element-name
nmos    nb - ns     5.0000e-09    1.0000e+00    nm0 mn00     $point1
nmos    nb - ns     1.0000e-08    6.5000e-01    nm0 mn00     $point2
nmos    nb - ns     2.7000e-08    1.0000e+00    nm0 mn00     $point3
nmos    nb - ns     3.2000e-08    6.5000e-01    nm0 mn00     $point4
nmos    nb - ns     4.9000e-08    1.0000e+00    nm0 mn00     $point5
```

Elements that have `.BIASCHK` out of limit during the transient simulation:

```
type    parameters    Number Count    model-name    element-name
nmos    nb - ns      5               nm0           mn00
```

The figure below illustrates the peaks which are identified as violations.

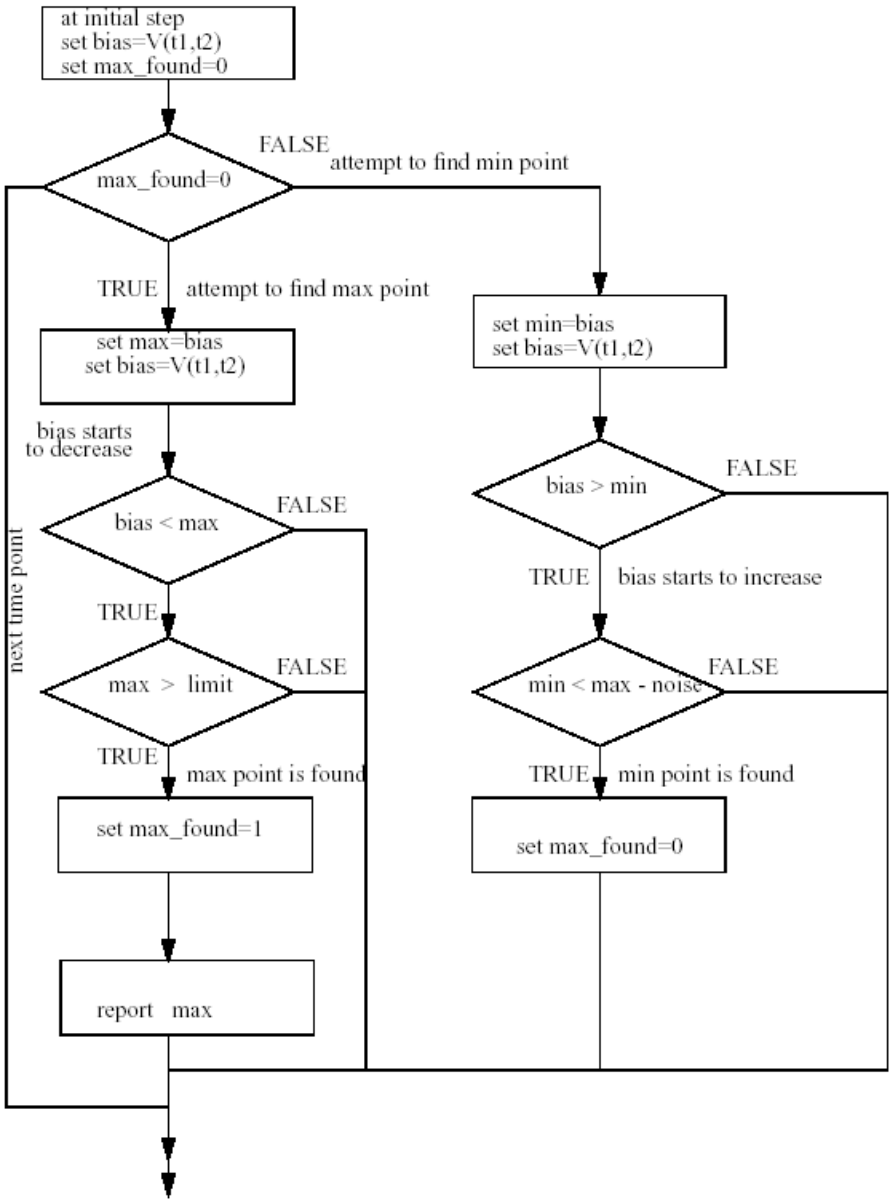


Example

```
.biaschk terminal1=ng terminal2=nb tstart=1ns tstop=1.25ns
+ limit=0.51 noise=0.01 name=*x1.m1
```

In this example, the bias check will be performed in the time window between `1ns` and `1.25ns` during a transient analysis.

The “basic” algorithm works according to the scheme below:



Conditional .biaschk

Condition expression may be used for every device type supported by `.biaschk` as well as subcircuits. If set, a violation is reported only if the provided conditional expression is satisfied. Expression support is generic; any kind of expression is accepted. For node voltages, abstract nodes according to device type being checked may be used as well as existing netlist nodes.

Example

```
.biaschk d terminal1=nplus terminal2=nminus limit=0.5 name=d1
detailed
condition="abs(v(nplus)-v(nminus))>0.48"
.biaschk NMOS detailed condition="abs(v(nd)-v(ns))+abs(v(ns)-
v(nb))>1.8"
terminal1=ngm terminal2=nsb limit=0.5 noise=0.01
```

```
name=m.xnandf.x7.mn1
.biaschk x terminal1=n1 terminal2=n2 limit=0.25 name=x2
condition="abs(v(n1)-v(5))>0.3"
```

Special variables are supported for the following device types:

- MOSFETs: `length`, `width`
- Diodes: `length`, `width`, `area`
- Resistors: `length`, `width`, `rvalue`
- Capacitors: `length`, `width`, `cvalue`

When expression is evaluated, above parameters, if used, are queried on device that is being checked. `length`, `width`, `area` correspond to the relevant device geometry parameters. For resistors, `rvalue` is the effective resistance while for capacitors, `cvalue` is the effective capacitance.

Examples

```
.biaschk nmos terminal1=n1 terminal2=n2 limit=5 mname=nmos1
+ condition="length <= 1u && length <= 2u" detailed
```

This statement checks bias voltage exceeding 5V for model nmos1 devices with length between and equal to 1u and 2u

```
.biaschk d terminal1=nplus terminal2=nminus limit=0.500 name=d*
+ condition="(v(nplus)<v(nminus)) && area!=1"
```

Check reverse biased diodes with area parameter not equal 1.

```
.biaschk r terminal1=n1 terminal2=n2 limit=2 condition="rvalue >
100" name=r1
```

Check resistors named `r1` which effective resistance is bigger than 100 Ohm.

Supports variables in a condition expression for a MOSFET device type:

```
vds, vgs, vbs, gm and vth
```

When an expression is being evaluated, the above parameters, if used, are queried on the device that is being checked.

Example

```
.biaschk NMOS terminal1=ngm terminal2=nsb
+ condition="vbs != 0 && vds > 0.92 && vds < 1.3"
```

Options for .BIASCHK Statement

biasfile

Applying this option causes the output of all `.BIASCHK` statements to be sent to the specified file.

Example

```
.OPTION BIASFILE='mos.bias'
```

biaswarn

If this option is set to `1`, a warning message is sent out instantly if any local bias voltage exceeds the limit during Transient analysis. The results summary is sent out after Transient analysis completion. If this option is set to `0` (the default value), no messages are sent out during Transient analysis. The summary is sent out after Transient analysis completion.

Example

```
.OPTIONS BIASWARN=1
```

biaschkmode

This option can be set to 0 (default), 1 or 2 to invoke different algorithms for voltage monitoring. Refer to the following description of the statement .BIASCHK for further details.

Example

```
.OPTION BIASCHKMODE=1 or .OPTION BIASCHKMODE (“extra” algorithm)
```

biasstop

The controlling option biasstop affects the simulation flow in transient analysis if the internal condition for .BIASCHK is satisfied. Simulation flow will be stopped, resulting in the message “Error: .biaschk: biaschk aborted because .option biasstop is specified”. The granularity of abortion is a time point. That means when all circuit instances (devices or circuit nodes) were checked for the current time point, the analysis will be stopped.

Example

```
.OPTION BIASSTOP
```

BIASSTOPTIME

The controlling option BIASSTOPTIME affects the simulation flow in transient analysis if the internal condition for .biaschk is satisfied. Simulation flow will be stopped indicating a message:

```
Error: .biaschk: biaschk aborted because .option biasstoptime is
specified.
```

Also, final transient analysis time and the value of BIASSTOPTIME will be printed. The granularity of stopping is a time point. That means when all circuit instances (devices or circuit nodes) were checked for current time point analysis will be stopped. Analysis will be stopped if BIASSTOPTIME is reached and a breakdown condition is detected before BIASSTOPTIME. If BIASSTOPTIME is reached, and no breakdown condition has occurred, then transient analysis will overcome the BIASSTOPTIME point and will stop after first breakdown condition is detected.

Example

```
.OPTION BIASSTOPTIME=60ns
```

biaschk output during transient analysis (basic algorithm):

```
type    parameters   time          bias          model-name   element-name
nmos    nd - ns      1.5166e-08    5.0002e+00    modn         m.x7.m3
nmos    nd - ns      1.5166e-08    5.0002e+00    modn         m.x7.m4
nmos    nd - ns      5.3689e-08    5.0029e+00    modn         m.x7.m3
nmos    nd - ns      5.3689e-08    5.0029e+00    modn         m.x7.m4

Error: .biaschk: Simulation aborted because option   BIASSTOPTIME
was specified.

Time=6.008922e-08  biasstoptime=6.000000e-08
```

Note: To include any bias-check file in the input deck, the bias-check file must be saved as a file format without containing text*, for instance, `.inc biaschk_file` or `.include biaschk_include`.

BIASSIGN

This option is specified on the `.BIASCHK` statement and has the following effect:

```
BIASSIGN=0 : report both positive and negative bias
BIASSIGN=1 : report positive bias only
BIASSIGN=-1: report negative bias only
```

Polarity is evaluated as V(terminal1)-V(terminal2)

Example

```
.OPTION BIASCHKMODE=1
.biaschk   terminal1=nb   terminal2=nd   biassign=-1   limit=0.500
noise=0.000
+ name=mn00
.biaschk   terminal1=nb   terminal2=nd   biassign=1   limit=0.500
noise=0.000
+ name=mn00
.biaschk   terminal1=nb   terminal2=nd   biassign=0   limit=0.500
noise=0.000
+ name=mn00
```

.BIASCHK (Check Terminal Subcircuit Nodes)

The .BIASCHK statement supports monitoring of the voltages across terminal subcircuit nodes, and across internal nodes which belong to the same subcircuit. The new type X is used to describe how SmartSpice deals with subcircuits. In some situations, type X can be skipped. Syntax n1, n2 is used for terminal node selection. Terminal nodes are numbered from the left to the right using a sequential index. When a wildcard is used, SmartSpice will not print a message if the terminal node specification is out of range. The .BIASCHK statement syntax for subcircuits is the same as usual for devices, and can be used simultaneously with the .BIASCHK statements for devices.

Examples

Top level:

```
n1    n2    n3    n4    n5   n6    n7    n8
x1    1     2     3     4    bit1  bit2  cin  cout   twobit
```

Subcircuits definitions are:

```
.subckt twobit  ina1 inb1 ina2 inb2  bit1 bit2  cin cout
x1  ina1  inb1  cin  bit1 10    onebit
x2  ina2  inb2  10   bit2 cout  onebit
.ends twobit
.subckt onebit  1   2   3   4   5
x1  1  2  7   nand
x2  1  7  8   nand
.endc
.subckt nand 1 2 3
m1  3  1 vss vss   mosp w=10u  l=1.3u
m2  3  2 vss vss   mosp w=10u  l=1.3u
.endc
```

.BIASCHK statements:

1. `.biaschk x terminal1=n1 terminal2=n2 limit=0.2500 noise=0.0 name = x1`

   This statement causes the biascheck of the voltages for nodes 1 and 2 of the subcircuit twobit.

2. `.biaschk terminal1=n terminal2=n2 limit=0.25 noise=0.0 name = x1*`

   This statement causes the biascheck of the voltages for subcircuits x1, x1.x1.x2, x1.x1.x1, etc.

3. `.biaschk x terminal1=n1 terminal2=n2 limit=0.25 noise=0.0 mname=twobit`

   This statement causes the biascheck of the voltages for all x-calls of the subcircuit twobit.

4. `.biaschk x terminal1=n1 terminal2=n2 limit=0.25 noise=0.0 mname=two*`

   This statement causes the biascheck of the voltages for all x-calls of the all subcircuits beginning from two.

5. `.biaschk terminal1=nl terminal2=n2 limit=0.25 noise=0.0 mname=twobit`

   The statement causes the biascheck of the voltages for devices with model twobit, because type is not specified as X.

6. `.biaschk x terminal1=n10 terminal2=n2 limit=0.25 noise=0.0 mname=twobit`

This statement causes the printing of the error message “.biaschk: no correspondent node for terminal n10 in x1 x-call cell twobit has 8 terminal nodes”.

7. .biaschk x terminal1=n1 terminal2=n2 limit=0.25 noise=0.0 mname=nand

   This statement causes the biascheck of the terminal voltages for all x-calls of subcircuit nand.

8. .biaschk x terminal1=n1 terminal2=n2 limit=0.25 noise=0.0 name=x1.x1*

   This statement causes the biascheck of the terminal voltages for all x-calls containing x1.x1 at the beginning of the name.

9. .biaschk x terminal1=n6 terminal2=n2 limit=0.25 noise=0.0

   This statement causes the biascheck of the second and sixth node voltages for all subcircuits, and silently ignores subcircuits which do not contain 6 terminal nodes (onebit, nand).

10. .biaschk x terminal1=n4 terminal2=n2 limit=0.25 noise=0.0

    This statement is the same as case 9, but onebit subcircuits will be analyzed because n4 and n2 exist. Subcircuit nand will not be included into the analysis.

11. .biaschk x terminal1=x1.x2.12 terminal2=x1.x2.13 limit=0.25 noise=0.0

    This statement causes the biascheck of the voltages for internal nodes of the subcircuit onebit.

12. .biaschk x terminal1=x1.x1.12 terminal2=x1.x1.13 limit=0.25 noise=0.0

    This statement causes the biascheck of the voltages for internal nodes of the subcircuit onebit.

Cases 11 and 12 allow you to analyze the internal nodes of the subcircuits. If the device is represented by a subcircuit, .BIASCHK must be used to analyze the terminal voltages of any components in the subcircuit.

Syntax Extension in x-type of BIASCHK

The .BIASCHK functionality has been enhanced. For x-type (check for circuit nodes), any circuit node can be assigned to terminal1 or terminal2, including the symbols gnd, 0, ground, vss, vdd, etc. If a specified node is not found, SmartSpice stops and prints the following error message:

```
.biaschk x terminal1=x1.x1.12 terminal2=ground limit=0.25
noise=0.0
.biaschk x terminal1=x1.x1.12 terminal2=gnd limit=0.25
noise=0.0
.biaschk x terminal1=x1.x1.12 terminal2=0 limit=0.25
noise=0.0
.biaschk x terminal1=0 terminal2=0 limit=0.25 noise=0.0
.biaschk x terminal1=vss terminal2=x1.x1.12 limit=0.25
noise=0.0
.biaschk x terminal1=vss terminal2=gnd limit=0.25 noise=0.0
.biaschk x terminal1=ground terminal2=gnd limit=0.25
noise=0.0
.biaschk x terminal1=vdd terminal2=gnd limit=0.25 noise=0.0
```

Syntax Extension for device-type of BIASCHK

The .BIASCHK functionality has been enhanced. For device-type (check for device's nodes), any circuit node can be assigned to terminal1 or terminal2, including symbols gnd, 0, ground, vss, vdd, etc. If a specified node is not found, SmartSpice stops and prints the following error message:

```
.biaschk nmos terminal1=ng terminal2=vss limit=4.5 noise=0.01
.biaschk    terminal1=ng  terminal2=vss    limit=4.51    noise=0.01
+name=m1
.biaschk    terminal1=ng  terminal2=gnd    limit=4.51    noise=0.01
+name=m1
.biaschk   terminal1=ng  terminal2=ground   limit=4.51   noise=0.01
+name=m1
.biaschk    terminal1=ng  terminal2=0    limit=4.51    noise=0.01
+name=m1
.biaschk   terminal1=ground  terminal2=vss  limit=4.51  noise=0.01
+name=m1
.biaschk  terminal1=ground terminal2=vssssss limit=4.51 noise=0.01
+name=m1
.biaschk  terminal1=x1.x2.11  terminal2=ng  limit=4.51  noise=0.01
+name=m1
.biaschk  terminal1=ns terminal2=x1.x2.x1.10 limit=4.51 noise=0.01
+name=m1
.biaschk    terminal1=x1.x2.11  terminal2=x1.x2.x1.10  limit=4.51
noise=0.01
 +name=m1
```

.BIASCHK (Check Current Bias)

This .BIASCHK statement is very similar to .BIASCHK (Check Voltage Bias) but intended for tracking and monitoring the current of the device (I(device)) and reporting violations according to the “extra” algorithm. Setting up the limit and the noise in the .BIASCHK statement targets a condition for monitoring current bias of selected device. The .BIASCHK functionality reports the following information during the Transient analysis:

- type of the device
- time at which overreach had happened
- current bias overreaching the limit
- model name
- device name
- number of times the bias surpassed the limit

Description of the Algorithm

The “extra” algorithm (same as biaschmod=1) is used. This algorithm is based on a simple criterion: the "out-of-range" condition is defined by |I(device)| > limit+ noise. The value of limit must be specified on the .BIASCHK statement.

Syntax

```
.biaschk current|monitor=i type
+ limit=val <terminal1=termname1>
+ <noise=val>
+ <name=devname1> <name=devname2> ...
+ <mname=modelname1> <mname=modelname2>
+ <stoptime=val>
+ <detailed>
+ <duration=val>
+ <detailed>
```

current	Keyword for activation the current monitoring.
monitor=i	This parameter specifies the type of value to observe. “i” means current. Can be used instead of current. Can also have value v (voltage) - see description of .BIASCHK (Check Voltage Bias).
type	device type
limit=val	User specified limit. Prints a message if the current of the appointed devices and models is bigger than the limit.
<terminal1= termname1>	Terminal to check current on. If omitted then default device current will be tracked (i(device)).
noise=val	User specified level of noise. Default is 0.
name=devname	Device name for which current bias is verified.
mname=modelname	All devices of this model are verified.
stoptime=val	If violation duration exceeds specified time limit, simulation is aborted.

detailed	Flag to turn on detailed report. In addition to regular output, voltages on device terminals are also printed. Time point index is reported.

Example

```
.biaschk current NMOS limit=0.0001 noise=0.00005 name=m.xnandf.x7.*
+ stoptime=1n detailed
```

Output

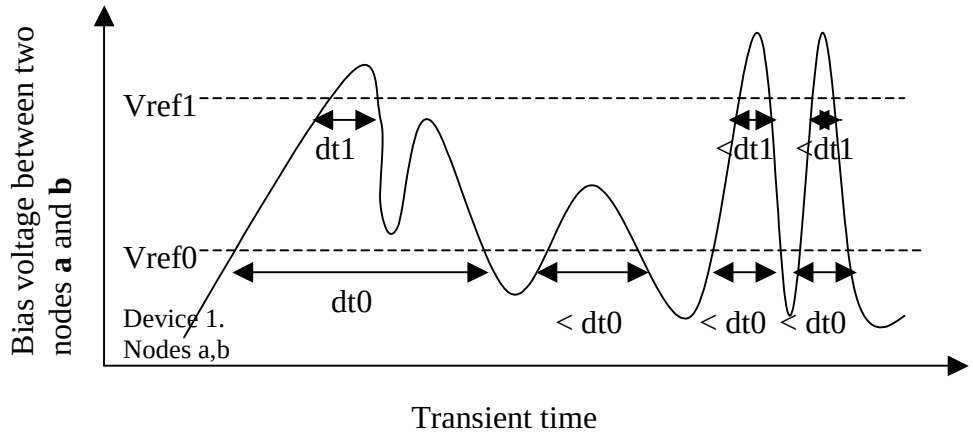
```
type   parameters   time         bias         model-name    element-name
nmos   i            1.4628e-009  1.7136e-004  n             m.xnandf.x7.mn2
[ i:  1.7136e-004 ] [tindex: 153]
nmos   i            1.4728e-009  1.6736e-004  n             m.xnandf.x7.mn1
[ i:  1.6736e-004 ] [tindex: 154]
nmos   i            1.4728e-009  1.9307e-004  n             m.xnandf.x7.mn2
[ i:  1.9307e-004 ] [tindex: 154]
nmos   i            1.4828e-009  1.8706e-004  n             m.xnandf.x7.mn1
[ i:  1.8706e-004 ] [tindex: 155]
nmos   i            1.4828e-009  2.1333e-004  n             m.xnandf.x7.mn2
[ i:  2.1333e-004 ] [tindex: 155]
```

```
type     currents    Number Count   model-name   element-name
nmos     i           40             n            m.xnandf.x7.mn1
nmos     i           43             n            m.xnandf.x7.mn2
```

.BIASCHK3 (Report the Maximum Time Duration of Bias Check)

The `.BIASCHK3` statement is intended to track and report the maximum duration time (dt) when the voltage difference between the specified device nodes or subcircuit terminals is larger than the user-defined reference voltage. It provides syntax to check specified terminals of chosen devices and subcircuits by specifying subcircuit names, hierarchy xcalls, device names, device types and device model names.

Description of Algorithms



This algorithm reports the maximum time duration when the monitored bias voltage is out of range. This algorithm is based on a simple criterion: the “out-of-range” condition is defined by `|Vbias| > |Vref|`. The value of user-defined reference voltage `Vref` must be specified in the `.BIASCHK3` statement. Like the figure above, there are two user-defined voltage references: Vref0 and Vref1. The maximum duration time when `V(a,b)>Vref0` is `dt0`, and `dt1` is the maximum duration time when `V(a,b)` exceeds `Vref1`.

Note: The information is sent to a file if the option `bias3file=filename` is specified in the netlist. See section Options for .BIASCHK3statement below.

Syntax

```
.biaschk3
+ vref0=val1 vref1=val2 ..................... vrefN=val(N+1)

* Inclusion block:


+ <subckt="subcktname_list" |
        ="subcktname_list1",
         "subcktname_list2",
         ...
         "subcktname_listN" |
        ="@hierlevel" |
        ="@all" |
        ="*" |
        ="@top">
```

```
+ <xcall="subcktname_xcall_list" |
       ="subcktname_xcall_list1",
        "subcktname_xcall_list2",
        ...
        "subcktname_xcall_listN" |
       ="@all" |
       ="*" |
       ="X1 X2.X3 X2.X4 ......... ">

+ <devices="subcktname_dev_list" |
         ="subcktname_dev_list1",
          "subcktname_dev_list2",
          ...
          "subcktname_dev_listN" |
         ="@all" |
         ="<active | passive>" |
         ="m* q* r* ....... ">

+ <devicetypes="devicetype_list" |
              ="devicetype_list1",
               "devicetype_list2",
               ...
               "devicetype_listN" |
              ="NMOS PMOS ............." |
              ="@all">

+ <devicemodels="devicemodel_list" |
               ="devicemodel_list1",
                "devicemodel_list2",
                ...
                "devicemodel_listN" |
               ="NCH1 NCH2 ............." |
               ="@ALL">

+ <deviceterminals="devterminal_list" |
                  ="devterminal_list1",
                   "devterminal_list2",
                   ...
                   "devterminal_listN" |
                  ="n1 n2 ............ " |
                  ="@all">

+ <subcktterminals="subcktterm_list" |
                  ="subcktterm_list1",
                   "subcktterm_list2",
                   ...
                   "subcktterm_listN" |
                  ="@all">
```

* Exclusion block:

```
+ exclude={

+ <subckt          =   same construvct as in "include" section>
+ <xcall           =        "         "          "       ">
```

```
+ <devices         =       "       "       "       ">
+ <devicetypes     =       "       "       "       ">
+ <devicemodels    =       "       "       "       ">
+ <deviceterminals =       "       "       "       ">
+ <subcktterminals =       "       "       "       ">

+ }
```

Note: < > indicates optional statements. However, in each `biascheck` statement, at least one of the keywords `devices`, `devicetypes`, `devicemodels` or `subcktterminals` have to be specified, otherwise SmartSpice will report a warning/error for incomplete `biascheck` statement.

Items divider – <space> is used as divider in a group or list. Groups divider – <,> is used as divider between groups or lists.

Example

For subcircuit name `sub1`, `sub2` and `sub3`:

- `sub1 sub2 sub3` is a kind of "list of `subckt` names" uses <space> as item divider.
- `sub1 sub2`, `sub3` is a kind of "list of `subcktname_1`, list of `subcktname_2`, …" uses <space> as item divider and <,> as group divider. In this case, `sub1` and `sub2` are in the same group and `sub3` is in the other group.

Note: The specifications for all keywords support wildcards except `devicetypes` (it only supports @all and *). Device and model names obey the wildcard rules (? is for single character, * is for 0 or more characters including hierarchy path).

`vref0=val1 vref1=val2 … vrefN=valN` (real)	User-specified reference voltages will be used like criterion for reporting maximum `dt`'s; `vref0` should be user specified, `vref1,…, vrefN` are optional.
`subckt` (string)	User-specified names (definitions) of subcircuit will be processed. If not specified, all subcircuit names (definitions) will be processed. This keyword will be processed only when at least one of the key words - `devices`, `devicemodels`, `devicetypes` and `subcktterminals` are specified.

Note: `subckt="@all"` and `subckt="*"` include top level and all subcircuit names. `subckt="@hierlevel"`, `hierlevel` is an integer number to specify hierarchical level.

`xcall` (string)	User-specified `xcalls` of selected subcircuit will be processed. If no specified, all `xcalls` of selected subcircuit will be processed. This key word will be processed only when at least one of the key words - `devices`, `devicemodels`, `devicetypes` and `subcktterminals` are specified.

Note: `xcall ="X1 X2.X3 ..."` specifies `xcall` with hierarchical `xcall` instances.

`devices` (string)	User-specified devices of selected `xcalls` will be processed. If no specified, no devices will be processed.

Example

```
+ subckt ="opamp"
+ devices ="active"
```

This means all actives devices of all `xcalls` of `.subckt` opamp will be analyzed.

Note: `devices ="m*  q*  r*..."` specifies devices with device instances and wild cards.

`devicetypes` (string)	Specifies devices using device type. User-specified device types of selected `xcalls` will be processed. If not specified, no device types will be processed. `devicetypes` follows the "Device types" column in the table Supported Devices, Device Types and Terminals for .BIASCHK3 (see below).

Example

"list of device types" can be "`NMOS PMOS`"

Multiple `devicetypes` are supported.

Example

```
+ devicetypes=" PMOS "
+ devicetypes=" NMOS "
+ devicetypes="PNP"
```

is equal to

```
devicetypes = " PMOS NMOS PNP "
```

`devicemodels` (string)	Specifies the devices using device model name. User-specified device models of selected `xcalls` will be processed. If no specified, no device models will be processed.

Example

```
devicemodels ="NCH1 NCH2 ..."
```

devterminals (string)	Specifies device terminals for bias check. User-specified device terminals of selected devices of xcalls will be analyzed. If no specified, all terminals of selected devices of xcalls will be analyzed. This keyword will be processed only when at least one of the keywords devices, devicetypes or devicemodels is specified. devterminals supports numeration rules. devterminals follows the “Terminals” column in Supported Devices, Device Types and Terminals for .BIASCHK3 (see the following table).

Example

For MOSFETs; devterminals = “n1 n2 n3 n4” when using numeration method, devterminals = “nd ng ns nb” when using names.

subcktterminals (string)	Specifies subcircuit terminals for bias check. User-specified subcircuit terminals of selected xcalls will be analyzed. If no specified, no subcircuit terminal check will be processed. subcktterminals supports numeration rules.

Example

```
subcktterminals=“ n1 n2 … ”
```

when using numeration rules.

exclude	Follows the same rules as inclusion block.

If exclusion blocks is used, at least one of the keywords (devices, devicetypes, devicemodels and subcktterminals) has to be specified to process .biascheck3 statement, otherwise SmartSpice will issue a warning/error for incomplete bias check statement.

Note: For each single .BIASCHK3 statement, “auto adjustment” concept is used. Internally SmartSpice always maps exclusion block to “exclusion plus block” and keeps the “exclusion plus block” as final exclusion block. The “auto adjustment” effect can be seen when exclusion block contains nodes or terminals which are not existed in the inclusion block, SmartSpice will automatically adjust the exclusion block to “exclusion plus block” to ignore nodes or terminals which are not existed in the inclusion block.

If multiple .BIASCHK3 statements are used in a deck, these statements will be treated as independent statements and will be executed one by one.

Supported Devices, Device Types and Terminals for .BIASCHK3

Devices	Device Types	Description	Terminals
R	R, RES	RESISTOR	n1, n2
C	C, CAP	CAPACITOR	n1, n2
L	L, IND	INDUCTOR	n1, n2
V	V	VSRC	n1, n2
I	I	ISRC	n1, n2
A	A	ASRC	n1, n2
F	F	CCCS	n1, n2
H	H	CCVS	n1, n2
E	E	VCVS	nplus, nminus, ncplus, ncminus
G	G	VCCS	nplus, nminus, ncplus, ncminus
W	CSW, ISWITCH	CSW	n1, n2
S	SW, VSWITCH	VSW	nplus, nminus, ncplus, ncminus
O	ltra	LTRA	n1, n2, n3, n4
T	T	TRA	n1, n2, n3, n4
TXL	TXL	TXL	n1, n2, n3, n4
U	U	U	in1, …, in5, inrefout1, …, out5, outref
W	W	WTRA	in1, …, inN, inrefout1, …, outN, outref
M	NMOS, PMOS	MOS or SOI	nd ng, ns, nb (MOSFET) nd, ng, ns, ne(SOI) nd, ng, ns, np, ne (SOI with body)
M	NTFT, PTFT	TFT	nd, ng, ns
J	NJF, PJF	JFET	ng, ns, nb

Devices	Device Types	Description	Terminals
Z	NMF, PMF	MESFET	nd, ng, ns
Q	PNP, NPN, LPNP	BJT or HBT	nc, nb, ne, ns
D	D	DIODE	n1, n2

Output from .BIASCHK3

The .BIASCHK3 statement reports the following information during the transient analysis:

- device name
- device type
- device model
- subcircuit name
- subcircuit definition
- net name of each node
- maximum dt between every two nodes
- maximum dt between each node and global ground

Example

```
.OPTION BIAS3FILE='Mytest1.bias3'
* Expressions and Parameters in vref
.param a = 0.1
.biaschk3 vref0 = 1 vref1 = '1.9 + a' vref2 = 2.2
+ devices = @all
*SmartSpice output:
Output table for maximum delta time between nodes when voltage
difference is larger than user-defined voltage level.

Device name: q.x1.x1.x2.q2
Device type: BJT
Device model: qnl

####Net name of each node
Collector: x1.x1.x2.9
Base: x1.x1.x2.5
Emitter: x1.x1.7
Substrate: 0

####Maximum dt between every two nodes
  Vref0= 1.0000e+00 Vref1= 2.0000e+00 Vref2= 2.2000e+00
|V(Base,Collector)| :        - - -
|V(Collector,Emitter)| :   7.6183e-08 3.0976e-08 3.0069e-08
|V(Collector,Substrate)| : 2.3929e-08 7.6093e-09 -
|V(Base,Emitter)| :    3.2225e-08 2.5532e-08 2.4325e-08
|V(Base,Substrate)| :    2.9197e-08 2.2465e-08 2.1112e-08
|V(Emitter,Substrate)| :   7.8631e-08 7.3826e-08 7.2580e-08

####Maximum dt between each node and global ground
  Vref0= 1.0000e+00 Vref1= 2.0000e+00 Vref2= 2.2000e+00
|V(Collector)| : 2.3929e-08 7.6093e-09 -
|V(Base)| : 2.9197e-08 2.2465e-08 2.1112e-08
```

```
|V(Emitter)| : 7.8631e-08 7.3826e-08 7.2580e-08
...

Subcircuit name: x1.x1.x1
Subcircuit definition: nand
####Net name of each node
1: 1
2: 2
3: x1.x1.7

####Maximum dt between every two nodes
Vref0= 1.0000e+00 Vref1= 2.0000e+00 Vref2= 2.2000e+00
|V(2,1)| : 2.3672e-08 1.6599e-08 1.5253e-08
|V(3,1)| : 3.5128e-08 3.0976e-0 3.0432e-08
|V(3,2)| : 7.1012e-08 6.7746e-086.7746e-08

####Maximum dt between each node and global ground
Vref0= 1.0000e+00 Vref1= 2.0000e+00 Vref2= 2.2000e+00
|V(1)| : 2.3879e-08 1.6856e-08 1.5446e-08
|V(2)| : 3.3382e-08 2.6910e-08 2.5474e-08
|V(3)| : 7.8631e-08 7.3826e-08 7.2580e-08
```

Options for .BIASCHK3statement

bias3file

Syntax

```
.OPTION BIAS3FILE='filename'
```

Applying this option causes the output specified file.

Example

```
.OPTION BIAS3FILE='filename.bias3'
```

Examples

(Assume the reference voltage `Vref0` is `3v` for all examples)

General

1. All device terminals of all devices will be analyzed.

   ```
   .biaschk3   vref0=3  devices='*"
   ```

2. All device terminals of devices `m3`, `m4` and `d1` in `xcalls` `x1`, `x2`, `x3` from subcircuit `"opamp"` will be analyzed.

   ```
   .biaschk3   vref0=3
   +subckt="opamp" xcall="x1 x2 x3" devices="m3 m4 d1"
   ```

3. All device terminals of all mosfets only in `xcalls` starting from `"xanalog"` from subcircuit `"opamp"` will be analyzed.

   ```
   .biaschk3 vref0=3 subckt="opamp"  xcall="xnalog*"
   +devices="m*"
   ```

Specification on Item and Group Dividers

1. All device terminals of all mosfets in `xcalls x1` and `x4` with subcircuit `“opamp”`, `xcall x2` with subcircuit `“opamp1”` and `xcall x3` with subcircuit `“opamp2”` will be analyzed.

```
.biaschk3 vref0=3
+subckt=”opamp,   opamp1,   opamp2”   xcall=”x1   x4,   x2,   x3”
+devices=”m*, m*, m*”
```

Operations with Macro @top

1. All device terminals of all mosfets on top level will be analyzed.

```
.biaschk3 vref0=3
+subckt=”@top” devices=”m*”
```

Operations with Device Terminals

1. Three terminals (`d,g,s`) of all MOSFETs in `xcall x1` from `.subckt “opamp”` will be analyzed.

```
.biaschk3 vref0=3
+subckt=”opamp”  xcall=”x1” devices=”m*”
+devterminals=”n1 n2 n3”
```

Specification on Hierarchy Level

1. All device terminals of diodes in hierarchy level 5 (count from `top`, `top=level1`) will be analyzed.

```
.biaschk3 vref0=3
+subckt=”@5” devices=”d*”
```

Specification on Subcircuit Terminals

1. The first three subcircuit terminals of `xcall x1` with subcircuit name `“opamp2”` and all subcircuit terminals of `xcall x2.x3` with subcircuit name `“opamp3”` will be analyzed.

```
.biaschk3   vref0=3
+subckt=”opamp2, opamp3”  xcall=”x1, x2.x3”
+subcktterminals=”n1 n2 n3, @all”
```

Macros @all, active, passive

1. All active devices in `xcall x1` from subcircuit `“opamp2”` will be analyzed:

```
.biaschk3 vref0=3
+subckt=”opamp2”  xcall=”x1”  devices=”active”
```

2. All devices in all `xcalls` from subcircuits `“opamp2”` and `“opamp3”` will be analyzed:

```
.biaschk3 vref0=3
+subckt=”opamp2 opamp3”  devices=”@all”
```

3. All devices in `xcalls x1`, `x2.x3` and `x4.x5.x6` will be analyzed.

```
.biaschk3 vref0=3
+subckt=”@all” xcall=”x1 x2.x3 x4.x5.x6”
+devices=”@all”
```

Multiple .biaschk

All device terminals of all MOSFETs only in xcalls starting from “xanalog” from subcircuit “opamp” and all device terminals of all BJTs only in xcalls which has three characters (includes “x”) starting from x4 from subcircuit “comparator” will be analyzed.

```
.biaschk3   vref0=3
+subckt=”opamp” xcall=”xnalog*” devices=”m*”

.biaschk3   vref0=3
+subckt=”comparator”  xcall=”x4?” devices=”q*”
```

Additional Examples

The following examples assume that all xcalls are selected and the default values are displayed in <>.

1. Bias check for all device terminals of all active devices.

```
.biaschk3 vref0=3 devices=”active”
<subckt=”@all”>  <xcall=”@all”>  <devterminals=”@all”>
```

2. Bias check for all terminals of all passive devices only.

```
.biaschk3 vref0=3 devices=”passive”
<subckt=”@all”>  <xcall=”all”> <devterminals=”all”>
```

3. Bias check for all terminals of all active devices and passive devices.

```
.biaschk3 vref0=3 devices=@all
```

4. Bias check for all terminals of all active devices of subcircuits “sub1”, “sub2” and “sub3”:

```
.biaschk3 vref0=3 subckt=”sub1 sub2 sub3”
+devices=”active”
```

5. Bias check for all terminals of all active devices under all subcircuits.

```
.biaschk3 vref0=3 <subckt=”@all”>  <xcall=”@all”> +devices=”active”
 <devterminals>=”@all”
+exclude={ subckt=”@top” <xcall=”@all”> +devices=”active”
 <devterminals>=”@all”}
```

6. Bias check for all terminals of all active devices and for all terminals of all subcircuits.

```
.biaschk3 vref0=3 devices=”active” +subcktterminals=”@all”
+exclude={ subckt=”@top” devices=”active” }
```

7. Bias check for all terminals of subcircuits “sub1”, “sub2” and “sub3”.

```
.biaschk3 vref0=3 subckt=”sub1 sub2 sub3”
+subcktterminals=”@all”
```

8. Bias check for all terminals of all active devices and for all subcircuit terminals of subcircuits “sub1”, “sub2” and “sub3”.

```
.biaschk3 vref0=3 devices=”active”
.biaschk3 vref0=3 subckt=”sub1 sub2 sub3”
+sbcktterminals=”@all”
```

9. Bias check for all terminals of active devices of subcircuits “sub1”, “sub2” and “sub3” and for all terminals of all subcircuits.

```
.biaschk3 vref0=3 subckt=”sub1 sub2 sub3”
+devices=”active”
.biaschk3 vref0=3 subcktterminals=”@all”
```

10. Bias check for all terminals of subcircuits “sub1”, “sub2” and “sub3” and for all terminals of all active devices but exclude active devices under subcircuits “sub1”, “sub2” and “sub3”.

```
.biaschk3 vref0=3 subckt=”sub1 sub2 sub3”
+subcktterminals=”@all”
.biaschk3 vref0=3 devices=”active”
+exclude={ subckt=”sub1 sub2 sub3”  devices=”active” }
```

11. Bias check for all terminals of subcircuits “sub1”, “sub2” and “sub3” and for all terminals of all devices but excludes passive devices under subcircuits “sub1”, “sub2” and “sub3”.

```
.biaschk3 vref0=3 subckt=”sub1 sub2 sub3”
+subcktterminals=”@all”
.biaschk3 vref0=3 devices=”@all”
+exclude={ subckt=”sub1 sub2 sub3” devices=”passive”}
```

12. Bias check for all terminals of subcircuits “sub1”, “sub2” and “sub3” and for all terminals of all active devices and all passive devices in top level.

```
.biaschk3 vref0=3 subckt=”sub1 sub2 sub3”
+subcktterminals=”@all”
.biaschk3 vref0=3 devices=”active”
.biaschk3 vref0=3 subckt=”@top”  devices=”passive”
```

13. Bias check for all terminals of all subcircuits but exclude subcircuits “sub1”, “sub2” and “sub3”.

```
.biaschk3 vref0=3 subcktterminals=”@all”
+exclude={subckt=”sub1 sub2 sub3” subcktterminals=”@all”}
```

14. Bias check for all terminals of all active devices of all subcircuits but excludes active devices of subcircuits “sub1”, “sub2” and “sub3”.

```
.biaschk3 vref0=3 <subckt=”@all”> <xcall=”@all”> +devices=”active”
 <devterminals>=”@all”
+exclude={ subckt=” @top sub1 sub2 sub3”
+<xcall=”@all”> devices=”active” +<devterminals>=”@all”}
```

15. Bias check for all terminals of device types “nmos” and “res” (See Supported Devices, Device Types and Terminals for .BIASCHK3).

```
.biaschk3 vref0=3 <subckt=”@all”> <xcall=”@all”> +devicetypes=”nmos
res”
 <devterminals>=”@all”
```

16. Bias check for all terminals of active device with names obeying wildcards “m*” and “q*”.

```
.biaschk3 vref0=3 <subckt=”@all”> <xcall=”@all”> +device=”m*
q*” <devterminals>=”@all”
```

.CHKACELL (Check Cell Activity)

This statement checks activity of cells of a specified subcircuit during transient simulation. A cell is considered active if at least one terminal of the cell is activated at a moment. As a result SmartSpice produces additional vectors inside the corresponding transient analysis plot.

Syntax

```
.chkacell subckt=subckt_name <dv=val> <report=stat|cell>
```

subckt	Subcircuit name to check cell activity.
<dv=<val>	Voltage difference threshold. Subcircuit terminal is considered active if absolute voltage change on that node is greater than `val`. Default is 0.01
<report=<stat|cell>	Specifies reported vectors. • `stat`: there will be created vectors for cells activation quantity and an average of activation in %. • `cell`: in addition to previous activation vector of each cell will be generated. Default value - `stat`.

Reported vectors have following names:

`acell_subckt(subckt_name)`	quantity of activated cells at current point of transient analysis
`acell_average(subckt_name)`	average quantity of the activated cells as a percentage for last period of time
`acell_cel(xcall_of_cell)`	activity of the cell during transient analysis

Examples

```
.chkacell subckt=ram_1x1
```

In this example all cells of `ram_1x1` subcircuit will be checked during transient analysis. Only `stat` vectors will be reported: `acell_subckt(ram_1x1)` and `acell_average(ram_1x1)`

```
.chkacell subckt=RAM_8x8 report=cell dv=0.1
```

In this example all cells of `ram_8x8` subcircuit will be checked. `report=cell` specifies that in addition to statistical vectors there will be generated vector `acell_cel(...)` for each x-call (e.g., `acell_cel(x1.x2.x_8x8)`). Threshold to determine activity of terminals is 0.1 volt.

.CHKANODE (Check Node Activity)

This statement is synonym for .ACHECK command. See Section .ACHECK (Check Node Activity) for syntax definition. The only difference between .ACHECK and .CHKANODE statements is default parameter values to provide FineSim compatibility.

report	Default value is inactive.
type	Default value is gate.

.CONNECT (Connect Nodes Together)

Syntax

```
.CONNECT node1  node2 <node3> ... <nodeN>
```

node1, node2, ..., nodeN	Names of the nodes to be connected.

This statement allows you to connect several nodes without modifying the circuit description.

Example

```
V1  NODE1  0  DC 1v
R1  NODE2  0  R=1
.CONNECT NODE1  NODE2
```

In this example, resistor R1 will be connected to voltage source V1 without modifying the R1 connected nodes name.

.CONTROL (Beginning of Control Part of Input Deck)

Syntax

```
.CONTROL
```

This statement marks the beginning of a control portion of the SmartSpice input deck. All statements between `.CONTROL` and `.ENDC` are executed as interactively issued SmartSpice commands. The control portion is executed immediately after the input deck has been parsed. When there are multiple `.CONTROL` decks in a deck, the last one parsed replaces all others and is the only one executed.

Examples

```
.CONTROL
run
let ids = -i(vds)
plot ids
.ENDC
VDS 1 0
vgs 2 0
vbs 4 0 -1
.DC vds 0 5 0.5 vgs 1 5 1
.PRINT DC I(vds)
.PLOT DC I(vds)
m1 1 2 0 4 nmos1 L=8u W=25u
.MODEL nmos1 NMOS
.END
```

In this example, the deck is parsed, and the `.CONTROL` block is executed. These commands run the simulation, create a new vector `ids`, and plot the vector.

Within a `.CONTROL` block you can read in a rawfile, and generate new vectors through a `.MEASURE` statement. This can save re-simulation time if the initial run did not contain all the required measurements, or additional measurements were required.

.CUTOFFTAB (Cut-off MOS Device Nodes Check)

This statement causes SmartSpice to check and report all shared (source/drain) nodes between any 2 MOS devices operating in cut-off mode. Statement supports OP and Transient analyses.

Syntax

```
.cutofftab mode <novsrcnodes> <gatesonly> <nsp=val>
+ <start=val interval=val <stop=val>
...
+ <start=val interval=val <stop=val>>>
```

mode	Statement operation mode. Possible modes are OP, TRANOP and TRAN:

- OP - perform check during OP analysis.
- TRAN - perform check during transient analysis.
- TRANOP - perform check and report nodes during both OP calculation and transient analysis.

novsrcnodes	Filter out nodes connected to voltage sources during report. If given, vsrc-connected nodes will be omitted. Default is not given.
gatesonly	Report only nodes that have gate connection of any MOS device. If given, nodes that do not have gate connections will be omitted. Default is not given.
nodgnodes	Filter out nodes that connect to both drain and ground of the same transisters.
nsp	Number of second phase passes. During second phase pass SmartSpice analyses resistor devices and MOS devices not in cut-off mode connected between detected cut-off MOS devices. Pull-up/pull-down transistors that could pull cut-off connection are also detected during this phase. The number of passes required for successful detection of such connections should be equal to maximum number of devices that are needed to be detected. If set to 0, second phase will be turned off and only nodes that are directly shared between cut-off MOS devices will be reported. Default is 1.
start	(tran/tranop only) Defines the start of the transient time window where the check is performed.
interval	(tran/tranop only) Defines the interval at which check is performed.
stop	(tran/tranop only) Defines end of the transient time window where check is performed. Value of -1 means end of transient analysis. End of transient analysis is assumed if stop is not given.

Several `start`/`interval`/`stop` settings may be given, each of them will define window and interval at which check will be performed.

Examples

```
.cutofftab op novsrcnodes
```

In this example, check will be performed at the end of OP calculation. Nodes that have VSRC connections will be filtered out from report.

```
.cutofftab tranop start=250n interval=10n stop=320n start=400n
interval=1u gatesonly
```

In this example, the check will be performed at the end of OP calculation and during transient analysis in two defined time windows. The First window causes checks to be performed starting from 250n, each 10n, until 320n is reached. The Second window causes checks each 1u, starting from 400n, and until the end of transient analysis. The `gatesonly` keyword causes a report to contain only nodes that have MOS device gate connections.

```
.cutofftab tran start=0 interval=100n gatesonly novsrcnodes nsp=0
```

In this example, the whole transient analysis will be checked. `nsp=0` setting turns off second phase passes which speeds up check but only directly shared cut-off nodes will be reported.

Note: When pull-up/pull-down transistor is detected, then pulled node is considered to be vsrc-connected. Thus if such nodes should not be reported, add novsrcnodes keyword into the statement.

.CUTOFFTAB with Conditional Extension (cutoffmode=1)

The `.cutofftab` statement with keyword `cutoffmode=1` causes SmartSpice to check and report the nodes that have gate connection to MOS devices operating in given cut-off (by default), linear or saturation mode and MOS device parameters are satisfied by the optional user specified condition.

Syntax

```
.cutofftab mode cutoffmode=1 <cutoff|linear|saturation
+ <condition = "expression">
+ <start=val interval=val <stop=val>
...
+ <start=val interval=val <stop=val>>>
```

where

- optional keywords `cutoff`, `linear`, `saturation` (by default - `cutoff`) specify the required operation mode for MOS devices,
- optional `condition="expression">` specifies the user conditional expression which MOS device parameters should satisfy to being reported, example:

  `condition="abs(vgs-vth)< 1e-3".`

  Conditional expression should obey the standard SmartSpice rules for expressions.

Supported device parameters for MOS: vgs, vbs, vds, gm, vth.

Example

```
.cutofftab tranop cutoffmode=1 start=250n interval=400n start=400n
interval=0.1p stop=1u
```

In this example, the check will be performed at the end of OP calculation and during transient analysis within specified time intervals. Only nodes connected to MOS gates in `cutoff` mode will be reported.

Example

```
.cutofftab tranop cutoffmode=1 start=250n interval=400n start=400n
interval=0.1p stop=1u condition="abs(vgs-vth)< 1e-3"
```

In this example, the check will be performed at the end of OP calculation and during transient analysis within specified time intervals. Only nodes connected to MOS gates in `cutoff` mode will be reported if `abs(vgs-vth)< 1e-3`.

Example

```
.cutofftab tranop cutoffmode=1 saturation start=250n interval=400n
start=400n interval=0.1p stop=1u  condition="abs(vgs-vth)< 1e-4"
```

In this example, the check will be performed at the end of OP calculation and during transient analysis within specified time intervals. Only nodes connected to MOS gates in `saturation` mode will be reported if `abs(vgs-vth)< 1e-4`.

Output

The typical report contains the following information:

```
.Cutofftab nodes table(gatesonly, Mode, 'MOS_mode'), Analysis
<Condition>
.Cutofftab nodes table(gatesonly, Mode, 'MOS_mode'), Analysis
<Time1>
```

```
Node name:
Device name:
Node voltage:
Region:
<param1_value:>
<param2_value:>
. . .
vgs:
vth:
vds:
.Cutofftab nodes table(gatesonly, Mode, ‘MOS_mode’), Analysis
<Time2>
Node name:
Device name:
Node voltage:
Region:
<param1_value:>
<param2_value:>
. . .
vgs:
vth:
vds:
```

where

`gatesonly`	Report will contain the nodes that have gate connection to MOS devices,
`Mode`	extended mode `Cutoffmode=1`
`MOS_mode`	MOS device operating mode (Cut-off, Saturation, Linear), specified in the `.cutofftab` statement
`Condition`	current conditional expression, specified in the `.cutofftab` statement
`Analysis`	current analysis name
`Time1`	current analysis time
`Node name`	name of node connected to MOS gate (and not connected to the ground)
`Device name`	MOS device name
`Node voltage`	node `Node name` voltage
`Region`	current MOS device operation mode
`param1_value, param2_value`	parameter’s `param1, param2 ,` … values presented in the conditional expression
`vgs, vth, vds`	these parameters will be always included in the report

Note: Parameter `param1` should be specified in the condition expression `<condition="expression">` to be included in the report. By default, if condition is not specified in the statement, the following parameters will be included in the report: “vgs”, “vth”, “vds”.

The following output corresponds to the first example mentioned above:

```
.Cutofftab nodes table (gatesonly, Cutoffmode = 1, 'Cut-off'),
Transient analysis
 Condition = 'no'
.Cutofftab nodes table (gatesonly, Cutoffmode = 1, 'Cut-off'),
transient analysis, time= 0.00e+00
 Node name: g2
 Device name: m.x2.mn1
 Node voltage: 0.0000000000000000e+00
 Region: Cut-off
 vgs: 0.0000000000000000e+00
 vth: 1.7361125254133974e-01
 vds: 1.7152005257630927e+00
 ...
.Cutofftab nodes table (gatesonly, Cutoffmode = 1, 'Cut-off'),
transient analysis, time= 3.02e-07
 Node name: g2
 Device name: m.x2.mn1
 Node voltage: 0.0000000000000000e+00
 Region: Cut-off
 vgs: 0.0000000000000000e+00
 vth: 1.7361125254133974e-01
 vds: 1.7152005285590852e+00
 ...
```

Extended syntax to filter subcircuits, xcalls and nodes using the parameters `subckt`, `xcall` and `filterout_nodecondition` in the `.cutofftab` statement.

Syntax

```
.cutofftab mode cutoffmode=1 <cutoff|linear|saturation
+ <condition = "expression">
<subckt=”list of subcicruit names” | subckt=@hierlevel |
+ subckt=”@all” |
+ subckt=”*”
+ xcall=”list of subcircuit X calls” | xcall=”@all” | xcall=”*” >
+ filterout_nodecondition=”expression”
```

where,

`subckt`	User-specified names (definitions) of subcircuit will be processed. If no specified, all subcircuit names (definitions) will be processed.
`xcall`	User-specified xcalls of selected subcircuit will be processed. If no specified, all xcalls of selected sub-circuit will be processed.

filterout_nodecondition	Specifies the user node(gate) conditional expression. When node expression returns true MOS device will be excluded from the .cutofftab report.

Example

```
filterout_nodecondition="((abs(vg))< 1e-3)"
```

In this example the parameter vg is used as MOSFET gate voltage.

Node conditional expression should obey the standard SmartSpice rules for expressions.

Device MOS parameters vgs, vbs, vds, gm and vth can be used in node condition expression.

Example

```
.cutofftab tran cutoffmode=1 start=10n interval=50n
+ subckt="inv1"
+ xcall="xu3.xu"
+ filterout_nodecondition=" (vg<vgmin3)"
+ xcall = "xu4.xu"
+ filterout_nodecondition=" (vg<vgmin4)"
```

In this example SmartSpice will check and report only the nodes that have gate connection to MOS devices inside of subcircuit inv1. Subcircuit calls xu3.xu and xu4.xu have unique node expressions.

.DATA (Data Statement)

Vector .DATA Statement

Syntax

```
.DATA dataname
+ swname targ1 <targ2 ...>
+ swval1 val11 <val21 ...>
+ swval2 val12 <val22 ...>
 ...
+ swvaln val1n <val2n ...>
```

or

```
.DATA dataname COMPLEX
+ swname targ1 <targ2 ...>
+ swval1 Re11 Im11 <Re21 Im21 ...>
+ swval2 Re12 Im12 <Re22 Im22 ...>
 ...
+ swvaln Re1n Im1n <Re2n Im2n ...>
```

The .DATA statement is used to specify target curves for function optimization and for parametric analysis. These curves can represent:

- Results of physical measurements.
- Theoretical curves.
- Output curves from device simulators such as S-PISCES, BLAZE, etc.

dataname	The data name.
COMPLEX	This parameter indicates that the real and imaginary parts of the target curve are specified.
swname	The name of a real swept variable.
targ1, ...	The names of the target curves.
swval1, ...	The swept variable values. The swept variable data can contain several segments of the same size with the same start and end values.
val11	The target curve targ1 value corresponding to the swept variable value swval1.
val21	The target curve targ2 value corresponding to the swept variable value swval1.
Re11	The real part of the target curve targ1 value corresponding to the swept variable value swval1.
Im11	The imaginary part of the target curve targ1 value corresponding to the swept variable value swval1.

The .DATA statement swept variable values must be consistent with the analysis statement swept variables.

Each target curve specified in a .DATA statement is stored, and can be referred to in .MEASURE, .PRINT and .PLOT statements.

Examples

```
.DATA cur_ids
+ swvds targ
+ 1v 5e-4
+ 2v 2e-3
+ 1v 1e-3
+ 2v 4e-3
.DC vds 1v 2v 0.5v vgs 2v 3v 1v
.MEASURE DC curerr ERR2 targ i(vds) NEST=-1
```

In this example, the .DATA statement defines the target curve targ as a function of the swept variable swvds. The swept variable data consists of two segments. Each segment contains the same swept variable values (1V and 2V).

The .DC statement sweeps the voltage source vds from 1V to 2V in increments of 0.5V for two vgs values. Each of these vgs values correspond to a segment in the .DATA curve. At vds=1.5V, the target curve values are calculated by linear interpolation. They are equal to *1.25e-3* for vgs=2V and *2.5e-3* for vgs=3V.

The .MEASURE statement calculates the difference between target and simulated curves, and stores it under the name curerr. The target curve name is referred to as v(targ). The name of the simulated current is i(vds). The parameter NEST=-1 causes the .MEASURE statement to calculate the difference curerr for the entire nested sweep.

Inline Parametric .DATA Statement

The parametric .DATA statement is used to perform the parametric analysis referred to in the nested SWEEP specifications within the .AC, .DC and .TRAN statements.

Syntax

```
.DATA dataname
+ pnamel <pname2>... <pnamei>... <pnamen>
+ pval11 <pval12>... <pval1i>... <pval1n>
+ pval21 <pval22>... <pval2i>... <pval2n>
+ pvalk1 <pvalk2>... <pvalki>... <pvalkn>
+ pvalp1 <pvalp2>... <pvalpi>... <pvalpn>
.ENDDATA
```

.DATA	This specifies p rows of values for n parameter labels.
dataname	The name of the .DATA statement.
pnamei	Where "i" is the index parameter in the parametric .DATA statement (e.g., pnamel pname2... pnamei). This parameter label is declared in a global .PARAM statement.
pvalki	The kth value of the ith parameter label in the array of values.
.ENDDATA	This statement terminates the parametric .DATA definition.

The minimum size of the data set defined within the parametric .DATA statement is one value for one parameter. If it is necessary to split a .DATA statement into several lines, a "+" may be

used (for backward compatibility but is no longer required) at the beginning of the new line. The field dataname, and at least one parameter, must be provided. Each field name not featuring a leading alphabetic character is considered a parameter value.

The parametric .DATA statement must be followed by the .ENDDATA card. The name dataname of the parametric .DATA statement is only referred to in the nested SWEEP specifications within the .AC, .DC and .TRAN statements. The .DATA statement is defined before or after being referred to in a SWEEP specification. The input deck can contain two or more parametric .DATA statements with unique names.

If an inline .DATA/.ENDDATA statement contains a parameter label that was not previously initialized in a global .PARAM statement, then this parameter label will be appended to the list of global parameters and will be initialized to *1.0e-7*.

Note: Unlike the vector .DATA statement used to specify vectors (curves) for the function optimization, the parametric .DATA statement defines the values of the parametric labels, and is terminated by the .ENDDATA card.

Example

```
.PARAM v1=1.5V wp=5u lp=0.8u c2=3nf r3=2k
.DATA datn1 v1 c2 r3
+  1V 3.5nf 2k
+  2V 7nf 2.5k
.ENDDATA

.DATA datn2 wp lp
+  5.5u 0.82u
+  4.5u 0.78u
.ENDDATA
```

In this fragment of an input deck, two parametric .DATA statements are defined. The statement datn1 defines two rows of values matching the parameter labels v1, c2 and r3.

The statement datn2 defines two rows of values for the parameter labels wp and lp. The first row is:

```
wp = 5.5u, lp = 0.82u
```

and the second is:

```
wp = 4.5u, lp = 0.78u
```

The .DATA statements:

```
.DATA datn3 wp lp 5.5u 0.82u 4.5u 0.7u
.ENDDATA
```

and:

```
.DATA datn3 wp lp
+ 1p 5.5u 0.82u
+ 4.5u 0.78u
.ENDDATA
```

have the same effect as datn2.

File Parametric .DATA Statement

The file parametric .DATA statement can be used where the parametric .DATA statement is used. Instead of specifying parameter values in an inline fashion, files are specified to be read. Values in files are arranged in columns. There are two ways of controlling the way data is merged and matched against parameters:

Concatenated File Parametric .DATA Statement

Data files featuring the same number of columns are appended one after another.

Syntax

```
.DATA dataname MER
  FILE=file1 pname1=position1 ... pnamei=positioni
  <FILE=file <pn=positionn>>
  <OUT=fileout>
.ENDDATA
```

dataname	Identifies the data set.

The parameter MER (or MERGE) identifies a concatenated file parametric .DATA statement. The parameter FILE is used to specify the file to be used. At the first FILE statement, parameters must be declared, and you must specify which column they will match in the input file. Other FILE statements can be specified, and parameter column positions of previously declared parameters can be changed. The OUT statement specifies the file that will hold the result of the concatenation.

For concatenated file parameter .DATA statements, the "+" character is not required as a leading character for the lines specified between the .DATA and .ENDDATA statement.

Example

Given three files f1, f2 and f3 containing the following data:

```
   f1      f2      f3
a b c   A B C   x y z
a b c   A B C
        A B C
```

and the following concatenated .DATA statement specification:

```
.DATA data1 MER
   FILE=f1 p1=1 p2=2 p3=3
   FILE=f2 p1=2 p2=1 p3=3
   FILE=f3 p1=3 p2=1 p3=2
   OUT=fo
.ENDDATA
```

This yields the following parameter/value assignments in the data set `data1`:

p1	p2	p3
a	b	c
a	b	c
B	A	C
B	A	C
B	A	C
z	x	y

and the content of the `OUT` file `'fo'` will be:

```
a b c
a b c
B A C
B A C
B A C
z x y
```

Column Laminated Parametric .DATA Statement

Instead of specifying files to be concatenated, you can specify files featuring the same number of rows to have their columns arranged side-by-side.

Syntax

```
.DATA dataname LAM
  FILE=file1 pname1 = position1 ... pnamei=positioni
 <FILE=file2 pnamei+1 = positioni+1 ...
             pnamei+j = positioni+j>
  <OUT=fileout>
.ENDDATA
```

dataname	Identifies the data set.

The parameter `LAM` specifies a column laminated parametric file `.DATA` statement. The parameter `FILE` is used to specify a file to be used, followed by parameter name and their matching columns positions within the file. The `OUT` statement specifies a file that will hold the result of the column lamination.

The “+” character is not required as a leading character for the lines specified in between the `.DATA` and `.ENDDATA` statement.

Example 1

Given three files f1, f2 and f3 containing the following data:

```
      f1          f2       f3
a1 b1 c1      A1 B1      x1
a2 b2 c2      A2 B2      x2
```

and the following column laminated .DATA statement specification:

```
.DATA data1 LAM
  FILE=f1 p1=1 p2=2 p3=3
  FILE=f2 p4=2 p5=1
  FILE=f3 p6=1
  OUT=fo
.ENDDATA
```

This yields the following parameter/value assignments in the data set data1:

```
p1    p2    p3    p4    p5    p6
a1    b1    c1    B1    A1    x1
a2    b2    c2    B2    A2    x2
```

and the content of the OUT file 'fo' is:

```
a1    b1    c1    B1    A1    x1
a2    b2    c2    B2    A2    x2
```

Example 2

Input deck:

```
.DATA sweep syntax
  vin 1 0 dc 1 ac acmag=1 acphase=0
  r1  1 2 Value
  r2  2 0 100
  c1  2 0 10n

.PARAM parr=100

.DATA name Value
+          100
+          200
+          500
.ENDDATA

#COM
.DATA name LAM FILE=data_file.dat Value=1
.ENDDATA
#ENDCOM

.ac dec 1 10K 100K DATA=name
.dc vin 1 2 1 DATA=name
.tran 1n 3n DATA=name

.print Value; @R1[reff]

.save all(v)
```

```
.end
```

data_file.dat

```
150
250
600
```

Run the input deck and it will use the `.DATA` values `100, 200, 500`. Comment out the `#COM, #ENDCOM` to activate external file data and values `150, 250, 600` will be used.

.DC (DC Analysis)

Syntax

```
.DC <LIN> swname <START=>swstart <STOP=>swstop
+ <STEP=>swincr <nest_sw> <CALLV> <SAVEV <=swsave>>
+ <speedplot> <hysteresis=yes|no> <sw2_spcf>
+ <readic="FILENAME">
+ <readns="FILENAME">
```

or

```
.DC DEC|OCT swname <START=>swstart <STOP=>swstop np <nest_sw>
+ <CALLV> <SAVEV <=swsave>> <speedplot> <hysteresis=yes|no>
+ <sw2_spcf> <<SWEEP> MONTE=val <PRMC<=val>>>
+ <readic="FILENAME">
+ <readns="FILENAME">
```

or

```
.DC LIST swname value1 <value2...> <nest_sw> <CALLV>
+ <SAVEV<=swsave>> <speedplot> <hysteresis=yes|no> <sw2_spcf>
+ <<SWEEP> MONTE=val <PRMC<=val>>>
+ <readic="FILENAME">
+ <readns="FILENAME">
```

or

```
.DC swname DEC|OCT|LIN np swstart swstop
+ <nest_sw> <CALLV> <SAVEV<=swsave>> <speedplot>
+ <hysteresis=yes|no> <sw2_spcf>
+ <<SWEEP> MONTE=val <PRMC<=val>>>
+ <readic="FILENAME">
+ <readns="FILENAME">
```

or

```
.DC swname swstart1 swstop1 swstep1
+ <swstart2 swstop2 swstep2>
+ <swstart3 swstop3 swstep3>
+ <hysteresis=yes|no>
+ <readic="FILENAME">
+ <readns="FILENAME">
```

The `.DC` statement computes a DC transfer curve for the circuit with capacitors opened and inductors shorted.

LIN	Linear sweep (default sweep type).
DEC	Sweep by decades.
OCT	Sweep by octaves.

swname	The name of the argument to be swept. It can be: • a parameter label defined in a global .PARAM statement. • the name of an independent voltage or current source. • the name of a device parameter. • the name of a model parameter. • the parameter TEMP (temperature sweep).
<START=>swstart	The starting value of the sweep argument for LIN, DEC and OCT sweeps. The parameter START is optional.
<STOP=>swstop	The final value of the sweep argument for LIN, DEC and OCT sweeps. The parameter STOP is optional.
<STEP=>swincr	The increment value of the sweep argument for LIN sweep. The parameter STEP is optional.
nest_sw	The second (nested) sweep specification. The rules for the type and values of the nested sweep are the same as for the first sweep.
CALLV	Causes SmartSpice to call the previously saved operating point before starting the DC transfer curve calculation.
SAVEV	Causes SmartSpice to save the operating point calculated for the argument value swsave of the first (inner) sweep.
speedplot	Allows you to speed up simulation by reusing previously created SmartSpice structures (plot). Only one plot will be created, the waveforms will not be preserved. All measure results will be printed and preserved in the .meas plot. The raw files will contain only the data from the last sweep step and all measure results, if there is no .probe statements in the netlist. Default is off.
swsave	The argument value of the first sweep at which the calculated operating point is to be saved (the default value is swstart).
np	Number of points per decade or per octave, or number of linear sweep points.
LIST	Sweep over a list of values.
value1, value2, ...	The values of the sweep argument for the LIST sweep.
sw2_spcf	The second (nested) sweep specification. It is specified in one of the following ways: • SWEEP swname swstart swstop swincr • SWEEP swname LIST nplist swval1 swval2 ... • SWEEP swname swtype np swstart swstop • SWEEP MODIF = dataname <PRTBL> • <SWEEP> DATA=dataname where:

swname	The name of the parameter to be swept. It can be: • a parameter label defined in a global .PARAM statement. • the name of an independent voltage or current source. • the full name of a device parameter. • the full name of a model parameter. • the parameter TEMP (temperature sweep).
swstart	The starting value of the swept parameter.
swstop	The stop value of the swept parameter.
swstep	The increment value of the swept parameter.
swincr	The increment value for a linear sweep.
nplist	Number of parameter values swval1, swval2, ... specified for a sweep of the type LIST.
swtype	The parameter that defines the type of sweep: • DEC • OCT • LIN
np	The number of points per decade or per octave, or the number of points for a LIN sweep
MODIF=dataname	This references the parametric .DATA statement.
DATA=dataname	This references the parametric, .DATA statement.
PRTBL	This optional parameter controls printing of the final table of all modified parameter labels and calculated .MEASURE results (default is OFF).
<SWEEP> MONTE=val	Causes SmartSpice to perform Monte-Carlo statistical analysis, where val is the number of Monte-Carlo repetitions. Parameters of Gaussian, Uniform or Limit function distributions are set by the .PARAM statements: • .PARAM parname=UNIF(nomval, relvar <,mult>) • .PARAM parname=AUNIF(nomval, absvar <,mult>) • .PARAM parname=GAUSS(nomval, relvar, sigma <,mult>) • .PARAM parname=AGAUSS(nomval, absvar, sigma <,mult>) • .PARAM parname=LIMIT(nomval, absvar), where:
parname	Parameter name whose value is calculated by distribution function.
UNIF	Uniform distribution function with relative variation.

AUNIF	Uniform distribution function with absolute variation.
GAUSS	Gaussian distribution function with relative variation.
AGAUSS	Gaussian distribution function with absolute variation.
LIMIT	Random limit distribution function with absolute variation; +/- absvar is added to nomval based on whether the random outcome of a -1 to 1 distribution is greater or less then 0.
nomval	Nominal value for Monte-Carlo analysis.
absvar	The absolute variation: the AUNIF and AGAUSS vary nomval by +/- absvar.
relvar	The relative variation: the UNIF and GAUSS vary nomval by +/- nomval x relvar.
sigma	The parameter for Gaussian distribution. The absvar or relvar is specified at the sigma level. The effective standard deviation of a random sample is absvar/sigma.
mult	The multiplier repeats function calculation mult times, saving only the largest deviation. Default is 1.
hysteresis=yes|no	Performs the DC hysteresis sweep. When hysteresis=yes, a reverse sweep will automatically be performed.SmartSpice will create 2 waveforms on one plot: direct and reverse. Default is off.

Example

```
V1 1 0 DC 5
V2 2 0 DC 0

Q1 7 2 0 QNL
Q2 3 2 7 QNL
Q3 1 3 7 QNL
R1 1 3 10K

.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF CJC=2PF
VA=50)

.DC V2 0.6 1 0.001 HYSTERESIS=YES

.PROBE DC V(3)

.OPTIONS NOMOD NODECK POST=2 PROBE

.END
```

This example will produce next output waveform:

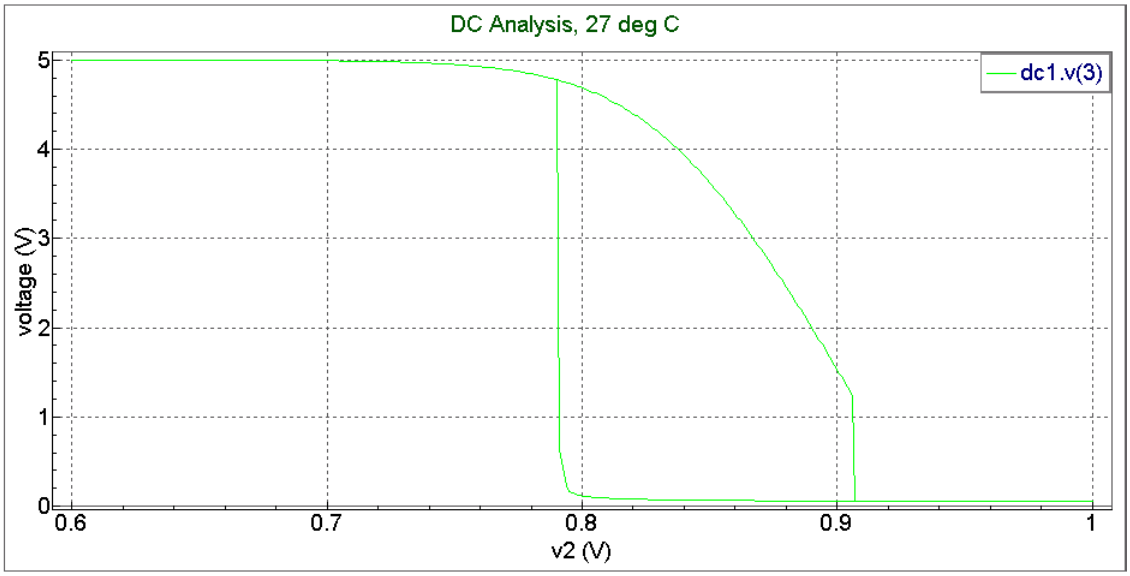


Figure 3-5 Hysteresis

The name and type of measurements, and names of output variables for Monte-Carlo analysis are set by .MEASURE statements.

PRMC<=val>	Causes SmartSpice to print the value of modified and measured parameters on each Monte-Carlo iteration. The PRMC=2 saves all parameters in the raw file, if the option sweepmonte is present. The feature doesn't work in HSPICE compatible mode.
swname	The name of the argument to be swept.
swstart1, swstart2, swstart3	The starting values of the sweep argument for LIN sweeps.
swstop1, swstop2, swstop3	The final values of the sweep argument for LIN sweeps.
swstep1, swstep2, swstep3	The increment values of the sweep argument for LIN sweeps.

This syntax allows SmartSpice to sweep the variable swname for a few consecutive sets (from swstart1 to swstop1, then from swstart2 to swstop2, then from swstart3 to swstop3 with corresponding increment values).

Example

```
.OPTIONS POST NOMOD NOPAGE
.param loadi=1u
I1 gnd in loadi
r1 in out 0.5
r2 out gnd 0.5
.dc loadi 1u 10u 1u 20u 1m 10u 2m 80m 1m
.print V(in) V(out)
```

```
.end
```

SmartSpice calculates the DC transfer curve, sweeping the parameter `loadi` from 1uV to 10uV in increments of 1uV, then sweeping the parameter `loadi` from 20uV to 1mV in increments of 10uV, and then sweeping the parameter `loadi` from 2mV to 80mV in increments of 1mV.

Syntax

```
.DC swname   POI   np  val .....
```

swname	Name of the argument to be swept.
POI	Keyword to indicate that the sweep type is a list of points.
np	Number of points in list.
val...	The argument's values.
readic	Loads a state file "FILENAME" as an initial condition.
readns	Loads a state file "FILENAME" as a node set.

Note: A state file contains nodes + voltages and header information. One example of a state file would be that produced from a `.SAVE BIAS` statement

Example

```
.dc temp poi 4 0 5 10 15
```

In this example SmartSpice runs a DC analysis at four temperatures: 0, 5, 10 and 15°C.

Note: The `.DC` statement contains either the `nest_sw` or the `sw2_spcf` specification, but not both.

Final Value of the Sweep Argument for DC Analysis

Previously, when running a DC analysis, if the end point was not consistent with the start point plus an integer number of steps there would be a residual difference to resolve.

Example

```
.DC v1 START=0 STOP=0.8499 STEP=0.01
```

In this example the final sweep value computed is `0.84` based on start and step increment.

This therefore leaves a residual difference between the final step value and the specified stop value in the analysis to be accounted for.

SmartSpice has been changed for OCTAVE, DECADE or LINEAR DC analysis sweeps to add an additional point equal to the final stop value. The specified value could otherwise be missed because it is inconsistent with the specific start, stop and step values.

In the example above, the final sweep argument value will be exactly `0.8499`, as specified in the `DC` statement.

When a nested sweep is specified, the entire first sweep is repeated for each value of the second sweep.

Examples

```
.DC VIN 1 6 1
.DC VIN 1 6 1 SWEEP TEMP 0 50 10
.DC VIN 0.25 5.0 0.25 QNL(BF) 80 100 10
+ CALLV SAVEV=0.25
```

In the first example, SmartSpice calculates the DC transfer curve, sweeping the voltage source `VIN` from `1V` to `6V`, in increments of `1V`.

In the second example, SmartSpice sweeps the temperature from `0` to `50` in `10` degree increments.

In the third example, SmartSpice sweeps the voltage source `VIN` for three model parameter values. `QNL` is the name of the BJT model selected by the user. `BF` is the name of the model parameter. `QNL(BF)` is the full-path name of the model parameter. The parameter `SAVEV=0.25` causes SmartSpice to store the bias point calculated for the input voltage value of `0.25V`. The parameter `CALLV` loads the previously calculated bias point. The bias point is used as the initial condition at the input voltage value of `0.25V`.

.DCMATCH

See Chapter 19 Mismatch Analyses.

.DCVOLT (Transient Initial Conditions)

Syntax

```
.DCVOLT V(node1)= val1 <V(node2) = val2...>
```

Can be used as an alias for `.IC`, and is subject to the same subcircuit expansion rules.

.DEFINE (User-defined Functions)

Syntax

```
.DEFINE <name>(<arg1<, arg2 <...>>>) <=> body
.DEFINE MYFUN(x)  2*x+1
```

name	Define name.
arg1, arg,...	Define arguments.
<=>	Optional symbol.
body	Function body or expression.

This statement can be used to define functions for use with behavioral devices.

.DEFPARAM (Define Parameters on a Level)

The .DEFPARAM statement allows you to override a parameter on any level, given the fully qualified path to the level.

Examples

```
.defparam x1.x4.wn=10u
.defparam l=5u
.defparam x1.r2.res=100k
```

The first example overrides the wn parameter on the level x1.x4. The second example overrides the l parameter on the top level. The third example overrides the res parameters on the resistor r2 on level x1.

.DEGDIRECTION (Degradation Simulation Statement)

Syntax

```
.DEGDIRECTION 0|1
```

Specifies a direction to calculate degradation. By default, degradation calculation uses forward direction.

0	Forward
1	Reverse

Example

```
.DEGDIRECTION 1
```

Note: This Statement is prepared for future use.

.DEGINFO (Degradation Simulation Statement)

Syntax

```
.DEGINFO <YES/NO> Mxxx <Myyy> ...
```

Mxxx <Myyy>	transistor names

Note: This Statement is prepared for future use.

.DEGRADATION (Degradation Simulation Statement)

Syntax

```
.DEGRADATION time
```

The `Time` value will also take the same unit keywords (`s`, `m`, `d`, etc.) as the Time value for the `.AGE` statement.

Note: This Statement is prepared for future use.

.DEGTIME (Degradation Simulation Statement)

Syntax

```
.DEGTIME Tdegstart Tdegstop
```

This statement limits calculation of a circuit degradation between time points `Tdegstart` and `Tdegstop`. By default, `Tdegstart` is equal to `Tstart`, and `Tdegstop` is equal to `Tstop`, where `Tstop` and `Tstart` are parameters from a transient analysis.

Example

```
.DEGTIME 10ns 90ns
```

Note: This Statement is prepared for future use.

.DEL LIB (Remove Library)

Syntax

```
.DEL LIB filename entryname (syntax_form=0)
```

or

```
.DEL LIB filename (syntax_form=1)
.ALTER and .DEL LIB with syntax_form=1
```

It is possible to use the .DEL LIB and .LIB statement in the .ALTER statement sections of the input deck with syntax_form=1.

filename	Name of the library file.
entryname	The entry name in the library file filename to be removed.

The syntax_form=0 .DEL LIB statement is used with the .ALTER statement to remove any reference to entryname from filename that was previously defined using the .LIB statement. The syntax_form=1 .DEL LIB statement is used with the .ALTER statement to remove any reference to a library filename previously defined using the .LIB statement. A new .LIB statement can be used to replace the removed entry or file. For syntax_form=0, the filename can be enclosed in single or double quotes, or the quotes can be omitted.

Example

```
. . .
.LIB '/usr/examples/lib1' mods2
*
.alter
.DEL LIB '/usr/examples/lib1' mods2
.LIB 'lib.new' mods3
*
.alter
.DEL LIB 'lib.new' mods3
.LIB '/usr/examples/lib2' mods2
*
.END
```

In this example, the circuit runs three times, each time using a different library entry. The first time, the entry mods2 from the library file /usr/examples/lib1 is used. The second time, the entry mods3 from the library file lib.new is used. The third time, the entry mods2 from the library file /usr/examples/lib2 is used.

Example

```
* Title of a netlist
.......
set parameters
netlist of circuit
.......
.lib my_library TT
...........

.Alter
.DEL LIB my_library TT
.LIB my_library SS
```

```
.Alter
.DEL LIB my_library SS
.LIB my_library FF
.END
```

This simulates with the TT process center models. Then unloads the TT model library and uses the models from the SS slowest process corner. Then this model library would be unloaded and the simulation would be run with the FF fast process corner. In this way, a number of process corners can be run on a circuit to check performance/yield variations.

.DISTO (Distortion Analysis)

This statement performs distortion analysis of the circuit in frequency domain. A multi-dimensional Volterra series analysis is done using a multi-dimensional Taylor series to represent the nonlinearities at the operating point. Terms up to the third order are used in the series expansions.

Syntax

```
.DISTO DEC|OCT|LIN nump fstart fstop <f2overf1>
```

DEC	Sweep by decades.
OCT	Sweep by octaves.
LIN	Linear sweep.
nump	Number of points per decade or per octave, or the number of points in a linear sweep.
fstart	The starting frequency.
fstop	The final frequency.
f2overf1	An optional parameter with a value between, but not equal to, `0.0` and `1.0`.

If `f2overf1` is not specified, `.DISTO` does a harmonic analysis, i.e., it computes distortion in the circuit using only a single input frequency f_1, which is swept, as specified by the arguments `nump`, `fstart` and `fstop` (exactly as in the `.AC` statement). Inputs at this frequency may be present in more than one input source, and their magnitudes and phases are defined by the parameter `DISTOF1` of the independent voltage and current sources.

The arguments of the parameter `DISTOF2` are not relevant in this case. The analysis produces information about the AC values of all node voltages and branch currents at the harmonic frequencies $2f_1$ and $3f_1$, against the input frequency f_1 as it is swept. A complex distortion output of 1 signifies $\cos(2\pi(2f_1)t)$ at $2f_1$ and $\cos(2\pi(3f_1)t)$ at $3f_1$, using the convention that 1 at the input fundamental frequency is equivalent to $\cos(2\pi f_1 t)$.

The desired distortion component ($2f_1$ or $3f_1$) can be selected using postprocessing commands and be printed or plotted. Normally, the magnitude of the harmonic components is of primary interest, so the magnitude of the AC distortion value is examined. It should be noted that these are the AC values of the actual harmonic components and are not equal to HD2 or HD3. To obtain HD2 and HD3, divide by the corresponding AC values.

When `f2overf1` is specified, the `.DISTO` statement provides the spectral analysis. It considers the circuit with sinusoidal inputs at two different frequencies: f_1 and f_2. f_1 is swept as specified by arguments `nump`, `fstart` and `fstop`. f_2 is kept fixed at a given frequency as f_1 sweeps. The value at which f_2 is fixed is equal to `f2overf1·fstart`. Each independent source in the circuit can have two superimposed sinusoidal inputs for distortion at the frequencies f_1 and f_2.

The arguments of the parameter DISTOF1 in a source's input line (see the description of independent sources) defines the magnitude and phase of the f_1 component, and the arguments of the parameter DISTOF2, AC values, defines the magnitude and phase of the f_2 component. The analysis produces plots of all nodes voltages/branch currents at the intermodulation product frequencies f_1+f_2, f_1-f_2, and $2f_1-f_2$, against the swept frequency f_1. As in the harmonic analysis case, the results are the actual AC voltages and currents at the intermodulation frequencies, and must be normalized with respect to AC values to obtain the parameters IM.

If the parameters DISTOF1 or DISTOF2 are missing from the description of an independent source, that source is assumed to have no input at the corresponding frequency. The default values of the magnitude and phase are 1.0 and 0.0 respectively. The phase is specified in degrees.

Ideally, the number f2overf1 should be an irrational number. But since this is not feasible in practice, the denominator should be as large as possible in its fractional representation. It should be above 3 for accurate results (i.e., if f2overf1 is represented as a fraction *a/b*, where *a* and *b* are integers with no common factors, *b* should be as large as possible. Note that $a<b$ because f2overf1 is constrained to be <1).

To illustrate the reason, consider the two cases f2overf1=49/100 and f2overf1=1/2. In a spectral analysis, the outputs produced are at $f_1=f_2$, f_1-f_2, and $2f_1-f_2$. In the second (f2overf1=1/2) case, $f_1-f_2=f_2$, so the result at the f_1-f_2 component is erroneous because of the strong fundamental f_2 component at the same frequency. Also, $f_1+f_2=2f_1-f_2$ in the latter case, and each result is erroneous individually. This is not the case where f2overf1=49/100 because $f_1 - f_2 = \frac{51}{100}f_1 \neq \frac{49}{100}f_1 = f_2$. In this case, there are two very closely spaced frequency components at f_2 and f_1-f_2.

One advantage of the Volterra series technique is that it computes distortions at mixed frequencies expressed symbolically (i.e., $nf_1 \pm mf_2$). Therefore, the strengths of distortion components can be measured accurately, even if the separation between them is very small, as opposed to transient analysis, for example. The disadvantage is that if two of the mix frequencies coincide the results are not merged together and presented (though this could presumably be done as a post-processing step). It is important to keep track of the mix frequencies, and add the distortions at coinciding mix frequencies together, if necessary.

Examples

```
.DISTO DEC 10 1kHz 100Meghz
.DISTO DEC 10 1kHz 100Meghz 0.9
```

.DISTRIBUTION (User-Defined Distribution)

Syntax

```
.DISTRIBUTION name (deviation,probability)
+ <(deviation,probability)>
```

The `.DISTRIBUTION` statement is used to specify a user defined distribution for device tolerance as a piecewise function. The curve defined in this statement is a frequency function of p(x), and is used to generate a random variable with the same density. `deviation` is a value between `-1` and `1`. `probability` is a value between `0` and `1`. In a sequence of deviations, each successive point must be greater than or equal to the previous point. This statement is used only in Monte-Carlo analysis and in the `.MODIF` statement. It is not necessary to specify the curve in accordance with the normalization condition.

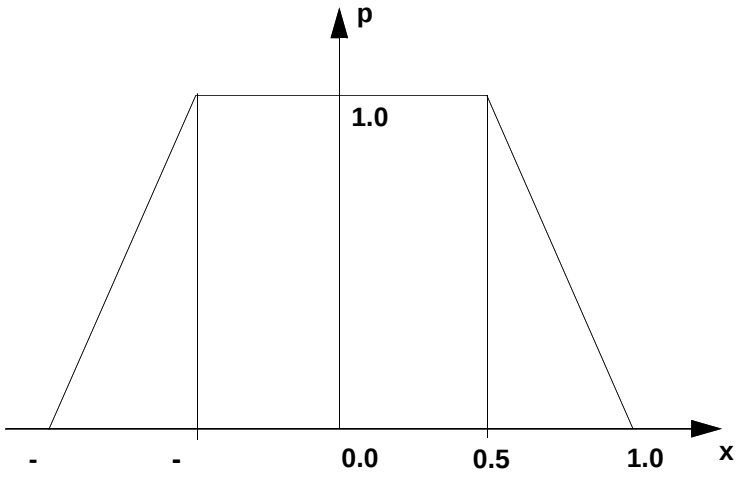


Figure 3-6 Example of Non-normalized Distribution

.DOUT (Expected State of Digital Output Signal)

The .dout statement compares simulation results (node voltage) with the expected digital state during the transient analysis and generates an output report.

Syntax

```
.dout NodeName Threshold (time state <time1 state1...>)
.dout NodeName ThresholdLow ThresholdHigh (time state <time1
state1...>)
```

NodeName	The node name;
Threshold	The single voltage threshold.
ThresholdLow	The voltage of the logic low state.
ThresholdHigh	The voltage of the logic high state.
time	An absolute time point.
state	Expected state of the node at the specified time; has to be 0 (low state) or 1 (high state), otherwise the state will be ignored.

For the first syntax, when the node voltage is above Threshold, it is a logic 1 (high state); otherwise, it is a logic 0 (low state).

For the second syntax, when the node voltage is above ThresholdHigh, it is a logic 1, when the node voltage is below ThresholdLow, it is a logic 0.

Example

```
.param vth = 3.3
.param vlo = 0.3
.param vhi = 2.5
.dout 1 vth (0 1 5n 1 7.5n 0 10n 1 12.5n 0 17.5n 1)
.dout 2 0.3 2.3 (0 0 7.5n 0 10n 0 12.5n 0 17.5n 0)
.dout x3.2 vlo vhi (2.5n 1 17.5n 0)
```

When the simulation is finished, SmartSpice generates report for all .dout statements in a netlist.

```
***** Output vector error report *****
```

If the expected and simulation states are different, SmartSpice generates following warning:

```
output signal at node [1]:
**warning** : incorrect logic state at node [1] time =
0.000000e+00, high expected!
**warning** : incorrect logic state at node [1] time = 1.000000e-
08, high expected!
**warning** : incorrect logic state at node [1] time = 1.250000e-
08, low expected!
**warning** : incorrect logic state at node [1] time = 1.750000e-
08, high expected!
output signal at node [2]:
**warning** : incorrect logic state at node [2] time = 7.500000e-
09, low expected!
```

```
**warning** : incorrect logic state at node [2] time = 1.000000e-
08, low expected!
```

If the expected and simulation states are equal, SmartSpice displays following:

```
output signal at node [x3.2]:
verified with no error.
```

When the voltage threshold parameters (VTH, VLO and VHI) are specified in the .dout statement, SmartSpice processes only VTH and ignores VLO and VHI.

At the end of simulation SmartSpice displays the total number of errors detected during processing the .dout statement:

```
Total number of .dout comparision errors : 6
```

Example

```
.dout x3.2 vth vlo vhi (2.5n 1 17.5n 0)
.dout x3.2 2.5 0.3 3.3 (2.5n 1 17.5n 0)
```

.END (End of Input Deck)

Syntax

```
.END <comment>
```

comment	Any comment, usually the name of the input deck being terminated.

The .END statement specifies the end of an input deck. When the input file contains more than one input deck, they are separated by the .END statement.

Examples

```
.END
.END DIFFPAIR CKT
```

.ENDC (End of Control Block)

Syntax

```
.ENDC
```

This statement marks the end of a `.CONTROL` section of the SmartSpice input deck. All of the statements between the `.CONTROL` and `.ENDC` statements are executed as interactively issued SmartSpice commands.

Example

```
.ENDC
```

.ENDL (End of .LIB statement)

Syntax

```
.ENDL <library_name>
```

The `.ENDL` statement specifies the end of a library definition. The optional `library_name` is useful in identifying the end of a specific library within a file containing several libraries. See Section .LIB (Library Reference).

`library_name`	Name of the library being terminated.

.ENDS (End of Subcircuit Definition)

Syntax

```
.ENDS <subcktname>
```

subcktname	Name of the subcircuit being terminated.

The .ENDS statement specifies the end of a subcircuit definition. The optional subcktname is useful for identifying the end of a specific subcircuit within a nested subcircuit definition.

Examples

```
.ENDS
.ENDS DELAY_BUFFER
```

.EOM (End of Subcircuit Definition)

Syntax

```
.EOM <subcktname>
```

subcktname	Name of the subcircuit being terminated.

The .EOM statement is identical to the .ENDS statement. It specifies the end of a subcircuit definition. The optional subcktname is useful for identifying the end of a specific subcircuit within a nested subcircuit definition.

Examples

```
.EOM
.EOM DELAY_BUFFER
```

.EQUIV (Model Name Aliasing)

Syntax

```
.EQUIV NewName1=OldName1 <NewName2=OldName2 ...>
```

NewName	Model name used in .MODEL statements.
OldName	Model name used in DEVICE statements.

Note: SmartSpice tries to search for a model with the name specified in the device statement first. If such a model was not found, SmartSpice will try to search for model alias from .EQUIV statement.

Example

```
Q1 c b e QNL
.EQUIV NewModelName=QNL
.MODEL NewModelName NPN (BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS
+ CJE=3PF CJC=2PF VA=50)
```

.FCC

The statements reports flash core cell elements.

Syntax

```
.FCC <dvth value> <file filename> <save filename> device_name_1
+ <device_name_1... >
```

dvth	Threshold voltage shift. The statement reports the flash core cell elements if:
	`device_dvth_value <= statement_dvth_value, statement_dvth_value < 0`
	
	`device_dvth_value >= statement_dvth_value, statement_dvth_value > 0`

file	Saves the output into the specified file instead of printing it on the screen
save	Saves the output file in the following format: `.HSIMPARAM HSIMDDVTO=dvth_value inst=device_name`
device_name_1...	The names of instances that will be checked for specified threshold voltage shift. The instance name can contain an asterisk (*) wildcard character.

Example

```
.FCC dvth 0.3 x2.*
```

In the example, each element in instance `x2` is reported if the element threshold voltage shift is greater than or equal to 0.3 V at the present time.

For more information about the flash memory modeling, refer to Section 4.8 Flash Memory Core Cell Modeling in the SPICE Models Manual.

.FFT (Fast Fourier Transform)

Syntax

```
.FFT <outvar> <START|FROM=val> <STOP|TO=val>
+ <NP=val> <FORMAT=NORM|UNORM>
+ <WINDOW=RECT|BART|HANN|HAMM|BLACK|HARRIS|GAUSS|KAISER|
+ PARZEN|WELCH|RIESZ|RIEMANN|POISSON|CAUCHY>
+ <ALFA=val> <FREQ=val> <FMIN=val> <FMAX=val>
```

The .FFT statement performs a Fast Fourier Transform (FFT) on the named vectors, which must be part of a Transient analysis plot.

outvar	The output variables for performing the FFT.
START	Specifies the timepoint from which the FFT should be performed. It defaults to the start of the Transient analysis.
FROM	Alias for START.
STOP	Specifies the timepoint at which the FFT should be stopped. It defaults to the end of the Transient analysis.
TO	Alias for STOP.
NP	Specifies the number of points (samples) to be used in computing the FFT. If it is not a power of two, SmartSpice sets it to the highest power of 2 which is higher than NP. Default is 1024.
FORMAT	Specifies the output format. NORM means that the amplitudes should all be normalized to the largest component in the frequency spectrum, so that the largest value in the vector is 1.0 (one). UNORM means that they are not normalized. Default is NORM. In -hspice mode, the amplitudes of frequency spectrum are normalized by the default (FORMAT is set to NORM). In regular mode, FORMAT is set to UNORM.
WINDOW	Specifies the kind of window to be used. The values listed correspond to rectangular (RECT), Bartlett (BART), Hanning (HANN), Hamming (HAMM), Blackman (BLACK), Blackman–Harris (HARRIS), Gaussian (GAUSS), Kaiser–Bessel (KAISER), Parzen (PARZEN), Welch (WELCH), Riesz (RIESZ), Riemann (RIEMANN), Poisson (POISSON) and Cauchy (CAUCHY) windows. Formulas for the windowing functions are shown in Table 3-2
ALFA	Specifies the window parameter where appropriate. Four windows accept this parameter: Gaussian, Kaiser–Bessel, Poisson, and Cauchy. Of these, the Gaussian and Kaiser–Bessel windows have a default value of 3.0, while the Poisson and Cauchy windows have a default of 4.0.
FREQ	Specifies a frequency of interest for printing in the output file. The output file will contain data at this frequency, and at its harmonics. Limited by FMAX. Default is 0.0.

FMIN	Specifies a lower limit for the frequencies listed in the output file. Frequencies lower than this value will be omitted from the output file. Default is 1/(STOP-START).
FMAX	Specifies an upper limit for the frequencies listed in the output file. Frequencies higher than this value will be omitted from the output file. Default is $\frac{NP-1}{2\times(STOP-START)}$.
fftgridsize	If this variable is set in your `SmartSpice.ini` file, SmartSpice overrides the default value of `NP (1024)`.

If no `outvars` are specified, all vectors in the current plot will transform.

All `value` parameters can be parameterized.

Examples

```
.fft v(3) np = 8192 window=kaiser alfa=4
```

This example performs an FFT on the first `8192` points of the vector `v(3)` using the Kaiser–Bessel window, with $\alpha = 4.0$.

Table 3-2 Windowing Functions in .FFT Statement

Name	Windowing Function
Rectangular	$W(i) = 1, \forall i$
Bartlett	$W(i) = \begin{cases} 2\frac{i}{N-1}, \text{ for } 0 \le i \le \frac{N-1}{2} \\ 2-2\frac{i}{N-1}, \text{ for } \frac{N-1}{2} \le i < N \end{cases}$
Hanning	$W(i) = 0.5 - 0.5\cos\left(\frac{2\pi i}{N-1}\right)$, for 0≤i<N
Hamming	$W(i) = 0.54 - 0.46\cos\left(\frac{2\pi i}{N-1}\right)$, for 0≤i<N
Blackman	$W(i) = 0.42323 - 0.49755\cos\left(\frac{2\pi i}{N-1}\right) - 0.07922\cos\left(\frac{4\pi i}{N-1}\right)$, for 0≤i<N
Blackman–Harris	$W(i) = 0.35875 - 0.48829\cos\left(\frac{2\pi i}{N-1}\right) + 0.14128\cos\left(\frac{4\pi i}{N-1}\right) - 0.01168\cos\left(\frac{6\pi i}{N-1}\right)$, for 0≤i<N
Gaussian[a]	$W(i) = \exp\left(-\frac{\alpha^2}{2}\left(\frac{i-\frac{N-1}{2}}{\frac{N-1}{2}}\right)^2\right)$, for 0≤i<N

Table 3-2 Windowing Functions in .FFT Statement

Name	Windowing Function
Kaiser–Bessel[b]	$W(i) = \dfrac{I_0\left(\pi\alpha\sqrt{1-\left(\dfrac{i-\frac{N-1}{2}}{\frac{N-1}{2}}\right)^2}\right)}{I_0(\pi\alpha)}$, for 0≤i<N
Parzen	$W(i) = \begin{cases} 1.0-6\left[\dfrac{i}{(N-1)/2}\right]^2\left[1.0-\dfrac{\lvert i\rvert}{(N-1)/2}\right], & 0\le\lvert i\rvert\le\dfrac{N-1}{4} \\ 2\left[1.0-\dfrac{\lvert i\rvert}{(N-1)/2}\right]^3, & \dfrac{N-1}{4}\le\lvert i\rvert\le\dfrac{N-1}{2} \end{cases}$
Welch	$W(i) = 1-\left(\dfrac{i-\frac{N-1}{2}}{\frac{N+1}{2}}\right)^2$, for 0≤i<N
Riesz	$W(i) = 1-\left(\dfrac{i-\frac{N-1}{2}}{\frac{N-1}{2}}\right)^2$, for 0≤i<N
Riemann	$W(i) = \sin\left[2\pi\left(i-\dfrac{N-1}{2}\right)/(N-1)\right]/\left[2\pi\left(i-\dfrac{N-1}{2}\right)/(N-1)\right]$, for 0≤i<N
Poisson[c]	$W(i) = \exp\left(-\alpha\dfrac{\lvert i-(N-1)/2\rvert}{(N-1)/2}\right)$, for 0≤i<N
Cauchy[c]	$W(i) = \dfrac{1}{1+\left(\alpha\dfrac{i-(N-1)/2}{(N-1)/2}\right)^2}$, for 0≤i<N

Note: *i* is an integer. *N* is the number of points.

[a]The default value is $\alpha = 3.0$

[b]See the Gaussian windowing function (note a above) for default value. $I_0(x)$ is the modified Bessel function of order zero. The occurrence of π in the formulas for this window is due to the custom of specifying α in units of π.

[c]The default value is $\alpha = 4.0$

Abbreviated Window Names

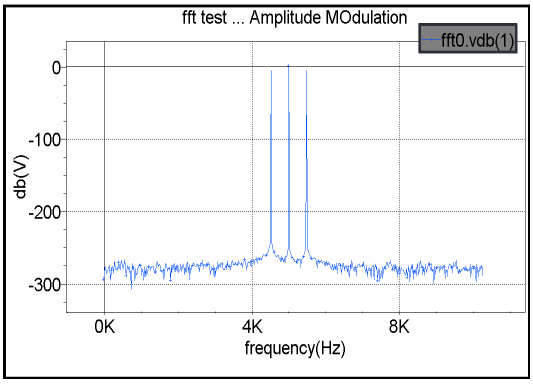
SmartSpice will look at the first four letters to determine the window. The following list shows in capitals the required minimum for each window: `RECTangular`, `BARTlett`, `HANNing`, `HAMMing`, `BLACkman-harris`, `GAUSsian`, `KAISer-bessel`, `PARZen`, `WELCh`, `RIESz`, `RIEMann`, `POISson` and `CAUChy`. As with all input decks, no distinction is made by SmartSpice between lower and upper case.

Example

```
.fft v(out) window=gaussian
.fft v(in) window=hamming
```

produce the same output as:

```
.fft v(out) window=gauss
.fft v(in) window=hamm
```



.FOUR (Fourier Analysis)

Syntax

```
.FOUR freq <num_harmonics> outvar1 <outvar2 ...>
```

freq	The fundamental frequency.
num_harmonics	The number of harmonics to be calculated.
outvar1...	The output variables or expressions for which harmonics are calculated.

See Chapter 6 Outputs, and Chapter 4 Expressions of this manual for a description of output variables and expressions.

The `.FOUR` statement performs a Fourier analysis on the results of the Transient analysis for the specified output variables. As a result, the first `num_harmonics` are calculated. If `num_harmonics` is omitted, the value of the variable `nfreqs` is used. If `nfreqs` is not set, the default value of `10` is used for the number of harmonics. The values are interpolated onto a fixed-space grid with the number of points given by the variable `fourgridsize` (default is `200`). The interpolation will be of degree `polydegree`, if that variable is set (default is `1`). If `polydegree` is `0`, no interpolation will be done. Use the `set` command to set variables `nfreqs`, `fourgridsize` and `polydegree`.

The analysis is performed only on the transient results for the last `1/freq` time period. This means the Transient analysis must be at least `1/freq` long.

The results of the `.FOUR` statement are printed immediately after the analysis is done, and no `.PRINT` or `.PLOT` cards are required.

Examples

```
.FOUR 100K V(5) V(11,12) I(VIN)
.FOUR 1MEG 20 V(2)
```

The first example performs harmonic decomposition of the transient signals `V(5)`, `V(11, 12)` and `I(VIN)` with a fundamental frequency of `100K`. The DC component and the first nine harmonics will be printed.

In the second example, Fourier decomposition is performed on the signal `V(2)` with a fundamental frequency of `1MEG`. The DC component and the first nineteen harmonics will be printed.

.FUNC (Function Definition)

Syntax

```
.FUNC <name> (<arg1<, arg2 <...>>>) <=> body
```

name	Function name.
arg1, arg2, ...	Function arguments.
<=>	Optional symbol.
body	Function body. An expression containing numbers, parameters and previously defined functions.

The `.FUNC` statement defines a function or macro which is used in expressions throughout the input deck.

Each function name must be unique. The number of arguments in a function call must be equal to that in the corresponding definition. There is no limit on the number of arguments. Even if a function has no arguments, parentheses must still be used.

The parameter `.DEFINE` can be used in place of `.FUNC` (see the command `define` in Chapter 5 Commands for additional information).

Examples

```
.FUNC f2(x,y) x*x+y*y
.FUNC dist(x,y) sqrt(f2(x,y))
.FUNC fun() 1+par1-par2
.FUNC indp(z,w) mag(z)*mag(Z)/(w*im(z))
```

.GLOBAL (Global Node Definition)

Syntax

```
.GLOBAL node1 <node2 ...>
```

The .GLOBAL statement defines nodes node1, node2,... to be globally accessible nodes. These nodes are used in subcircuits without being declared in the subcircuit definition. Any internal subcircuit nodes that have the same name as a global node will be assumed to refer to the global node.

Examples

```
VCC VCC 0 4V
.GLOBAL VCC
```

In this example, the .GLOBAL statement defines node VCC as a global node, connecting all subcircuits with the internal node name VCC to the same power supply VCC.

.GRAPH (Plot Results to Hardcopy Device)

Syntax

```
.GRAPH <anytype> <MODEL=mname> outvar1 <outvar2 ...> <(plo,phi)>
```

anytype	The type of analysis for which the specified outputs are desired. Must be one of the following parameters: • **AC:** For AC analysis outputs. • **DC:** For DC sweep outputs. • **DISTO:** For distortion analysis outputs. • **MEAS:** For saved results of the .MEASURE statement. • **NET:** For network analysis outputs. • **NOISES:** For noise spectral density outputs. • **NOISET:** For integrated noise outputs. • **OP:** For operating point analysis outputs. • **PZ:** For pole-zero analysis outputs. • **TF:** For transfer function outputs. • **TRAN:** For Transient analysis outputs.
mname	The plot model name.
outvar1...	Output variables or expressions to be printed. See Chapter 6 Outputs, and Chapter 4 Expressions for a description of output variables and expressions.
plo, phi	Lower and upper plot limits. When used, they must be at the end of the .GRAPH statement.

The .GRAPH statement is used to print simulation results on a hardcopy device. It first creates hardcopy files using the command hardcopy described in Chapter 6 Outputs (the file is stored in the directory /tmp), then sends this file to the printer.

The input deck contains any number of .GRAPH statements. Each .GRAPH statement contains any number of output variables.

Examples

```
.GRAPH AC V(v1,vcc) VP(6,8)
.GRAPH DC V(5) I(vout) ie(q7) (0,10)
.GRAPH TRAN v(1,2) i(cap1)
```

Model Statement for .GRAPH

The .GRAPH statement allows you to specify XGRID, YGRID, XMIN/XMAX, XSCAL/YSCAL and YMIN/YMAX.

Syntax

```
.MODEL modelname PLOT (name1=val   name2=val2 ...)
```

modelname	The plot model name showed up in .GRAPH statement.

PLOT	The keyword for .GRAPH statement model.
name1=val1...	Model parameters. If parameters have not been set up, SmartSpice takes the default values of the model parameters described in Table 3-3.

Table 3-3 Model Parameters

Parameter Name	Default Value	Description
XGRID, YGRID	0.0	Turns on the axis grid lines when setting to 1.0
XMIN, XMAX	0.0	If XMIN is not equal to XMAX, then XMIN and XMAX determines the x-axis plot limits. If XMIN=XMAX, or if XMIN and XMAX are not set, then the limits are automatically set. These limits apply to the actual x-axis variable value regardless of the XSCAL type.
XSCAL	1.0	Scale for the x-axis. Linear (LIN) XSCAL=1. Logarithm (LOG) YSCAL=2.
YMIN, YMAX	0.0	If YMIN is not equal to YMAX, then YMIN and YMAX determines the y-axis plot limits. If YMIN=YMAX, or if YMIN and YMAX are not set, then the limits are automatically set. These limits apply to the actual y-axis variable value regardless of the YSCAL type.
YSCAL	1.0	Scale for the y-axis. Linear (LIN) YSCAL=1. Logarithm (LOG) YSCAL=2.
FREQ†	0.0	Plots symbol frequency. Value 0 suppresses plot symbol generation; a value of n generates a plot symbol every n points
MONO†	0.0	Monotonic option MONO=1 automatically resets x-axis if any change in x-direction
TIC†	0.0	Show tick marks

Note: Parameters marked with the symbol † are not available in this release, but will be added in the next release.

Example

```
.graph ac   vm(*) vp(*p*) (-40, 40)
.graph ac MODEL=df vm(opout) (-500, 500)
.graph ac MODEL=HARDCOPY  vm(*)  (-500, 500)
.MODEL HARDCOPY PLOT (FREQ=1 TIC=1 MONO=1
+ YSCAL=1 XSCAL=2 XGRID=0 YMIN=-300 YMAX=300)
.graph ac MODEL=HARDCOPY1  vr(*) vi(*p*) (-300, 300)
.MODEL HARDCOPY1 PLOT (FREQ=1 TIC=1 MONO=1
+ YSCAL=1 XSCAL=2 XGRID=1 YMIN=-140 YMAX=140)
```

.IC (Initial Conditions)

Syntax

```
.IC V(node1)=val1 <V(node2)=val2 ...>
```

The .IC statement sets transient initial conditions, or operating point conditions for .OP and small-signal analysis at specified nodes. The statement has two different interpretations depending on whether the parameter UIC is specified in the analysis statement:

- When the analysis statement (.TRAN, .AC or .NOISE) contains the parameter UIC, analysis starts from these initial conditions, immediately bypassing the DC operating point calculation
- When the parameter UIC is not specified, SmartSpice uses the initial conditions during the operating point calculation. This means that the nodes specified in the .IC statement will have the specified voltage values when the operating point is found.

Analysis ignores the .IC statement if the analysis statement (.TRAN, .OP, .AC or .NOISE) contains the parameter CALLV.

The .IC statement should not be confused with the .NODESET statement. The .NODESET statement only helps DC convergence and does not affect the final bias solution (except in multi-stable circuits).

The .IC statement is local to subcircuits. This means that if an .IC statement is specified in a subcircuit definition, SmartSpice expands the node names that appear in the statement during subcircuit expansion. If two .IC statements are defined on the same hierarchy level refer to the same node, the first statement is overwritten by the second statement. Local .IC statements defined inside subcircuits have higher priority than top level .IC statements for the same node. .IC statements referring to the same node will generate a warning of “duplication”.

The .IC statement includes support for templates and bus notation.

Examples

```
.IC V(11)=5 V(4)=-5 V(2)=2.2
.IC V(10)=0.1 V(out1)=1.0
.SUBCKT SUB1 1 3
RSUB1 1 2 100
RSUB2 2 3 200
.IC V(2)=2.5
.ENDS
VIN 2 0 PULSE(0 5 1N 1N 2N 10N)
R1 2 10 1K
XSUB 10 30 SUB1
R2 30 0 10K
.IC V(XSUB.2)=3.5
```

In the example above, .IC statement defined inside subcircuit SUB1 will have higher priority than another one specified at top level. The initial condition of node XSUB.2 will be set to 2.5 volts.

```
.global vss
.subckt nand ina inb out
m1 out in1 vss vss mosp w=10u l=1.3u
m2 out inb vss vss mosp w=10u l=1.3u
m3 out ina 10 gnd mosn w=5u l=1.3u
m4 10 inb 0 gnd mosn w=5u l=1.3u
```

```
.ends nand
x1 a b c nand
.ic v(x1.ina)=5
.ic v(a)=5
```

In the example above, the first `.IC` statement uses the formal nodename, while the second uses the expanded nodename. The two `.IC` statements are equivalent.

```
.ic v(bus<1:2>)=0.3
.ic v(node*)=0.5
```

In the example above, the first `.IC` statement sets the 2 bus wires (`bus1`, `bus2`) to `0.3`V. The second `.IC` statement sets all nodes with the prefix `node` to `0.5`V.

Voltage Source Equivalent Parallel Conductance Calculation

The transient and small-signal analyses need a solution at operating point. The initial guess is usually calculated by performing a DC operating point analysis using the DC equivalent model of the circuit (unless the parameter `UIC` is specified). The `.IC` statement allows you to specify fixed voltage values at selected circuit nodes during operating point analysis, and to obtain the initial solution with the specified voltage values. To perform this operation, a voltage source equivalent (a current source with parallel conductance G_{ic}) must be connected to each initialized node during the operating point analysis. After finishing this initial guess calculation, the equivalent voltage sources are disconnected.

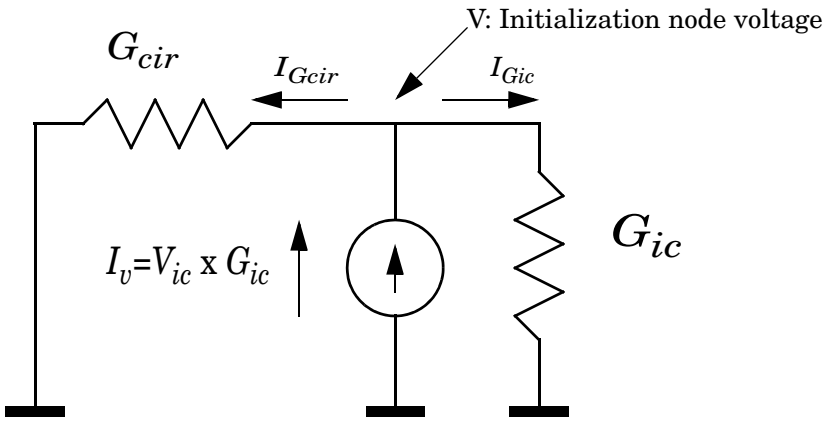


Figure 3-7 Voltage Source Equivalent

The current source value is calculated as $I_v = V_{ic} \, x \, G_{ic}$, where V_{ic} is the voltage specified by the `.IC` statement, and G_{ic} is the conductance. From current equation at initialization node:

$$I_v = I_{Gcir} \div I_{Gic}, \text{ or } V_{ic} \times G_{ic} = V \times G_{cir} \div V \times G_{ic},$$

the initialization voltage will be equal:

$$V = V_{ic} \times \frac{G_{ic}}{G_{ic} + G_{cir}} = V_{ic} \times \frac{1}{1 \div \frac{G_{cir}}{G_{ic}}}$$

Such an approach works well, when $G_{ic} >> G_{cir}$ (G_{cir} is the equivalent conductance for all branches from initialization node to ground). When $G_{ic} >> G_{cir}$, the result is $V = V_{ic}$. However, if $G_{cir}=G_{ic}$, the result is $V = V_{ic}/2$, which is not ideal. you must understand that when the voltage is specified at the node using the `.IC` statement, the resulting voltage at this

node after performing a DC operating point analysis cannot be exactly equal to a specified value because it is calculated as shown above and is dependent on the relation between G_{ic}, and G_{cir}. To smooth this dependence, SmartSpice calculates G_{ic} as:

$$G_{ic} = max(ICG, 10000 \times G_{cir})$$

where *ICG* is the `.OPTIONS` statement parameter.

.IF-.ELSE (Condition Control Statement)

Syntax

```
.IF(condition A)
<statements>
<.ELSEIF(condition B)
<statements>>
<.ELSE
<statements>>
.ENDIF
```

The condition controlled statement .IF-.ELSE can be used to change the circuit topology or select device statements according to a specific condition. The .ELSEIF and .ELSE statements are optional. Each .IF statement in a netlist must be matched by a closing .ENDIF statement. Any number of .ELSEIF statements can appear between the .IF and .ENDIF statements, but only one .ELSE is allowed.

The .IF, .ELSEIF, .ELSE and .ENDIF are nested statements, and can appear inside other .IF statements. Each nested .ELSEIF, .ELSE or .ENDIF statement belongs to the closest preceding .IF statement.

In a .IF or .ELSEIF condition statement expressions, functions, predefined macros, parameters, constants and operators can be used. For more details see Chapter 4 Expressions.

Example

```
X1   in  out1  cell a=1
X2   in  out2  cell a=2
.subckt  cell  in  out
.if (a==1)
MN1 out in PGND PGND nmos w=1.6 l=0.36
MP1 out in PVDD PVDD pmos w=4 l=0.36
.elseif (a==2)
MN1 out in PGND PGND nmos w=1.6 l=0.36
MP1 out in PVDD PVDD pmos w=3 l=0.36
.else
MN1 out in PGND PGND nmos w=1.6 l=0.36
MP1 out in PVDD PVDD pmos w=2 l=0.36
.endif
.ends
```

Example

```
.subckt test  in out   a=0 b=0
.if (a==0)
r1 in   1  1000
.if (b==1)
r5 1 2   1000
.elseif(b==2)
r5  1 2  2000
.endif
r6 2 3 1000
.else
r1 in  3   5000
.endif
r3 3 out 1000
.ends
```

Example

```
.param t=2
.if (t==0)
.temp 25
.tran 1n 10n
.elseif (t==1)
.temp 30
.tran 1n 11n
.else
.temp 35
.tran 1n 12n
.endif
```

Example

```
.if (b==1)
X1 in out1 cell1 a=1
X2 in out2 cell1 a=1
.subckt cell1 in out a=0
.param c='a'
MN1 out in PGND PGND nmos w=1.6 l='0.36+c/100'
MP1 out in PVDD PVDD pmos w=4 l=0.36
.model nmos nmos ...
.model pmos pmos ...
.save v(out)
.ends
.elseif (b==2)
X1 in out1 cell2 a=2
X2 in out2 cell2 a=2
.subckt cell2 in out
.param c='a'
MN1 out in PGND PGND nmos w=1.6 l='0.36+c/100'
MP1 out in PVDD PVDD pmos w=3 l=0.36
.save v(out)
.model nmos nmos ...
.model pmos pmos ...
.ends
.endif
```

In the first example, SmartSpice selects device statement according to parameter value.

The second example is the nested .IF-.ELSE statement usage.

The third example includes anaysis and statements.

The fourth example includes parameters and model statements inside the block.

.INCLUDE (Include File)

Syntax

```
.INCLUDE filename
```

This statement causes the named file to be inserted into the input deck at the point where the statement appears. Included files can be nested arbitrarily. The `filename` is placed in single or double quotes (`'filename'` or `"filename"`).

Examples

```
.INCLUDE 'MODELS.DEF'
.INC "DEVICE1.INC"
```

The usage of environment variables is allowed for above statements.

Example

```
.INCLUDE $MY_VAR/ex.in
```

where `MY_VAR` is name of environment variable. If it's not set at the user's machine, an error message will be issued.

.INFO

Syntax

```
.INFO what=ARG1 where=ARG2 name=ARG3
```

what	a keyword
ARG1	The parameter that defines the type of information will be saved. Can be one of the following: • `inst`: Saves input parameters for instances of all components. • `models`: Saves input parameters for models of all components. • `output`: Saves effective and temperature-dependent parameter values. • `parameters`: Saves top-level circuit parameters and their values. • `primitives`: Saves model, oppoint, output, instance, region parameters and terminal names of a primitive without their actual values. • `subckts`: Saves subcircuit parameters and terminal names.
where	a keyword
ARG2	The parameter that defines where the information will be saved. Can be one of the following: • `rawfile`: Saves the data to PSF rawfile. • `nowhere`: Does not save the data anywhere.
name	a keyword
ARG3	The parameter that defines the name of the output file. Additional extensions will be added to the file name automatically.

Example

```
.INFO what=subckts where=rawfile name="subckt_info"
```

In this example SmartSpice saves subcircuit parameters and terminal names in file `subckt_info.info.subckts`.

Note: The `.INFO` statement is only supported in SPECTRE mode and requires PSF file format enabled, otherwise the following warning messages will appear:

```
.INFO statement is supported in SPECTRE mode only
.INFO statement requires enabled PSF file format
```

.IPLOT (Interactive Plotting)

Syntax

```
.IPLOT <anytype> <outvar1 <outvar2 ...>>
```

This statement identifies vectors that should be interactively plotted during simulation. If this statement is used without arguments in the interactive mode, SmartSpice pops up a list of vectors that are to be interactively plotted.

Note: This feature is unavailable if a simulation is performed in silent batch(`-sb`) mode.

Examples

```
.IPLOT v(1) v(2)
.IPLOT
```

In the first example, only the specified vectors `v(1)` and `v(2)` are interactively plotted.

In the second example, all the vectors that can be interactively plotted are listed in interactive mode. It is ignored in batch mode.

.IPRINT (Interactive Printing)

Syntax

```
.IPRINT <anytype> <outvar1 <outvar2 ...>>
```

This statement identifies vectors that should be interactively printed during simulation. If this statement is used without arguments in the interactive mode, SmartSpice pops up a list of vectors that are to be interactively printed.

Examples

```
.IPRINT v(1) v(2)
.IPRINT
```

In the first example, only the specified vectors `v(1)` and `v(2)` are interactively printed.

In the second example, all the vectors that can be interactively plotted are listed in interactive mode. It is ignored in batch mode.

.LET (Vector Creation)

Syntax

```
.LET <anytype> lhs1=rhs1 ...
```

anytype	The type of analysis for which the vector(s) are created. Must be one of the following: • **AC:** For AC analysis outputs. • **DC:** For DC sweep outputs. • **DISTO:** For distortion analysis outputs. • **FFT:** For fast Fourier transform outputs. • **FOUR:** For Fourier analysis outputs. • **NET:** For network analysis outputs. • **NOISES:** For noise spectral density outputs. • **NOISET:** For integrated noise outputs. • **OP:** For operating point analysis outputs. • **PZ:** For pole-zero analysis outputs. • **TF:** For transfer function outputs. • **TRAN:** For Transient analysis outputs. If `anytype` is omitted, the vector(s) are created for all analyses specified in the input deck.
lhs1	The name of the new vector.
rhs1	A standard output variable or expression. See Chapter 4 Expressions of this manual for the description of output variables and expressions.

The `.LET` statement is used to define new output variables as functions of other output variables. The new variables are computed during the corresponding analysis. These variables can be printed, plotted, interactively printed and plotted, measured, and saved in a rawfile. It is possible to define variables as functions of other variables previously defined in `.LET` statement with the same analysis type.

It is possible to use standard and user-defined functions and macros to define new variables. However, since these variables are computed during simulation (not after simulation) when certain functions are used, the resulting vectors are slightly different from those which could be created in the post-processor, using the same functions after simulation is finished. For example, when used in post-processor, the function `LENGTH(X)` produces the vector of the length `1` which contains the length of `X`. However, when a variable is defined in the `.LET` statement using this function, for example:

```
.LET Y=LENGTH(X)
```

then `Y` will be the same length as `X`, and will contain the numbers `1, 2, ..., LENGTH(X)`.

Examples

```
.LET AC DB_OUT = VDB(OUT)
.LET TRAN V2_SQ='V(2)*V(2)' V3_SQ='V(3)*V(3)'
```

```
+ SUM_SQ='V2_SQ+V3_SQ'
.FUNC FUN1(X,Y) COS(X)*SIN(Y)
.LET UNIT=FUN1(0,3.14152/2)
.LET ZERO=LN(UNIT)
```

.LIB (Library Reference)

Syntax

```
.LIB filename entryname (syntax_form=0)
```

or

```
.LIB filename (syntax_form=1)
```

filename	Name of the library file. For syntax_form=0, the filename may be enclosed in single or double quotes, or quotes may be omitted.
entryname	The entry name of the section of the library file to be included in the input deck and can be included in single quotes (in the case of syntax_form=0).

This statement is used to include a designated section of the library file in the input deck. Library calls can be nested arbitrarily.

Examples

```
.LIB ’/usr/examples/lib1’ typical
.LIB "/usr/examples/lib1” typical
.LIB /usr/examples/lib1 typical
.LIB /usr/examples/lib1 ‘TT’
.LIB smspice.lib
```

Library File Structure

When syntax_form is set to 0, each section of the library file starts with .LIB followed by entryname, and ends with .ENDL followed by an optional entry name:

```
.LIB entryname
. . .
.ENDL <entryname>
```

Each section can contain any valid input statement.

Binning Consistency Between Parameters of the Model and Instance

SmartSpice generates a warning for the following cases:

1. The model, specified in device line, exists with the exact name (which will be used) as the binning alias.

Example

```
MWithoutBinning vdd in gnd vss nch   W=0.7U L=0.7U
.model nch nmos level=49 version=3.1 vth0=1.1 lmin=0.6um lmax=0.8um
wmin=0.6um wmax=0.8um
.model   nch.1   nmos   level=49   version=3.1   vth0=1.1   lmin=0.6um
lmax=0.8um wmin=0.6um wmax=0.8um
```

2. When the binning model is specified explicitly in the device line, but the model and device parameters are inconsistent.

Example

```
QExplicitBinning vdd in gnd qnl.1 area=300
.model qnl.1 npn areamin=100 areamax=200
```

Examples

The following is an example of a library file for use with syntax_form, which contains three entries: nor_1, trans_mods and mods2.

```
.lib nor_1
.subckt nor in1 in2 out n1=2 nw=3 pl=2 pw=18
mxu4 out in2 u4s vdd pmos w=pw l=pl
mxu6 u4s in1 vdd vdd pmos w=pw l=pl
mxu3 out in1 gnd gnd nmos w=nw l=nl
mxu5 out in2 gnd nmos w=nw l=n1
.ends nor
.endl nor_1
*
.lib tran_mods
.model ntran nmos level=2 vto=.7 gamma=.2 kp=3e-05
+ lambda=0.02 tox=6e-07
.model ptran pmos level=2 vto=.7 gamma=.4 kp=1.5e-05
+ lambda=0.03 tox=6e-07
.endl
*
.lib mods2
.model pmos pmos level=2 vto=.7 gamma=.4 kp=1.5e-05
+ lambda=0.03 tox=6e-07
.model nmos nmos level=2 vto=.7 gamma=.2 kp=3e-05
+ lambda=0.02 tox=6e-07
.endl
```

When syntax_form is set to 1, the library file can contain model or subcircuit definitions. The following example is a library file that contains an opamp subcircuit and a NMOS model.

```
* Syntax_form=1 Library Example
.SUBCKT opamp 1 2 3 4 PARAMS: gain=100Meg
.model nmos nmos vto=.7 gamma=.2 kp=3e-05
.ENDS opamp
.model nmod nmos level=49 TOX=1.0E-8 VTH0=0.62
+ DVT0=0.176 DVT1=0.52 DVT2=-.05 ...
```

.LIFETIME (Degradation Simulation Statement)

Syntax

```
.LIFETIME deglevel
```

deglevel	is a number.

Note: This Statement is prepared for future use.

.LIN (Linear Network Analysis)

Syntax

```
.LIN <SPARCALC=1|0> <MODELNAME=model_name>
+ <FILENAME=file_name> <FORMAT=touchstone | selem>
+ <DATAFORMAT=ri|ma|db> <FREQDIGIT=value> <SPARDIGIT=value>
+ <GDCALC=0|1>
```

Parameter	Description
SPARCALC=1 | 0	Calculate S-parameters. Default is = 1.
MODELNAME=model_name	Set a model name for extracted .LIN parameters.
FILENAME=file_name	Set a file name to write S-parameters.
FORMAT=TOUCHSTONE | SELEM	Set format file .LIN. Currently supports TOUCHSTONE version 1.0 and SELEM file formats.
DATAFORMAT=RI | MA | DB	Sets data format for: RI (real imaginary), MA (magnitude, phase) and DB (decibels)
FREQDIGIT=value	Number of digits in a file for frequency.
SPARDIGIT= value	Number of digits in a file for S-parameters.
GDCALC=0 | 1	Calculate Group Delay for S-parameters. Default is =0.

.LIN analysis performs the extraction of scattering (S), impedance (Z) and admittance (Y) parameters for an N terminal network. Analysis calculates impedance characteristics (VSWR, ZIN, YIN), stability factors (K_STABILITY_FACTOR and MU_STABILITY_FACTOR) and gain measurements (G_MAX, G_MSG, G_TUMAX and G_U). The analysis also supports Group Delay calculation for S-parameters (GDCALC=1, zero by default) and writing S-elem file, parameter information (FORMAT=selem).

.LIN analysis performs the extraction of the following output parameters for N-terminal networks:

- Scattering (S), impedance (Z), admittance (Y) and hybrid (H) parameters;
- Impedance characteristics: VSWR(i) (Voltage standing wave ratio), ZIN(i) (Input Impedance) and YIN(i) (Input admittance);
- Stability factors: K_STABILITY_FACTOR and MU_STABILITY_FACTOR;
- Gain factors: G_MAX (Maximum Available Power Gain), G_MSG (Maximum Stable Gain), G_TUMAX (Maximum Unilateral Transducer Gain) and G_U (Unilateral power gain).

Post-processor syntax for S, Y, Z and H output parameters:

```
.PROBE P(m,n)(data_type), Pmn(data_type)
.PRINT P(m,n)(data_type), Pmn(data_type)
```

where:

- m and n are port indices;
- P is one of the parameters X, Y, Z or H;

- type is one of the following types: R-real, I-imaginary, M-magnitude, P or PD-phase in degrees, PR-phase in radians and DB-decibels. If type is omitted, by default it is M-magnitude.

Example

```
.PROBE S(1,2)(M), S11(R), Y(1,2)(M), Y11(R)
```

Post-processor syntax for VSWR, ZIN and YIN output parameters:

```
.PROBE P(n)
.PRINT P(n)
```

where:

- n is port index;
- P is one of the parameters VSWR, ZIN or YIN;

Example

```
.PROBE VSWR(1), ZIN(2), YIN(2)
```

Post-processor syntax for group delay:

```
S(m,n)(T), S(m,n)(TD), Smn(T) or Smn(TD),
```

where m and n are port indices.

Example

```
.PRINT S(1,2)(T), S(1,2)(TD), S11(TD)
```

SmartSpice batch mode

To run a deck in batch mode run SmartSpice with the following parameters:

```
>smartspice –b Nport_1_LIN.sp –r Nport_1_LIN.raw
```

A RAW-file can be viewed by SmartView.

Example deck Nport_1_LIN.sp

```
* BAND PASS FILTER
.options post=2 dcstep=1 nomod
C1   IN 2  3.166pF
L1   2  3  203nH
C2   3  0  3.76pF
C3   3  4  1.75pF
C4   4  0  9.1pF
L2   4  0  36.81nH
C5   4  5  1.07pF
C6   5  0  3.13pF
L3   5  6  233.17nH
C7   6  7  5.92pF
C8   7  0  4.51pF
C9   7  8  1.568pF
C10  8  0  8.866pF
L4   8  0  35.71nH
C11  8  9  2.06pF
C12  9  0  4.3pF
L5   9 10  200.97nH
C13  10 OUT  2.97pF
RX  OUT  0  1.e+14
VIN  IN 0  AC 1
```

```
******** LIN analysis
.AC LIN 101 200meg 300meg
.LIN sparcalc=1 modelname=my_custom_model
+ filename=Nport_1_LIN_RI_DIFF.s2p format=touchstone
+ dataformat=ri
Pin  IN  4 port=1 z0=50
Pout OUT 8 port=2 z0=75

******* LIN analysis post-processing
.PROBE AC S(1,1) S(1,2) S(2,1) S(2,2)
.PROBE AC K_STABILITY_FACTOR MU_STABILITY_FACTOR
.PROBE AC G_MAX G_MSG G_TUMAX
.PROBE AC VSWR(1) VSWR(2) ZIN(1) ZIN(2) YIN(1) YIN(2)

******* LIN analysis post-processing
.MEASURE AC ms11_1 when s11(m)=0.4 cross=1
.MEASURE AC my11_1 when y11(m)=1 cross=1
.MEASURE AC K_maximum MAX K_STABILITY_FACTOR
.MEASURE AC K_minimum MIN K_STABILITY_FACTOR

.end
```

In this example nodes 4 and 8 are reference nodes.

Post-processor syntax for group delay is: *S(m,n)(T)*, *S(m,n)(TD)*, *Smn(T)* or *Smn(TD)*, where *m* and *n* are port indices.

Example

```
.AC LIN 101 200MEG 300MEG
.LIN modelname=ex_lin_3_2ports filename=ex_lin_3_2ports
+ format=selem dataformat=ri gdcalc=1
* Post-processor syntax for group delay vec
.PROBE AC S(1,1)(T) S(1,1)(TD) S(1,2)(T) S(1,2)(TD)
.PROBE AC S(2,1)(T) S(2,1)(TD) S(2,2)(T) S(2,2)(TD)
.PROBE AC S11(T)  S12(T)  S21(T)  S22(T)
.PROBE AC S11(TD) S12(TD) S21(TD) S22(TD)
```

This example shows how to calculate and save group delay values for S-parameters.

Example

```
*| NPort=2 DATA=13 NOISE=0 GROUPDELAY=1 COMPLEX_DATAFORMAT=RI
*| NumOfBlock=1 NumOfParam=0
*|
.MODEL ex_lin_3_2ports S
+ N=2 FQMODEL=SFQMODEL TYPE=S Z0=            50            50
.MODEL   SFQMODEL   SP   N=2   SPACING=POI   INTERPOLATION=LINEAR
MATRIX=NONSYMMETRIC
+ DATA=13
+ 2.000000e+007
+             1  4.61838e-012  6.66667e-011 -3.53678e-014
+  -3.75807e-007  1.99372e-010      0.333333   0.000353678
+ 3.556559e+007
+             1  2.52293e-012  6.66667e-011 -1.98888e-014
+  -3.75807e-007  1.12115e-010      0.333333   0.000198888
+ 6.324555e+007
...
```

```
.MODEL ex_lin_3_2ports_GD SP N=2 SPACING=POI INTERPOLATION=LINEAR
+ MATRIX=NONSYMMETRIC VALTYPE=REAL
+ DATA=13
+ 2.000000e+007
+   2.14255e-020            0 -2.37404e-012            0
+  -2.37404e-012            0  4.74809e-012            0
+ 3.556559e+007
+   2.14255e-020            0 -2.37404e-012            0
+  -2.37404e-012            0  4.74809e-012            0
...
```

This example is a sample of the generated Selem-file.

.LOADBIAS (Load Bias File)

Syntax

```
.LOADBIAS | .LOAD filename
```

This statement loads the specified bias point file. The file is usually produced by the .SAVEBIAS statement and is a text file that contains comment lines and a .NODESET statement with the bias-point node voltages.

Examples

```
.LOADBIAS "SAVED.BIAS"
.LOAD /main/examples/sbias1.in
```

.LOADBIAS behavior together with .ALTER statement

If the .ALTER statement is specified in the netlist, .LOADBIAS will try to load the specified bias point file for every altered circuit with the name filename-alternumber. If the file is not available, SmartSpice will load the original file, specified in the .LOADBIAS statement.

.LSTB Analysis

Syntax

```
.LSTB DEC|OCT|LIN nump fstart fstop mode=[ single | diff | comm ]
+ vsource=[vlstb | vlstbp,vlstbn] method=middlebrook | tian
+ <sw2_spcf>

Vxyz driving_node feedback_node 0

.PRINT AC LSTB | LSTB(DB) | LSTB(M) | LSTB(P) | LSTB(R) | LSTB(I)
.PROBE AC LSTB | LSTB(DB) | LSTB(M) | LSTB(P) | LSTB(R) | LSTB(I)
```

DEC	Sweep by decades.
OCT	Sweep by octaves.
LIN	Linear sweep.
nump	Number of points per decade or per octave, or number of linear sweep points.
freqlist	Indicates the list of frequencies to be performed by AC analysis.
fstart	The starting frequency.
fstop	The final frequency.
Vxyz	Indicate the insertion point of test circuit. Direction of Vsource matters.
driving_node	driving node
feedback_node	feedback node
mode	• `single`: single-ended (default). • `diff`: differential mode • `comm`: common mode test.
vsource	• `vlstb`: The only one vsource for single-ended mode • `vlstbp`: One of the two vsources for differential/common mode • `vlstbn`: The other one of the two vsources for differential/common mode • `method`: Used for single ended mode
LSTB	Output all results, all types
LSTB(x)	• `x=DB`: Output the dB values of loop gain. • `X=M`: Output magnitude • `X=P`: Output phase • `X=R`: Output real part • `X=I`: Output imaginary part

sw2_spcf	The second (nested) sweep specification. It is specified in one of the following ways: • SWEEP swname swstart swstop swincr • SWEEP swname LIST nplist swval1 swval2 ... • SWEEP swname swtype np swstart swstop • SWEEP MODIF = dataname <PRTBL>, • <SWEEP> DATA=dataname. where:
swname	The name of the parameter to be swept. It can be: • a parameter label defined in a global .PARAM statement. • a name of an independent voltage or current source. • a full-path name of a device parameter. • a full-path name of a model parameter. • the parameter TEMP (temperature sweep).
swstart	The starting value of the swept parameter.
swstop	The stop value of the swept parameter.
swstep	The increment value of the swept parameter.
swincr	The increment value for a linear sweep.
nplist	Number of parameter values swval1, swval2,... specified for a sweep of the type LIST.
swtype	The keyword that defines the type of sweep: DEC, OCT or LIN.

Example

```
***** MULTI_SWEEP_AC ******

vin 1 0 dc 1 ac acmag= 'mag*1'  acphase='ph-2'
vin2 3 0 dc 1 ac 0
r1  1 2 parr
r2  2 0 100
c1  2 0 'capac/10'
** ****************** Label Definition
.PARAM parr=100 pstart=100 incr=pstart mag=1 ph=2  capac=10n

** ****************** STB + SWEEP

.lstb dec 10 10K 100meg  vsource=vin
+     SWEEP  parr 100 200 100  sweep capac 10n 20n 10n

.lstb dec 10 10K 100meg  vsource =  vin , vin2
+     SWEEP  parr 100 200 100  sweep capac 10n 20n 10n

.lstb dec 10 10K 100meg  vsource= vin,vin2
+     SWEEP  parr 100 200 100  sweep capac 10n 20n 10n
```

```
.lstb dec 10 10K 100meg  vsource=vin, vin2
+     SWEEP  parr 100 200 100  sweep capac 10n 20n 10n


.lstb dec 10 10K 100meg  vsource=vin ,vin2
+     SWEEP  parr 100 200 100  sweep capac 10n 20n 10n

.lstb dec 10 10K 100meg  vsource = vin2
+     SWEEP  parr 100 200 100  sweep capac 10n 20n 10n

********************* MODIF
*.MODIF capac=100n loop=2 r2(res)+=(pstart)incr prtbl

.options nomod  post=1
.end
```

SmartSpice stability analysis simulates loop gain, phase and gain margins without breaking the feedback loop. The user is required to locate a probe (`vsource` key word in `.LSTB`) at which measurement macros will be inserted in the circuit. Two types of macro can be selected: `vprobe` for single ended mode and composite `cmdmprobe` for differential and common mode. The macros are placed in feedback loop to be measured.

SmartSpice will always save the loop gain to the `*.cx#` file. The `*.cx#` file is a general HSPICE file format for the complex outputs.

MODE=single(single ended)

The problem when trying to simulate loop gain is that in opening up the loop to make the proper measurements, the DC bias point of the circuit will be altered. Since the circuit is linearized around the DC bias point in AC analysis, this will throw off the results of the entire simulation. One technique can make these measurements without opening up the loop. This technique is taken from an article by Dr. R. D. Middlebrook which appears in the International Journal of Electronics, volume 38, number 4, 1975.

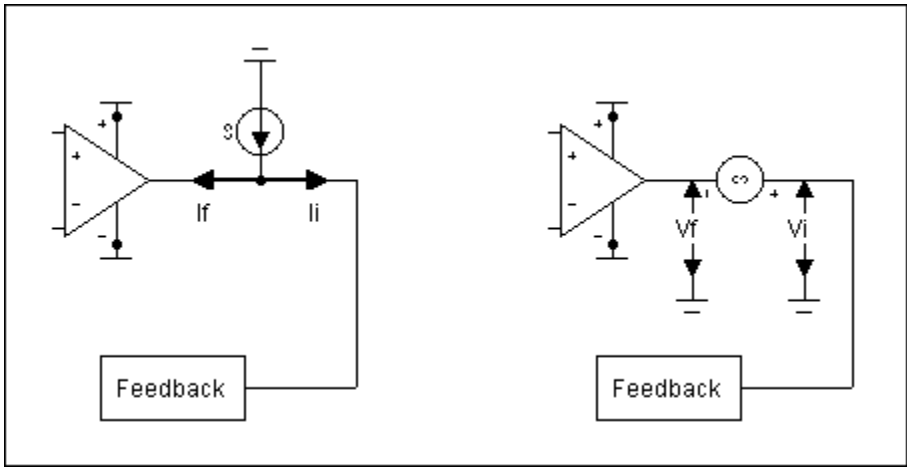
Voltage and current gains are defined as:

```
Gv = vf / vi
Gi = if / ii
```

It is not necessary to break the loop in order to measure these gains. The loop may be opened in its feedback path and the appropriate test signal injected. Injecting a current into the signal path will split the current into its feedback and input currents. This ratio can then be measured to produce a current loop gain. The voltage gain may also be measured through the same technique by placing a voltage source in the loop. The gain measurement setups are displayed in the figure below.

For the actual measurements, macros will be created for the voltage and current injections in order to measure the voltages and currents with user functions. Both the voltage loop gain and the current loop gain must be taken into account to measure the total loop gain. They are related through Equation :

$$G = (Gi * Gv - 1)/(Gi + Gv + 2)$$



Measurements of phase and gain margins for single mode are fully automated. SmartSpice prints their values after simulation

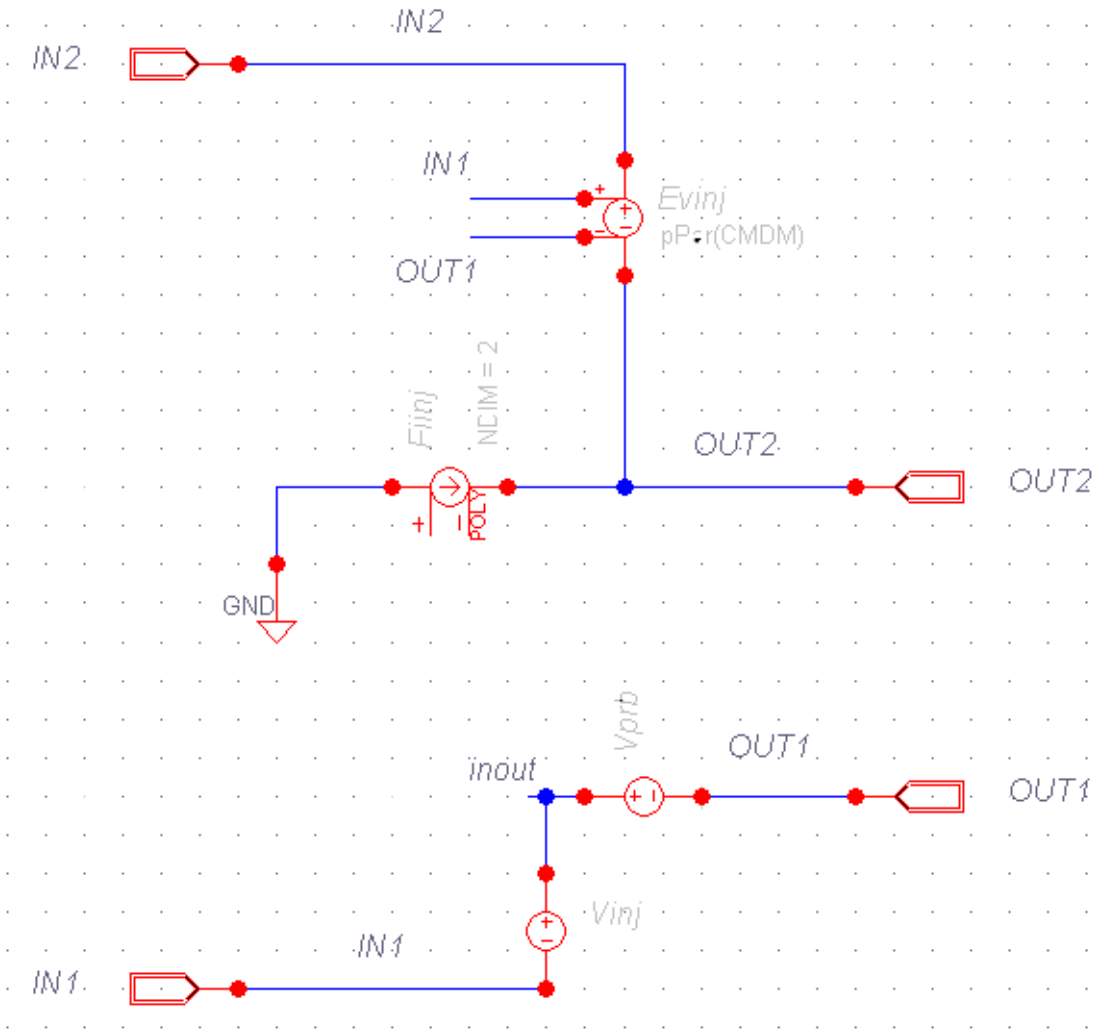
Example

```
OPAMP CLOSED-LOOP STABILITY ANALYSIS

.param freqrange=100MEG
.param resval=1k
.param acsupply=0

.option  nomod   nodeck

VS      1       0               AC     acsupply   PWL(0US 0V 0.1US 1V 10US 1V)*
R1      2       0    "resval"
* CS    2       0     10PF
R2      2       4    "resval"
CCOMP   2       4     3pF
XOP     1 2     vi_inj_vprobe   OPAMP1

Vprobe           vi_inj_vprobe   4       ac  0   dc 0
Rout       4  0     100

* OPAMP MACRO MODEL, SINGLE-POLE
* connections:      non-inverting input
*                   |   inverting input
*                   |   |   output
*                   |   |   |
.SUBCKT OPAMP1      1   2   6
* INPUT IMPEDANCE

RIN    1     2      10MEG

* DC GAIN (100K) AND POLE 1 (100HZ)

EGAIN  3 0   1 2    100K
RP1    3     4      1K
CP1    4     0      1.5915UF
```

```
* OUTPUT BUFFER AND RESISTANCE

EBUF-  5 0    4 0    1
FER
ROUT   5      6      10

.ENDS
*
* ANALYSIS
.LSTB      DEC     30 .1 "freqrange" mode=single  vsource=Vprobe
*.TRAN     0.01US   1US

.END
```

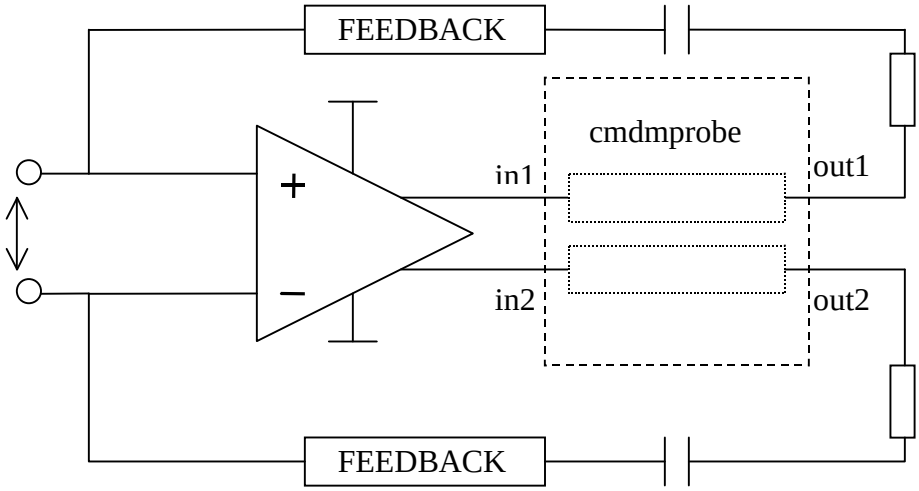
MODE=diff|comm

Differential and common mode are using `cmdmprobe` macro shown below:



Vsource=[v1 v2] analysis keyword is specifying insertion point in the circuit.

The inclusion of `cmdmprobe` in the circuit is shown below:


FEEDBACK
cmdmprobe
in1
out1
+
−
in2
out2
FEEDBACK

.MACRO (Subcircuit Definition)

Syntax

```
.MACRO | .MAC subcktname <n1 n2...>
```

This statement is an alias for .SUBCKT. See Section .SUBCKT (Subcircuit Definition) for details.

.MALIAS (Assign Alias to Model Name)

Syntax

```
.MALIAS real_model_name = alias_name_1 <alias_name_2 ...>
.MALIAS real_model_name   alias_name_1 <alias_name_2 ...>
```

real_model_name	The model name defined in the .MODEL statement.
alias_name_1	The alias that an instance of the model references.

The .MALIAS statement is used to assign an alias to a model that you defined in a .MODEL statement. The aliases can be used in a device definition.

Example

```
Q1 2 3 9 bjt_alias_1 1.5 IC=0.6,5.0
Q2 10 11 12 20 bjt_alias_2 OFF TEMP=50
.malias npn=bjt_alias_1 bjt_alias_2
```

In this example the instances Q1 and Q2 refer to the model npn through the model's aliases bjt_alias_1 and bjt_alias_2.

.MC (Monte-Carlo Analysis)

Syntax

```
.MC num_runs DC|TRAN|AC out_var measure <RANGE(from, at)>
+ <LIST> <OUTPUT output_type> <SEED=val> <speedplot>
```

This statement performs a Monte-Carlo statistical analysis of the circuit. Multiple runs of the specified analyses (DC, AC, or TRAN) are performed. Initially, the first run is performed using nominal parameter values. During the nominal run, all analyses specified in the input deck (.TRAN, .AC, .DC, .SNS, etc.) are performed. The model parameters are modified randomly in accordance with .MODEL statement specifications for each run of the .MC statement. During all subsequent runs, only the analyses specified in the .MC statement are performed. An input deck can have only one .MC, .WCASE or .SNS statement.

num_runs	Number of runs of the specified analyses.
DC | TRAN | AC	Type of analysis to be performed.
out_var	Name of the output variable for measurement.
measure	Type of measurement. Can be one of the following: • **YMAX:** Finds the greatest difference between the current waveform and the nominal waveform. • **MAX:** Finds the maximum value of the current waveform. • **MIN:** Finds the minimum value of the current waveform. • **RISE_CROSS (value):** Finds the argument at the first point above the threshold value. • **FALL_CROSS (value):** Finds the argument at the first point below the threshold value.
LIST	Prints the parameter values that are changed by the worst case analysis.
RANGE (from, at)	Defines the interval where the measure value will be evaluated. An asterisk (*) can be used in place of low or high_value. In this case, only one boundary will be effective.
OUTPUT	Parameter for output control. The variable output_type must be specified.
output_type	Defines the type of output. Must follow the parameter OUTPUT. Controls the output generated by .PRINT, .PLOT, .MEAS and .PROBE statements. Can be one of the following: • **ALL:** Permits output for all runs. • **FIRST val:** Permits output of the first val runs. • **EVERY val:** Permits output of every valth run. • **RUNS val**, or **val:** Permits output for the specified run numbers.

SEED=value	Defines the seed to initialize the random number generator. The variable value must be an odd integer between 1 and 32,765. Default is 17,533.
speedplot	Allows you to speed up simulation by reusing previously created SmartSpice structures (plot). Only one plot will be created, the waveforms will not be preserved. All measure results will be printed and preserved in the .meas plot. The raw files will contain only the data from the last sweep step and all measure results, if there is no .probe statements in the netlist. Default is off.

Examples

```
.MC 10 TRAN V(1) FALL_CROSS (0.001) LIST OUTPUT ALL
.MC 30 DC V(1,2) MAX LIST RANGE (*,5) SEED=135
.MC 50 AC VM(10) YMAX OUTPUT RUNS 5, 10, 20, 30, 40
```

The first example performs 10 transient analyses. For each analysis, the first point below 0.001 is found. All changed model parameters are printed before each run. All output produced by other statements are also printed.

The second example performs 30 DC analyses. The maximum of V(1,2) is evaluated for every run for all values less than 5. The seed for the random generator is set to 135.

The third example performs 50 AC analyses. Outputs are produced at the 5th, 10th, 20th, 30th and 40th runs.

.MEASURE (Analysis Measurements)

Syntax

```
.MEASURE|.MEAS analysistype resname meastype outvar1 <outvar2>
+ <measparam=number|val|name ...>
+ <NEST=-1|0|step> <OFF> <GOAL=val>
```

The .MEASURE statement calculates and stores circuit performance measures and other electrical specifications for output variables. It is used for parametric analysis and optimization. The following specifications are available (also, see Section 6.7 Measure Statement):

analysistype	Type of analysis where measurement is calculated. It must be one of the following: AC, DC, FFT, FOUR, NET, NOISE, PZ or TRAN.
resname	Name of measurement, used for references and output. The measurement result value is stored under this name.
meastype	Type of measurements. It must be one of the following:

AMAX	(see Section 6.7.9 MAX, MIN, AMAX, AMIN) Calculates the argument value of the maximum of outvar1 during the user-specified measurement interval.
AMIN	(see Section 6.7.9 MAX, MIN, AMAX, AMIN) Calculates the argument value of the minimum of outvar1 during the user-specified measurement interval.
AVG|MEAN	(see Section 6.7.2 AVG (MEAN), RMS, PP, INTEGRAL (INTEG)) Average: computes the area under the variable (outvar1), divided by the length of the user-specified time interval.
CROSS|WHEN	(see Section 6.7.3 CROSS (WHEN)) Calculates the argument value at the point of the intersection of outvar1 and outvar2, or outvar1 and a user-specified value during the specified measurement interval. Support for cross=last HSPICE compatible syntax.
DELAY|TRIG	(see Section 6.7.4 DELAY (TRIG)) Calculates the propagation delay between the middle points of the rise or fall of outvar2, and the rise or fall of outvar1 during the specified measurement interval. The middle points can either be specified directly using specified value or calculated as a function of the low and high values on either side of the middle point.
DERIVATIVE|DERIV	(see Section 6.7.5 DERIVATIVE (DERIV)) If measparam ARG0 is specified, then the value of the derivative of outvar1 at the point ARG0 is calculated. Otherwise, SmartSpice calculates the value of the argument where the derivative equals measparam VAL0. The specified measurement interval is considered.

ERR	(see Section 6.7.6 ERR, ERR1, ERR2, ERR3) Calculates error as the square root of the sum of squares of the difference between `outvar1` and `outvar2`.
ERR1	(see Section 6.7.6 ERR, ERR1, ERR2, ERR3) Average error.
ERR2	(see Section 6.7.6 ERR, ERR1, ERR2, ERR3) Average absolute relative error.
ERR3	(see Section 6.7.6 ERR, ERR1, ERR2, ERR3) Average logarithmic relative error.
EXPR|PARAM	(see Section 6.7.7 EXPR (PARAM)) Computes the value specified by expression.
FIND	(see Section 6.7.8 FIND) Calculates the value of `outvar1` when specified output vector intersects another specified output vector or intersects a constant value or at specified timepoint.
INTEGRAL|INTEG	(see Section 6.7.2 AVG (MEAN), RMS, PP, INTEGRAL (INTEG)) Computes the integral of the variable `outvar1` within the user-specified time interval.
MAX	(see Section 6.7.9 MAX, MIN, AMAX, AMIN) Calculates the maximum value of `outvar1` during the user-specified measurement interval.
MIN	(see Section 6.7.9 MAX, MIN, AMAX, AMIN) Calculates the minimum value of `outvar1` during the user-specified measurement interval.
POINT	(see Section 6.7.10 POINT) Calculates the value of `outvar1` at the specified point.
PP	(Peak to Peak, see Section 6.7.2 AVG (MEAN), RMS, PP, INTEGRAL (INTEG)) computes the maximum value minus the minimum value of the variable `outvar1` within the user-specified time interval.
PRINT	`PRINT=0` does not print the measure result into the measure raw file. `PRINT=1` (Default) prints the measure result into the measure raw file.
RMS	(Root Mean Square, see Section 6.7.2 AVG (MEAN), RMS, PP, INTEGRAL (INTEG)) computes the square root of the area under the variable `outvar1` divided by the length of the user-specified time interval.
WAVE	(see Section 6.7.11 WAVE) Calculates the duration of the rise or fall of `outvar1` over a specified interval.

For more information on measurement types and their syntax, see Chapter 6 Outputs of this manual.

outvar1, outvar2	Names of output variables (Trigger and Target specifications).
measparam=number|val|name	Parameters which specify the measurement conditions. It must be as follows:

FROM=val|name	Argument value where measurement begins. Defines the first point of the user-specified interval. The default value is the first point of the analysis interval that is stored.
TO=val|name	Argument value where measurement ends. Defines the last point of the user-specified interval. The default value is the last point of the analysis that is stored.
RISE=number	Specifies which rise must be calculated. A zero value is used to measure the last rising edge of the vector.
FALL=number	Specifies which fall must be calculated. A zero value is used to measure the last falling edge of the vector.
VAL=val|name	• For DELAY measurements, the values of outvar1 and outvar2 at which the counter for the rises and falls during the delay calculation is incremented by one. • For CROSS measurements, the crossing value. • For ERR, ERR1, ERR2, ERR3 measurements, the denominator value for relative error calculations, when the outvar1 is less than VAL.
CROSS=val	Calculates the point where outvar1 and VAL (rather than outvar2) cross.
CROSS=last	Calculates using the last crossing point in HSPICE compatible mode.

Note: Parameters RISE/FALL/CROSS accept ALLEVENT value during WHEN measurement. In this case, the measurement counts all the events (all rises or all falls or all crosses).

Support for cross=last HSPICE compatible syntax.

VAL0=val|name, VAL1=val|name	• For WAVE and DELAY measurements, VAL0, VAL1 are the values of outvar1 and outvar2 that are used as levels for either the rise or fall calculations (instead of VAL). • For DERIV measurements, VAL0 specifies the derivative value for which the corresponding argument value is to be computed.

ARG0=val|name, ARG1=val|name	• For WAVE and DELAY measurements, ARG0, ARG1 are the argument values which correspond to the VAL0 and VAL1 values and could be given instead of VAL, VAL0 and VAL1. ARG0 and ARG1 must be within the specified measurement interval. • or POINT and DERIV measurement, ARG0 specifies the argument value which determines where to compute the respective value or derivative of outvar1.
OCCUR=number	• For CROSS measurement, specifies which cross of outvar1 and outvar2 to be calculated. A zero value is used to measure the last crossing point. • For DERIV measurement, shows what argument value must be calculated. Default is 1.
COEF=val|name	Real levels for either the rise or fall calculations to be used instead of the values for VAL0 and VAL1. COEF must be greater than or equal to 0 and less than 0.5. Default is 0.1.
DIFF=val|name	The absolute value of the difference between VAL0 and VAL1 must be greater than DIFF. Default is 0.0.
AT=val|name	Start point of measurement.
TARG=outvar2	Indicates where information about the second variable begins.
TD=number	Amount of simulation time that must elapse before the measurement is enabled. The number of crossings, rises, or falls is counted only after TD value. Default is 0. When the parameter TD is specified in TRIG, and is not specified in the TARG section of the DELAY measurement .MEASURE statement, the TRIG TD value will be used for both the TRIG and TARG functions when counting crossing points. Rise/Fall/Delay measurement statement supports TRIG/TRIG-TIME value of TD parameter of TARG subcommand. A target measurement is postponed until the trigger measurement count all edges.

Example

```
.MEAS TRAN trig_td TRIG v(1) VAL=3 FALL=2 TARG v(2) VAL=1.5 RISE=1
TD=TRIG
```

In this example the target measurement is postponed until the trigger measurement counts two falling edges. The target measurement starts from the trigger time, not from the time of the beginning of the simulation.

name	Name of previously calculated measurement, parameter label, or an expression of measurement values, labels, and numbers. Expressions must be enclosed in single quotes.

NEST	Works for nested sweeps: • **NEST=0:** Default value. SmartSpice computes the measurement for each segment of the sweep and stores the last result. • **NEST=step:** SmartSpice computes the measurement for the `step` segment of the sweep. `step` must not be greater than the number of steps of the sweep. • **NEST=-1:** SmartSpice calculates the measurement without dividing the entire argument into segments of the sweep. Works with nested DC sweeps and for `MAX`, `MIN`, `AMAX`, `AMIN`, `ERR`, `ERR1`, `ERR2` and `ERR3` measurement types.
OFF	Suppresses result printing.
GOAL=val	Goal value for Bisection optimization. Default is `0`.

Note: If `NOISE` is specified, the corresponding measurement will be computed for the NOISES (spectral density) plot only.

Arguments must be monotonic for `CROSS`, `WAVE`, `DELAY`, `AVG`, `RMS`, `DERIV`, `INTEGRAL` and `POINT` measurement types, and their measurement intervals must have at least two points except for `POINT`.

In any measurement statement, `.MEAS` can be used in place of `.MEASURE`.

For further details, see Section 6.7 Measure Statement.

Examples

```
.MEASURE DC MAX_V3 MAX V(3)
```

In this example, SmartSpice calculates the maximum value of the output variable `V(3)` and stores this maximum value in the variable `MAX_V3`.

```
.MEASURE TRAN FALL_V1 WAVE V(1)
+ FALL=1 VAL0=MIN_V1 VAL1=MAX_V1
```

In this example, SmartSpice calculates the fall time of the first wave for the output variable `V(1)`. `MIN_V1` and `MAX_V1` are two previously calculated minimum and maximum levels of the wave.

```
.MEASURE TRAN DEL_V3_V1 DELAY V(1) RISE=1 VAL=0
+ TARG=V(4) FALL=1 VAL0=MIN_V4 VAL1=MAX_V4
```

In this example, SmartSpice compares the first fall of `V(4)` and the first rise of `V(1)`, and uses the results to calculate the propagation delay of `V(4)` with respect to the input signal `V(1)`.

```
.MEASURE AC MAX_VM3 MAX VM(3)
```

In this example, SmartSpice calculates the maximum value of the magnitude of the complex variable `V(3)`.

```
.MEASURE TRAN RSDEL DELAY V(IN) RISE=1 VAL0=0
```

```
+ VAL1=5 TARG=V(OUT)
.MEASURE TRAN FLDEL DELAY V(IN) FALL=1 VAL0=0
+ VAL1=5 TARG=V(OUT)
.MEASURE TRAN DELDIFF EXPR VAL="abs(fldel-rsdel)"
```

In these three examples, SmartSpice computes the absolute value of the difference between two propagation delays of the node voltages V(IN) and V(OUT).

```
.MEASURE DC ERR_V1_V2 ERR V(1) V(2) NEST=3
```

In this example, SmartSpice calculates the difference between V(1) and V(2) for the third step of the nested DC sweep.

```
.LET noises density='sqrt(onoise_s)'
* defines density
.MEASURE noise noise500k find density at=500k
* finds the value of density at 500kHz
 .MEASURE noise noise1k find density at=1k
* finds the value of density at 1kHz
.MEASURE noise ratio expr val='noise1k/noise500k'
* calculates the ratio
```

In this example SmartSpice computes the noise ratio useful for determining OpAmp characteristics. This shows how powerful the measure statement can be made.

Relative measure operates with ternary operators:

```
.PARAM t1=0.4
.MEASURE m find v(1) at 4n
.MEASURE mout param='m>t1 ? m : 1-m * m'
.MEASURE mout1 param='((mout-1)<m) ? mout+1:1-m'
.MEASURE mout2 param='mout1>m ? (mout1<t1 ? t1: mout1+1):1+m'
.MEASURE mout3 param='(m>mout2) ? 10 * mout1 : 1 - mout2'
```

For further reading, see Section 6.7 Measure Statement.

The support for following HSPICE compatible .meas syntax has been added.

```
.measure resname deriv outvar at= value
```

Compound relative measurement for AC and Transient analyses:

```
.MEASURE AC  relAC1  find 'vm(out1)' at=10x
.MEASURE AC  relAC2  find 'vm(out2)' at =20x
.MEASURE AC  relAC3  param='relAC1/relAC2'
```

relAC3 is a result of previously calculated .MEASURE's.

An Index feature has been added to measurement results. If a .MEASURE statement produces several measurements, there is a possibility to reference each result using an index. Index is zero-based. Using a measurement name without any index provides the last measured result.

```
.MEAS TRAN when_rise WHEN V(1)=2.0 RISE=ALLEVENT
.MEAS TRAN when_rise_calc EXPR VAL="when_rise*2"
.MEAS TRAN when_rise_calc_0 EXPR VAL="when_rise[0]*2"
.MEAS TRAN when_rise_calc_1 EXPR VAL="when_rise[1]*2"
.MEAS TRAN when_rise_calc_2 EXPR VAL="when_rise[2]*2"
```

.MODEL (Model Definition)

Syntax

```
.MODEL mname <AKO: refmname> mtype
+ <(> <pname1=val1 ...> <EACH|ALL </num_gen>
+ </dist_name>=val2<%>> <)>
```

mname	The model name. It is used by elements to reference a particular model.
AKO	Parameter (acronym for “A Kind Of”) followed by reference model name `refmname`. It indicates that the parameters of the model `mname` will have the same values as corresponding parameters of model `refmname`, unless they are specified in this `.MODEL` card.
pname1, param2,...	The parameter name. For each `pname`, a corresponding `val` is assigned. Model parameters that are not included are assigned their default values.

mtype	The model type. It is one of the following: • **C:** Capacitor model. • **CORE:** Magnetic core model. • **CSW:** Current-controlled switch model. • **D:** Diode model. • **LTRA, TXL:** Lossy transmission line model. • **MOSVAR:** MOS varactor model • **NJF:** N-channel JFET model. • **NMF:** N-channel MESFET model. • **NMOS:** N-channel MOSFET model. • **NPN:** NPN BJT model. • **NTFT:** N-channel TFT model. • **NTYPE:** User-defined model. • **PJF:** P-channel JFET model. • **PMF:** P-channel MESFET model. • **PMOS:** P-channel MOSFET model. • **PNP:** PNP BJT model. • **PTFT:** P-channel TFT model. • **PTYPE:** User-defined model. • **R:** Resistor model. • **SP:** Multi-terminal networks model. • **SW:** Voltage-controlled switch. • **T:** Lossless transmission line model. • **U:** Lumped transmission line model. • **W:** Coupled lossy transmission line model. • **OPT:** Bisection optimization model.
EACH | ALL	Defines the tolerance type for a parameter. If ALL is used, all devices referring to this model are given the same parameter value. When EACH is used for each device that refers to this model, the parameter value will be changed independently. ALL and EACH can both be set for one parameter.
num_gen	Index of the random generator for this parameter. There are 10 generators for EACH (0-9) and ALL (0-9). A generator can be referred to in different device tolerances.

dist_name	Name of distribution. Can be one of the following: • **UNIFORM:** Generates a uniformly distributed random variable ranging from (val-val2) to (val+val2). • **GAUSS:** Generates a random variable with Gaussian distribution over a range of $\pm 3\sigma$. The effective standard deviation of a random sample will be 3 · val2. • **user_name:** Generates random variables with user-defined distribution ranging from (val-val2) to (val+val2). See Section .DISTRIBUTION (User-Defined Distribution) for more information.
val2	The absolute or relative variation. If there is a percentage sign (%) after val2, the absolute variation is calculated as: *0.01val x val2*.

The .MODEL statement specifies a set of model parameters that are used by one or more devices. Devices reference only models of the correct type. For example, BJT device references only models of type NPN or PNP. One input file can contain more than one model of the same type, but each of these models must have a different name.

Examples

```
.MODEL RMOD R (TC1=0.003 ALL/1/GAUSS=1%
+ TC2=0.002
+ EACH/TEST=0.00001)
.MODEL RMOD R (TC1=0.03)
.MODEL DIODE D IS=1.0E-14 TT=0.1N CJO=2P
.MODEL QNL NPN (BF=80 RB=100 CSS=2PF TF=0.3NS
+ TR=6NS CJE=3PF CJC=2F)
.MODEL MN NMOS (LEVEL=2 LD=0.28u TOX=5E-8
+ NSUB=1E16 VTO=0.827125)
.MODEL NE2T NMOS (LEVEL=3 UO=510.0 VTO=800.0M
+ NFS=650.0G)
.MODEL TEST NJF KF=1E-20 AF=1 RS=10 RD=10
.MODEL QNL1 AKO:QNL NPN (BF=70 IS=1e-15)
```

In the first example, parameter TC1 in model RMOD is changed for all devices with a Gaussian distribution and a random generator number 1. The real deviation is $0.003 \times 1 \times 0.01$. The parameter TC2 is changed separately and independently for each device in accordance with the user defined definition TEST.

In the last example, parameter values of the model QNL1 are set to parameter values of the reference model QNL, except for the parameters BF and IS, which are explicitly set on this .MODEL card.

.MODIF (Parameter Modification)

Syntax

```
.MODIF <<param1 <+|-|*|/>=rhs>
+ <param2_spfc> <LOOP <=nreps>> <SEED=val>
+ <STOP lname|lval LE rname|rval>
+ <PROFF> <PRTBL> <PRMC> <RESTORE>
+ <MODIF parset2> <MODIF parset3> ...
+ <MODIF DATA rhs1 rhs2 ...>
+ <SWEEP ...>
+ <NESTED>
```

or for Bisection optimization

```
.MODIF <NESTED> <PROFF> <PRTBL>
+ OPTIMIZE  parname = OPT(lower  upper  <initval>)
+ TARGETS  measname = goalval
+ OPTIONS MEATHOD=val <MAXERR=val> <NUMITER=val> <SOLUTIONONLY<=1>>
```

This statement allows you to simulate a circuit for some sets of parameters as they are modified over a user-specified range of values. It simultaneously modifies all of the parameters in a set, and performs all analyses on the input deck for this set. SmartSpice terminates the parameter modification process for a set of parameters when one of the user-defined stop conditions, specified in this set, is satisfied.

param1, param2	Either the name of a device parameter or the name of a model parameter, or: • **TEMP:** Temperature. • **GMIN/DCGMIN:** Model and computational parameter. • **RELTOL:** Relative error tolerance. • **ABSTOL:** Absolute current error tolerance. • **VNTOL:** Absolute voltage error tolerance.

It is also possible to specify a parameter for all devices of the same type. The syntax for this is:

```
ALL@devtype <parname>
```

SmartSpice modifies the parameter `parname` of all devices of the type `devtype`.

+, -, *, /	The arithmetic operators.
rhs	Parameter value, increment or multiplier. It is one of the following types:

```
<(initval)> val
LIST (val1 <val2 val3 ...>)
```

For Monte-Carlo analysis:

- `AUNIF(nomval absvar <mult>)`
- `UNIF(nomval relvar <mult>)`
- `AGAUSS(nomval absvar sigma <mult>)`

- GAUSS(nomval relvar sigma <mult>)
- ALIMIT(nomval absvar)
- LIMIT(nomval relvar)
- AWEIBULL(nomval absvar <csign> cval rlim)
- WEIBULL(nomval relvar <csign> cval rlim)
- EXPONENTIAL(val): Exponential distribution function with mean val. A value val must be more than 0.

  The probability density function (pdf) of an exponential distribution is:

  $$\rho(x, \lambda) = \lambda \bullet \exp(-\lambda x)$$

  $$x \geq 0, \lambda > 0$$

- LOGNORMAL(val,dev): Lognormal distribution function with mean val and standard deviation dev.

  The pdf of a log-normal distribution is:

  $$\rho(x, \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \bullet \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right)$$

  $$x > 0, \sigma^2 > 0$$

- RAYLEIGH(val): Rayleigh distribution function with mean val. A value val must be more than 0.

  The pdf of the Rayleigh distribution is:

  $$\rho(x, \sigma) = (x)\sigma^2 \bullet \exp(-x^2(2\sigma^2))$$

  $$x \geq 0, \sigma > 0$$

Example

```
.MODIF LOOP=100
+ PAR_EXP=exponential(3.6e-4)
+ PAR_LOG=lognormal(3.6e-4, 3.6e-5)
+ PAR_RAY=rayleigh(3.6E-4)
```

Syntax

```
.MODIF <NESTED> ...
```

The parameter NESTED has been added to the specification. This allows one statement to be run inside another to give a compound effect.

Example

```
.PARAM R1=0
.PARAM R2=0
.PARAM R3=0
.MODIF NESTED PRTBL PRMC LOOP=3 R1=agauss(0,1,1) $ loop1
+ MODIF LOOP=4 R2=agauss(0,1,1) $ loop2
+ MODIF LOOP=5 R3=agauss(0,1,1) $ loop3
```

Previously, you could do successive MODIF sets, but they were independent of each other. Now you can run one upper MODIF set for all lower level MODIF iterations. For the example above loop1 is the lowest level of the iterations, and loop3 is the highest level of the

looping. For each set of variations in loop1 one iteration in loop2 is performed. The total number of iterations performed in the example is *3*4*5=60*. Care should be taken that the total number of iterations can become large, and consume a lot of resources and time. With this new capability, a multi-dimensional variation can be studied.

Operation

- STEP1: SmartSpice saves the loop variables and initializes the loop parameters.
- STEP2: SmartSpice performs all analyses on the input deck using loop1.
- STEP3: SmartSpice increments the loop2 parameters, initializes the parameters in loop1, and runs STEP2.
- STEP4: The previous steps are performed for every single iteration at the higher level.
- STEP5: SmartSpice finishes with the last iteration in the highest level loop (loop3).
- STEP6: Restores all the loop parameter sets.

Note: The parameters PROFF, PRTBL, PRMC and RESTORE must have the same consistent values within a set of NESTED .MODIF statements. These parameters should be defined in the first set definition.

NESTED	Can be used together with the bisection optimization. It allows running optimization for each nested MODIF sets.

Example

```
.PARAM OPTPARAM = 3.6n
.PARAM P1 = 0
.PARAM P2 = 0

.MODIF NESTED
+ PRTBL
+ OPTIMIZE OPTPARAM = opt (3.6n 4.0n 3.8n)
+ TARGETS TARGET1=1
+ OPTIONS METHOD=2 NUMITER=20 SOLUTIONONLY=1
+ MODIF PRTBL PRMC LOOP=3
+ P1=AGAUSS(0,1,1)
+ P2=AGAUSS(0,1,1)
```

In this example, SmartSpice will run bisection optimization 3 times. Before running each optimization, SmartSpice will change the P1 and P2 parameter values accordingly. For the first optimization the parameter P1=0 and P2=0. For the last two optimizations the parameters P1 and P2 will use Gaussian distribution function.

SWEEP	Allows the use SmartSpice SWEEP syntax in the .MODIF statement.

Example

```
.PARAM P1=0
.PARAM P2=0

.DATA MC_DATA P1 P2
+ 1 -1
+ -1 1
```

```
+ 2 -2
.ENDDATA

.MODIF NESTED
+ PRTBL
+ OPTIMIZE DDELAY  = opt (3.6n 4.0n 3.8n)
+ TARGETS  FLOP_SWITCH=1
+ OPTIONS  METHOD=2  NUMITER=20 SOLUTIONONLY=1
+ MODIF SWEEP DATA=MC_DATA
```

In this example, SmartSpice runs optimization 3 times. Before running each optimization, SmartSpice will change the P1 and P2 parameter values accordingly. For the first optimization the parameter P1=1 and P2=-1. For the second optimization P1=-1 and P2=1. And for the last optimization P1=2 and P2=-2.

Other Parameters

initval	The initial value of the parameter param1 enclosed in parenthesis.
val	The value for param1 modification. It can be a number or the name of a measure calculated in analysis. This name must be specified in a .MEASURE statement.
LIST	Specifies a list of values to be applied to the parameter. This list is enclosed in parentheses and must contain at least one value.
AUNIF	Uniform distribution function with absolute variation.
UNIF	Uniform distribution function with relative variation.
AGAUSS	Gaussian distribution function with absolute variation.
GAUSS	Gaussian distribution function with relative variation.
ALIMIT	Random limit distribution function with absolute variation, $\pm absvar$ is added to nomval based on whether the random outcome of a -1 to 1 distribution is greater or less than 0.
LIMIT	Random limit distribution function with relative variation.
AWEIBULL	Weibull distribution function with absolute variation.
WEIBULL	Weibull distribution function with relative variation.
nomval	Nominal value for Monte-Carlo analysis.
absvar	The absolute variation. The AUNIF, AGAUSS and AWEIBULL vary nomval by $\pm absvar$.
relvar	The relative variation. The UNIF, GAUSS and WEIBULL vary nomval by $\pm nomval \times relvar$.
mult	The multiplier repeats function calculation mult times, saving only the largest deviation. Default is 1.

sigma	The parameter for Gaussian distribution. The effective standard deviation of a random sample will be *3·absvar/sigma*.
csign	Plus or minus. This sign specifies the orientation of the non-symmetrical Weibull distribution.
cval	The parameter c of the standard Weibull density function $p(x) = cx^{c-1}\exp(-x^{c}),(x>0)$.
rlim	The parameter for Weibull distribution.
param2_spfc	The specifications for param2. These specifications have the same style as the specifications for param1.
LOOP	Specification of the desired number of repetitions.
nreps	The number of steps to be repeated in the parameter set. SmartSpice stops the parameter modification process for this set when the number of repetitions reaches nreps. Default is 5.
SEED=val	Seed to initialize the random number generator. val must be an odd integer between 1 and 32765. Default is 1.
STOP	Specification of a stop condition.
Nested	Allows one statement to be run inside another to give a compound effect.
lname, rname	The names of measures calculated during analyses.
lval, rval	A value for the stop condition. The .MODIF statement can contain only one (lval or rval).
LE	The mandatory relational parameter between the left and right sides of the stop condition. SmartSpice terminates the parameter modification process if: • lname ≤ rval • lname ≤ rname • lval ≤ rname
PROFF	Suppresses online printing of measures specified and calculated by means of .MEASURE statements for the current set of parameters.
PRTBL	Causes SmartSpice to print the final table of all parameters and measures calculated for the current set of parameters.
PRMC	Causes SmartSpice to print Mean, Variance, Sigma and Average Deviation for each measure calculated during Monte-Carlo analysis.

RESTORE	This parameter causes SmartSpice to restore all modified parameter values in the specified set to their original values. This works for any set, including those with STOP conditions.
OPTIMIZE	This parameter is followed by the bisection optimization parameter.
parname	Name of the bisection optimization parameter.
OPT	Mandatory parameter followed by lower, upper, and initval in parentheses.
lower	Left bound for bisection optimization parameter.
upper	Right bound for bisection optimization parameter.
initval	Initial value for bisection optimization parameter. Default is *initval=(lower+ upper)/2*.
TARGETS	This parameter is followed by the target for optimization.
measname	Name of the measure calculated by a .MEASURE statement. It can be a circuit performance measure like delay, rise/fall time, maximum/minimum value, or difference between a desired and a simulated curve calculated by the ERR .MEASURE statement.
goalval	Goal value.
OPTIONS	This parameter is followed by the list of optimization control options.
METHOD=val	Parameter to indicate which bisection optimization method to use: • 1 is BISECTION method. The measure results for lower and upper bounds of optimization parameter must be on opposite sides of GOAL value. • 2 is PASSFAIL method. The measure must pass for one limit and fail for the other limit.
MAXERR=val	Relative optimization parameter error tolerance. When the difference between the two latest test input values is smaller, then: $$parTol = max(interval \times MAXERR, 10^{-16}),$$ where $$interval = max((initval - lower), (upper - initval))$$, bisection optimization process will be terminated. Default is 0.01.

NUMITER=val	Maximum number of iterations. The bisection optimization process will be terminated, when the number of iterations reaches: $ITERlimit = max(((\log(1/(MAXERR)))/(\log(2)) + 10), NUMITER)$ Default is NUMITER=15.
SOLUTIONONLY	If SOLUTIONONLY=1, SmartSpice will keep only solution waveforms and measures, and suppress error messages of failed measurements. Default is 0.
MODIF	This parameter starts the next set of parameters, and is not preceded by a period (.).
MODIF parset2	The specifications for the second set of parameters.
AUTOSTOP	Parameter followed by the list of stop conditions for basic output variables. SmartSpice will stop the Transient analysis when all AUTOSTOP conditions are satisfied. The AUTOSTOP specification is placed at the end of the .MODIF statement.
outvar1	Name of a basic output variable.
numrep	Number of rises or falls.
pname1	Name of a device or a model parameter.
num1	Value for stop condition.
cond2	Stop condition for the basic variable outvar2.
DATA	Parameter followed by a number of rhs values. This parameter can be used to repeat the previous measurement of rhs on multiple sets of the same parameters with new values. The value of rhs can only be a number preceded by optional initial conditions.

The .MODIF statement cannot be run with the .ST statement in the same input deck. .MODIF, .TEMP and .DC statements must not attempt to set the same parameters.

Note: The parameter modification process is cumulative. Each set of parameters begins with the parameter values generated by the previous iteration.

Any device and model parameter can be specified in the .MODIF statement in one of two ways: using the full path name or the parameter labels.

The following forms are used to specify parameters using the full path name:

```
device(parname)=rhs
```

devname	User-specified device name.

parname	Name of a parameter. Must be in parentheses.
rhs	Value of the parameters.

or

```
modname(parname)=rhs
```

modname	User-specified model name.
parname	Name of a parameter. Must be in parentheses.
rhs	Value of the parameters.

A device or model parameter can also be specified using a parameter label defined in the .PARAM statement, and referred to in the model or device specification.

Each method offers certain advantages. The advantages of the full path name are:

- Allows use of an input netlist without any changes in the circuit description.
- Provides access to any model or device when subcircuits are used.

Transient analysis voltage and current source parameters have notations, but do not have names. For example: V1, V2, td, tr, tf, pw and per are notations for initial value, pulsed value, delay time, rise time, fall time, pulse width, and period of a pulse waveform. These notations can not be used in full-path names. The names PAR1, PAR2,..., PAR7 can be used in place of these notations. For example, the name vin(PAR3) can have any of the following meanings:

- Rise delay time for an exponential waveform.
- Delay time for a pulse waveform.
- Carrier frequency for a single-frequency waveform.
- Frequency for a sinusoidal waveform.

The advantages of the parameter label method are:

- The same value can be set or updated for different models and devices.
- Arbitrary dependencies between different device and model parameters can be introduced by parameter expressions in the .PARAM statement.
- Labels can be used in the .MEASURE statement as a right hand side (rhs) value for most parameters.

When the value of a model or device parameter is loaded by means of a parameter label defined in the .PARAM statement, the full path name of this parameter can not be used in a .MODIF statement. A parameter label can not be used in a .MODIF statement if:

- The label is not defined in a global .PARAM statement.
- The label is defined as a function of another parameter label.
- The label is referred to in a Polynomial, Piecewise Linear, or Behavioral type controlled voltage (E and H) or current (F and G) source.

Examples

The following examples are consistent with Example 2 of Chapter 33 Examples.

```
.MODIF RC1(RES)=2K CLOAD(CAP)=3PF
```

In this example, SmartSpice first sets the parameter RES of the resistor RC1 to a value of 2K, the parameter CAP of the capacitor CLOAD to a value of 3PF, and performs all of the analyses of the input deck.

```
.MODIF LOOP=10 RC1(RES)+=0.1K CLOAD(CAP)+=0.2PF
```

In this example, SmartSpice simultaneously modifies the parameters RES and CAP ten times. The increments are 0.1K and 0.2PF, respectively. On the first iteration, SmartSpice adds these increments to the original parameter values in the input file.

```
.MODIF RC1(RES)=2K CLOAD(CAP)=0.5PF
+ MODIF LOOP=9 STOP DEL_V3_V1 LE 1.3NS RC1(RES)*=0.9
+ CLOAD(CAP)*=0.95
```

In this example, SmartSpice performs nine or fewer iterations. In the first iteration, it sets the parameters RES to a value of 2K and CAP to a value of 0.5pF. On each of the following iterations, SmartSpice multiplies the parameter RES by the coefficient 0.9 and the parameter CAP by the coefficient 0.95. It terminates the parameter modification process when the measured value DEL_V3_V1 is less than or equal to 1.3NS.

```
.MODIF LOOP=9 STOP DEL_V3_V1 LE 1.3NS
+ RC1(RES)*=(2K)0.9 CLOAD(CAP)*=(0.5PF)0.95 PRTBL
```

This example is similar to the previous one. After the parameter modification process has finished, SmartSpice prints the final table of parameters and measures.

```
.MODIF MODIF STOP 2.6 LE MAX_TR_V3 RBIAS(RES)*=1.2
```

In this example, SmartSpice performs six or fewer iterations. On the first iteration, it performs all analyses in the input file without changing parameters. On each subsequent iteration, SmartSpice multiplies the parameter RES of the resistor RBIAS by the coefficient 1.2. It terminates the parameter modification process when MAX_TR_V3 is greater than 2.6.

```
.MODIF LOOP=30 RC1(RES)=UNIF(2K 0.1)
+ CLOAD(CAP)=AGAUSS(1PF 0.15PF 3 10)PRMC
```

In this example, SmartSpice performs 30 iterations of Monte-Carlo analysis using a relative uniform distribution for the parameter RES of the resistor RC1 and absolute bimodal Gaussian distribution variation *±0.15* pF at 3sigma for the parameter CAP. After the analysis is finished, SmartSpice prints a table with mean, variance, sigma, and average deviation for each calculated measure.

```
.MODIF ALL@R(RES)*=1.2 QNL(RB)=120
```

In this example, for all resistors, SmartSpice multiplies the parameter RES by coefficient 1.2, and sets the parameter RB of the transistor model QNL to a value of 120.

```
.MODIF VIN(PAR4)=1NS VIN(PAR5)=1NS
```

In this example, SmartSpice sets the rise and fall times of the pulse waveform of voltage source VIN to a value of 1ns.

```
.MODIF LOOP=10 TEMP=-50 VCC(DC) + = (10)0.2
+ MODIF DATA  27  (10)0.2
+           100  (10)0.2
```

This example is equivalent to:

```
.MODIF LOOP=10 TEMP=-50 VCC(DC)+=(10)0.2
+ MODIF LOOP=10 TEMP=27 VCC(DC)+=(10)0.2
```

```
+ MODIF LOOP=10 TEMP=100 VCC(DC)+=(10)0.2

.MODIF LOOP=3 TEMP=LIST (-50 27 100)
+ VCC(DC)=LIST(5 10 15)
+ VEE(DC)=LIST(-5 -10 -15)
```

On the first iteration in this example, SmartSpice sets TEMP to -50C, VCC(DC) to 5V and VEE(DC) to -5V. The second iteration uses the second set of listed values (27, 10 and -10), and the third set of values (100, 15 and -15) is used in the third iteration.

In the following input file, both the parameter labels and the full path parameter names are used. The parameter labels defined in the .PARAM statement are:

vccDC	Referred to in the voltage source VCC specification, in the first .MEASURE statement and in the .MODIF statement.
trtf	Referred to in the pulse waveform rise and fall time specification.
wp	Referred to in the transistor m1 width specification.

The full-path name m2(w) is specified in the .MODIF statement.

Parametric Analysis

```
***** Circuit description
vcc vss 0 DC vccDC
vin inp 0 PULSE(0 3 0 trtf trtf 10ns 40ns)
m1 2 inp vss vss pm w=wp l=1.6u
m2 2 inp 0 0 nm w=30u l=2.0u
cout 2 0 50ff
.MODEL pm PMOS (level=3 tox=.02e-6)
.MODEL nm NMOS (level=3 tox=.02e-6)
***** Analysis statement
.TRAN 0.1ns 25ns
***** Measure statements
.MEASURE TRAN delrise DELAY v(inp) RISE=1 VAL=1.5
+ TARG=v(2) FALL=1 VAL='0.5*vccDC'
.MEASURE TRAN maxv2 MAX v(2)
.MEASURE TRAN delfall DELAY v(inp) FALL=1 VAL=1.5
+ TARG=v(2) RISE=1 VAL='0.5*maxv2'
***** Parameter labels
.PARAM vccDC= 5V wp=4.9u trtf=1ns
***** Parametric analysis specification
.MODIF proff prtbl LOOP=6
+ vccDC+=(4.5)0.1v m2(w)=32.6u
+MODIF proff prtbl TEMP=75 LOOP=6 vccDC+=(4.5)0.1v
.END
```

The .MODIF statement performs parametric analysis for two sets of parameters. For the first set, the width w of the transistor m2 is set to a value of 32.6µm, then parameter vccDC is swept for six iterations from 4.5 to 5.

For the second set, the temperature is set to a value of 75 degrees Celsius, and the parameter vccDC is swept. The flag PROFF suppresses online printing of the .MEASURE statement results. The flag PRTBL instructs SmartSpice to print the final table of results.

In each iteration, SmartSpice performs a Transient analysis and calculates measures. The parameter VAL of the first .MEASURE statement is defined as a function of the power supply voltage vccDC. The .MEASURE statement that calculates the delay delfall uses the previously calculated .MEASURE statement maxv2.

The parametric analysis results for the first set of parameters are shown in Table 3-4

Table 3-4 Results for Set #1

Parameters and Measurements				
m2(w)	**vccdc**	**delrise**	**maxv2**	**delfall**
3.260e-05	4.500e+00	1.691e-11	4.507e+00	1.859e-10
3.260e-05	4.600e+00	2.946e-11	4.607e+00	1.809e-10
3.260e-05	4.700e+00	3.916e-11	4.707e+00	1.780e-10
3.260e-05	4.800e+00	4.750e-11	4.807e+00	1.710e-10
3.260e-05	4.900e+00	5.474e-11	4.907e+00	1.437e-10
3.260e-05	5.000e+00	5.972e-11	5.007e+00	1.374e-10

.NET (Small-Signal Network Analysis)

Syntax

```
.NET INPORT OUTPORT DEC|LIN|OCT nump fstart fstop
+ <param1=value ... paramN=value>
+ <sw2_spcf> < <SWEEP> MONTE=val <PRMC> >
```

Small-signal network analysis is a powerful tool integrated into SmartSpice**,** capable of extracting all types of two-port network parameters:

- Scattering parameters (S)
- Impedance parameters (Z)
- Admittance parameters (Y)
- Hybrid parameters (H)

Network analysis also analyzes transducer power gain, maximum available power gain, input/output impedance, and so forth. It is used for single-device simulation and parameter extraction, as well as for microwave amplifier, oscillator, and filter designs.

Network analysis is closely connected to small-signal AC analysis. When performing a network analysis, SmartSpice adds two measurement circuits (realizing the actual technique for S-parameter measurement) to the specified circuit. These measurement circuits do not disturb the DC operating conditions. SmartSpice also provides a feature for setting DC bias conditions for input and output ports in cases where the circuit does not have built-in power supplies, such as a single-device measurement. In this case, the DC sources (voltage or current) will be used only for DC bias supply and will not be taken into account during the network analysis.

The primary quantities extracted by SmartSpice are S-parameters obtained for some value of matching impedance (typically 50 Ohm). After that, SmartSpice automatically calculates Y, Z and H parameters, as well as various additional values. These values can be graphically displayed using rectangular, polar, or Smith charts.

You will not have to worry about which circuit to use when interested in specific parameters (such as H-parameters). SmartSpice is general enough to handle all possible cases automatically and provides accurate and reliable network parameter calculation.

The primary sweep for the .NET analysis uses the assigned .AC analysis and an optional secondary sweep can be added to see Monte-Carlo variation effects on the analysis output.

INPORT	Input port description in one of the following formats: • **V(n+, n-):** Two nodes (positive (n+) and negative (n-)) that serve as an input port. • **V(n+):** Two nodes (positive (n+) and negative (n-)) that serve as an input port, but with the negative node (n-) grounded. • **Ixxxx:** Input port is defined by a current source (used for DC biasing of that port). Current source Ixxxx must be defined in the input deck. The positive terminal becomes the positive input port node, and the negative terminal becomes the negative input port node. • **Vxxxx:** Input port is designated by voltage source (used for DC biasing of that port). Voltage source Vxxxx must be defined in the input deck. The positive terminal becomes the positive input port node, and the negative terminal becomes the negative input port node.
OUTPORT	Output port description in one of the following formats: • **V(n+, n-):** Two nodes (positive (n+) and negative (n-)) that serve as an output port • **V(n+):** Two nodes (positive (n+) and negative (n-)) that serve as an output port, but with the negative node (n-) grounded. • **Ixxxx:** Output port is defined by a current source (used for DC biasing of that port). Current source Ixxxx must be defined in the input deck. The positive terminal becomes the positive output port node, and the negative terminal becomes the negative output port node. • **Vxxxx:** Output port is defined by a voltage source (used for DC biasing of that port). Voltage source Vxxxx must be defined in the input deck. Its positive terminal becomes the positive output port node, and its negative terminal becomes the negative output port node.
DEC	Frequency sweep by decades.
LIN	Linear frequency sweep.
OCT	Frequency sweep by octaves.
nump	Number of points per decade or per octave, or number of points of a linear sweep.
fstart	Starting frequency.
fstop	Final frequency.

Optional parameters are summarized below:

Z0	Matching impedance. Default is 50Ω.
INDIN	Inductance through which the voltage DC source is connected to the input source (only if INPORT is given as Vxxxx)*.
RSIN	Series resistance of INDIN. Default is 1000Ω.
INDOUT	Inductance through which the voltage DC source is connected to the output port (only if OUTPORT is given as Vxxxx)*.
RSOUT	Series resistance of INDOUT. Default is 1000Ω.
RIN	Input or source resistance. Default is z0.
ROUT	Output or load resistance. Default is z0.
CIN	Capacitance through which the S-parameter test circuit is connected to the input port*.
COUT	Capacitance through which the S-parameter test circuit is connected to the output port*.

Note: Default values for the parameters marked (*) are calculated so that they do not disturb circuit behavior.

It is recommended that INDIN, INDOUT, RSIN, RSOUT, CIN and COUT remain unchanged.

sw2_spcf	The second (nested) sweep specification. This sweep is specified in one of the following ways:
	`SWEEP swname swstart swstop swincr`
	`SWEEP swname LIST nplist swval1 swval2 ...`
	`SWEEP swname swtype np swstart swstop`
	`SWEEP MODIF = dataname <PRTBL>`
	`<SWEEP> DATA=dataname`
	where:

swname	The name of the parameter to be swept. This parameter can be: • a parameter label defined in a global .PARAM statement; • a name of an independent voltage or current source; • a full-path name of a device parameter; • a full-path name of a model parameter; • the parameter TEMP (temperature sweep).
swstart	The starting value of the swept parameter.
swstop	The stop value of the swept parameter.
swstep	The increment value of the swept parameter.
swincr	The increment value for a linear sweep.

nplist	Number of parameter values swval1, swval2,... specified for a sweep of the type LIST.
swtype	The keyword that defines the type of sweep: DEC, OCT or LIN.
np	The number of points per decade or per octave, or the number of points for LIN sweep.
MODIF = dataname	This references the parametric .DATA statement.
DATA=dataname	This references the parametric .DATA statement.
PRTBL	This optional parameter flags the final table of all modified parameter labels and calculates .MEASURE results. Default is OFF.

Example

```
.PARAM SWEEP_VAR = 10
R       B   0   2.5K
C       B   0   1PF
RSIN    A   A1  1E-3
LIN     A1  A2  10
VIN     A2  0   DC 1.0 AC 0.1
IOUT    B   0   DC 0
R0 A    B   SWEEP_VAR
.NET V(A) IOUT DEC 10 1E3 1E9 DATA=SWEEP_DATA
.PRINT NET h11
.DATA SWEEP_DATA
+ SWEEP_VAR 10 20 30
.ENDDATA
.END
```

Examples of the .NET Statement

```
.NET V1 V2 DEC 10 1e6 1e10 z0=75
.NET I1 V2 DEC 10 1e6 1e10 z0=75
.NET V(1) V(2,3) DEC 10 1e6 1e10 z0=75
```

Output Results

After network analysis, the following results are available for post-processing:

```
S-parameters: S11, S12, S21, S22
H-parameters: H11, H12, H21, H22
Z-parameters: Z11, Z12, Z21, Z22
Y-parameters: Y11, Y12, Y21, Y22
```

Input impedance at port 1: $ZIN = -Z0\frac{S_{11}+1}{S_{11}-1}$

Output impedance at port 2: $ZOUT = -Z0\frac{S_{22}+1}{S_{22}-1}$

$D = S_{11}S_{22} - S_{21}S_{12}$: $K = \frac{1+|D|^2-|S_{11}|^2-|S_{22}|^2}{2|S_{22}S_{21}|}$

Transducer power gain in 50Ω system: $2G_t = |S_{21}|^2$

$$G_{ms} = \frac{|S_{21}|}{|S_{12}|}$$

$$GTU_{max} = \frac{G_t}{\left(1 - |S_{11}|^2\right)\left(1 - |S_{22}|^2\right)}$$

$$G_{amax} = \left\{ \begin{array}{l} \left|\frac{S_{21}}{S_{12}}\right|, \text{ when } k \le 1 \\ \left|\frac{S_{21}}{S_{12}}\left(k - \sqrt{k^2 - 1}\right)\right|, \text{ when } k > 1 \end{array} \right\}$$

S, H, Z and Y parameters are saved in rawfile.

Example Circuit

The following input deck is an example of network analysis for bipolar transistor S-parameter extraction.

```
BJT S-parameter extraction
*
vbe 1 0 dc 0.85
vce 2 0 dc 3.0
q1 2 1 0 test_model
*
.model test_model npn(is=1.0e-17 bf=80 nf=1.01
+ vaf=30.0 ikf=1.0e-3 ise=5e-16 ne=1.5
+ br=10.0 nr=1.0 var=10.0 ikr=0.02 isc=2.0e-16
+ nc=1.1
+ rb=100.0 irb=1.0e-3 rbm=1.0 re=2.0 rc=20.0
+ cje=0.1p vje=0.9 mje=0.42 fc=0.9
+ cjc=0.1p vjc=0.6 mjc=0.35 xcjc=0.5
+ cjs=0.1p vjs=0.55 mjs=0.48
+ tf=10p tr=600p
*
.net vbe vce dec 10 100Meghz 20Ghz z0=50
.save all
.end
```

Voltage sources vbe and vce are used for DC biasing of the transistor. These sources are used in .NET statement to designate input and output ports.

Hspice Compatibility

In -hspice mode, SmartSpice evaluates properly the original syntax of the obsolete Hspice `.NET` parameter analysis.

Syntax

One-Port Network:

```
.NET input <RIN=val>
```

After one-port network analysis, the following results are available for post-processing: `S11`, `ZIN` and `YIN` (`YIN=1/ZIN`).

Two-Port Network:

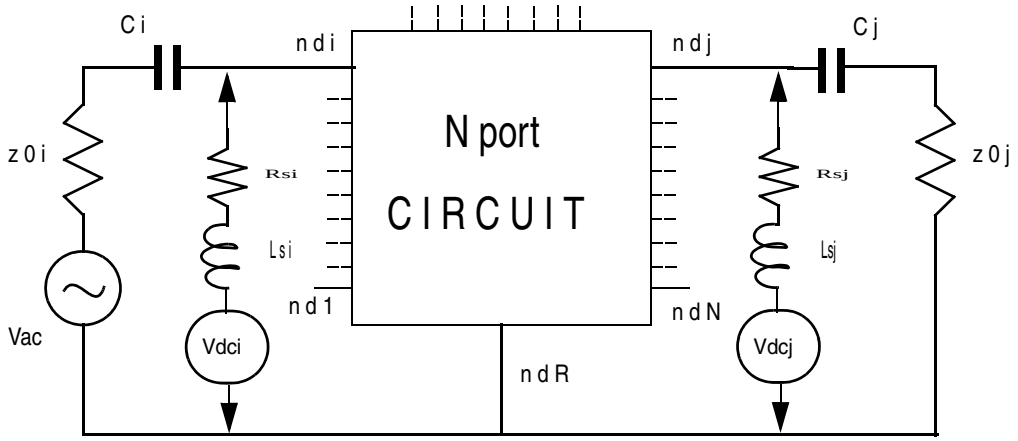
```
.NET output input <ROUT=val> <RIN=val>
```

.NET (Small-Signal Multi-Port Network Analysis)

Syntax

```
.NET N=val nd1 nd2 ... ndN ndR <DEC|LIN|OCT nump fstart fstop>
+ <z0=zval|(zval1<,zval2,...,zvalN>)>
+ <C=cval|(cval1<,cval2,...,cvalN>)>
+ <OUTTYPE=Y|S>
+ <OUTVALTYPE=CARTESIAN|CARTESIAN_POI| POLAR|POLAR_POI|POLARDEG|
+ POLARDEG_POI|DBPOLAR|DBPOLARDEG|DBPOLAR_POI|DBPOLARDEG_POI>
+ <OUTMATRIX=SYMMETRIC|HERMITIAN|NONSYMMETRIC|BLOCKSYMMETRIC>
+ <OUTMODE=MODEL|DATA>
+ <OUTFILE=filename>
+ <OUTNUMDGT=val>
```

This statement performs the extraction of scattering (S), impedance (Z) and admittance (Y) parameters for N terminal networks. This is an extension of the two-port network analysis described in the previous .NET statement to N ports, formed between N terminals and a reference terminal (ndR). When performing the network analysis, SmartSpice adds the measurement circuits (duplicating the actual bench technique for S-parameter measurement) to the specified circuit.



Active AC voltage source (Vac) through series resistor z0i, and capacitor Ci is connected to each *i*-terminal, and all remaining *j*-terminals (*j=1,.., N*; with $j \neq i$) are loaded by matching impedance z0j through series capacitor Cj. Such external measurement circuits are connected N times (*i=1,..., N*) at each frequency to extract a full set of S parameter matrices.

These measurement circuits do not disturb the DC operating conditions. SmartSpice also provides a feature for setting DC bias conditions for terminals in cases where the circuit does not have a built-in power supply, such as a single-device measurements. In this case, the DC sources (Vdci, Vdcj) connected through serial resistors (Rsi, Rsj), and serial inductors (Lsi, Lsj), will be used only for DC bias supply, and will not be taken into account during the network analysis.

The analysis output can be saved in a format to use directly as an S Element model, therefore helping to debug and simulate complex circuit chain events.

Note: The values of serial resistor Rs and inductor Ls can be specified in the corresponding independent voltage source statements. Default Rs and Ls values are calculated so that they do not disturb the circuit behavior.

The primary quantities extracted by SmartSpice are S-parameters, obtained for some value of matching impedance (typically 50 Ohm). After that, SmartSpice automatically calculates Y-parameters using the following formula:

$$Y = Z0^{-1/2} \cdot (I - S) \cdot (I + S)^{-1} \cdot Z0^{-1/2}$$

where *Z0* is the characteristic impedance matrix of the reference system (which is typically assumed to be diagonal, and its diagonal values are set to matching impedance *z0*). Similarly, Z-parameters are automatically calculated from S-parameters as follows:

$$Z = Z0^{1/2} \cdot (I - S)^{-1} \cdot (I + S) \cdot Z0^{1/2}$$

The automatic calculation of H-parameters is supported only for two-terminal cases. SmartSpice also automatically provides calculation of some useful parameters, as

i- terminal impedance: $ZI_i = z0_i \times \frac{S_{ii} + 1}{S_{ii} - 1}$;

$DS_{ij} = S_{ii} \times S_{jj} - S_{ij} \times S_j$, and $KD_{ij} = \frac{1 + \left|DS_{ij}\right|^2 - \left(\left|S_{ii}\right|^2 + \left|S_{jj}\right|^2\right)}{2\left|S_{ij} \times S_{ji}\right|}$

transducer power gain: $GT_{ij} = \left|S_{ji}\right|^2$, and $GMS_{ij} = \left|S_{ji}\right| / \left|S_{ij}\right|$;

maximum unilateral transducer power gain: $GTUMAX_{ij} = \frac{GT_{ij}}{\left|\left(1 - \left|S_{ii}\right|^2\right) \times \left(1 - \left|S_{jj}\right|^2\right)\right|}$;

maximum available power gain: $GAMAX_{ij} = \left|\frac{S_{ji}}{S_{ij}} \times \left(KD_{ij} - \sqrt{KD_{ij}^2 - 1}\right)\right|$, when $KD_{ij} > 1$

$GAMAX_{ij} = \left|S_{ji} / S_{ij}\right|$, when $KD_{ij} \le 1$,

where *i*-input terminal number, *j*-loaded terminal number (*j=1,.., N;* with $j \neq i$), *ij* - influence terminal (port) *j* on terminal (port) *i*.

.NET Statement Parameter Description

N=val	Number of network terminals (ports) without reference nodes.
nd1 nd2...ndN	N network terminals names.
ndR	Reference node name.
DEC	Frequency sweep by decades.
LIN	Linear frequency sweep.
OCT	Frequency sweep by octaves.
nump	Number of points per decade or per octave, or of points of a linear sweep.
fstart	Starting frequency.
fstop	Final frequency.

Note: Parameter N must be located after .NET and before terminal names.

If DEC|LIN|OCT nump fstart fstop parameter block is not specified, it is extracted from the .AC DEC|LIN|OCT nump fstart fstop statement.

Optional Measurement Circuit Parameters

z0=zval| (zval1<,zval2,...,zvalN>)	Matching impedance. Can be the same for all terminals if only one value is specified, and different for all terminals if all N values are specified. Default value is constant for all terminals, and equal to 50 Ohm.
C=cval| (cval1<,cval2,...,cvalN>)	Capacitance through which the S-parameter test circuit is connected to the terminal. Can be the same for all terminals if only one value is specified, and different for all terminals if all N values are specified. Default value is calculated so that it does not disturb circuit behavior. cval can be specified as =0.

Optional Output Control Parameters

These parameters allow you to save S- or Y-parameters in external file formats, which can be directly used in the S element (Multi-terminal network) model, specified by model parameter

DATAFILE, or printed in a SmartSpice output file format which can be directly copied to the S element .MODEL card (parameter DATA=(...) or DC=(...)):

OUTTYPE	Parameter type. • **Y:** Y-parameter. • **S:** Scattering parameter. If this parameter is not specified, S- or Y-parameter data points in S element format will not be saved.
OUTVALTYPE	Data type. • **CARTESIAN:** Complex number in real and imaginary format. • **CARTESIAN_POI:** Complex number in real and imaginary format. Data points are paired with nonuniform spacing frequency points. • **POLAR:** Complex number in polar format. Angles are written in radian. • **POLAR_POI:** Complex number in polar format. Angles are saved in radian. Data points are paired with nonuniform spacing frequency points. • **POLARDEG:** Complex number in polar format. Angles are saved in degrees. • **POLARDEG_POI:** Complex number in polar format. Angles are saved in degrees. Data points are paired with nonuniform spacing frequency points. • **DBPOLAR:** Complex number in polar format. Magnitudes are saved in decibel, and angles in radians. • **DBPOLARDEG:** Complex number in polar format. Magnitudes are saved in decibel, and angles in degrees. • **DBPOLAR_POI:** Complex number in polar format. Magnitudes are saved in decibel, and angles in radians. Data points are paired with nonuniform spacing frequency points. • **DBPOLARDEG_POI:** Complex number in polar format. Magnitudes are saved in decibel, and angles in degrees. Data points are paired with nonuniform spacing frequency points.
OUTMATRIX	Matrix (data point) format: • **SYMMETRIC:** Symmetric matrix. Only the lower-half triangle portion of matrix is saved. • **HERMITIAN:** Similar to SYMMETRIC, but off-diagonal terms are complex-conjugate of each other. Only the lower-half triangle portion of the matrix is saved. • **NONSYMMETRIC:** Nonsymmetric matrix. A full matrix is saved. • **BLOCKSYMMETRIC:** When the matrix can be represented as a four-block matrix (each block has size *N/2 x N/2*).

OUTMODE	File format. The parameter value MODEL changes the format and content of the created external file, which contains S- or Y-parameter matrices. When OUTMODE=MODEL the S- or Y-parameter matrices are written as .MODEL SP statement lines, they can be included in .MODEL input statement using .INCLUDE. The default OUTMODE=DATA saves results in the external file in the previous DATAFILE format.

Example

File sdata.par with DATAFILE format (OUTMODE=DATA):

```
* N=6 FSTART=1e+09 FSTOP=1e+10
* SPACING=LINEAR MATRIX=SYMMETRIC VALTYPE=CARTESIAN
*** S - parameters DATAFILE ***
*** ======================= ***
+ 10 $$ - number of data points
*** in 1 n 1 in 2
*** RealData ImaginaryData RealData
*** --- f[0]=1.000000e+09 ----------
3.548139090830e-02 1.681117508985e-03
4.657982349517e-05 4.070485598498e-03 3.547320948755e-02 ...
3.255123897577e-05 2.361486905465e-03 4.774625207240e-05 ...
9.644822616780e-01 -7.860329805485e-03 -2.632056547515e-05 ...
-2.504703481764e-05 -1.766573638517e-03 9.644644984382e-01 ...
-2.744819175282e-05 -1.879679982283e-03 -2.640582770474e-05 ...
*** --- f[1]=2.000000e+09 ----------
3.554542140701e-02 3.360250854423e-03
1.842940723429e-04 8.132855137570e-03 3.550913774506e-02 ...
1.293936542282e-04 4.721282832777e-03 1.845192935379e-04 ...
9.643114004049e-01 -1.571913862632e-02 -9.926995728703e-05 ...
-9.875707611075e-05 -3.523037411001e-03 9.642434342675e-01 ...
-1.097405545596e-04 -3.754516558774e-03 -9.972001067792e-05 ...
.........
```

can be used in S-element model as:

```
S1 1 3 5 2 4 6 0 FQMODEL=test_Quest TYPE=s
.model test_Quest SP
+ N=6
+ FSTART=1e9 FSTOP=1e10 NI=10 SPACING=lin
+ MATRIX=SYMMETRIC VALTYPE=CARTESIAN
+ DATAFILE=sdata.par
```

File smodel.par with MODEL line format (OUTMODE=MODEL):

```
*** S - parameters DATAFILE ***
+ N=6 FSTART=1e+09 FSTOP=1e+10
+ SPACING=LINEAR MATRIX=SYMMETRIC VALTYPE=CARTESIAN
+ TYPE = S
+ DATA=(
+ 10 $$ - number of data points
*** in 1 in 1 in 2
*** RealData ImaginaryData RealData
*** --- f[0]=1.000000e+09 ----------
+ 3.548139090830e-02 1.681117508985e-03
+ 4.657982349517e-05 4.070485598498e-03 3.547320948755e-02 ...
```

```
+ 3.255123897577e-05 2.361486905465e-03 4.774625207240e-05 ...
+ 9.644822616780e-01 -7.860329805485e-03 -2.632056547515e-05 ...
+ -2.504703481764e-05 -1.766573638517e-03 9.644644984382e-01 ...
+ -2.744819175282e-05 -1.879679982283e-03 -2.640582770474e-05 ...
*** --- f[1]=2.000000e+09 ----------
+ 3.554542140701e-02 3.360250854423e-03
+ 1.842940723429e-04 8.132855137570e-03 3.550913774506e-02 ...
+ 1.293936542282e-04 4.721282832777e-03 1.845192935379e-04 ...
+ 9.643114004049e-01 -1.571913862632e-02 -9.926995728703e-05 ...
+ -9.875707611075e-05 -3.523037411001e-03 9.642434342675e-01 ...
+ -1.097405545596e-04 -3.754516558774e-03 -9.972001067792e-05 ...
.........
+ )
```

can be used in S-element model as:

```
S1 1 3 5 2 4 6 0 FQMODEL=test_Quest
.model test_Quest SP
.include smodel.par
```

Block 1 Y11	Block 2 Y12
Block 3 Y21	Block 4 Y22

and block (sub-matrix) Y12=Y21, the matrix will be called “blocksymmetric”. This matrix can be stored in three blocks Y11, Y21, and Y22 array:

Block 1 Y11	
Block 3 Y21	Block 4 Y22

This type of matrix, as a rule, arises when the matrix is a multiconductor transmission line (W-element) Y-parameters matrix.

Note: For N=2, SYMMETRIC and BLOCKSYMMETRIC matrix is the same.

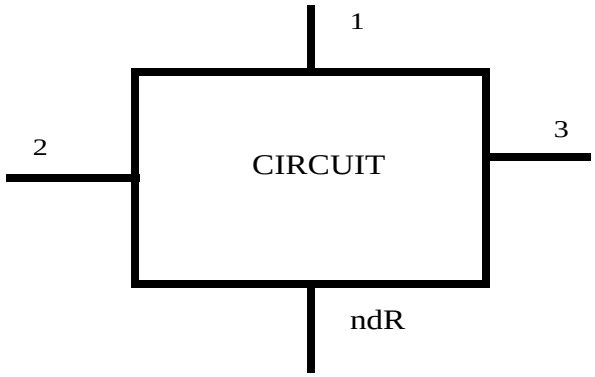
OUTFILE=filename	This option allows you to save S- or Y-parameter data points in an external file with name filename. If this parameter is not specified, the data points will be printed in SmartSpice standard output when the parameter OUTTYPE is specified.
OUTNUMDGT=val	Allows specification of the number of digits (after the point) in the output S- or Y- parameter data points. Default is 15.

S,Y-parameter Outfile Format

S- and Y-parameter outfiles contains:

- Additional information, starting from symbol * and is considered by the SmartSpice parser as a comment line. This additional commented data describes the matrix: size (N), structure (MATRIX=SYMMETRIC|NONSYMMETRIC|HERMITIAN|BLOCKSYMMETRIC), type of matrix elements (VALTYPE=CARTESIAN|POLAR|...), and frequency point distribution: frequency start (FSTART), frequency stop (FSTOP), type of frequency sweep (SPACING=DEC|LIN|OCT).
- Number of frequency points.
- List of S- or Y-parameter matrixes at frequency points.

Locations of S- or Y-parameters of each terminal in the matrix are summarized below (for N=3, MATRIX=NONSYMMETRIC, VALTYPE=CARTESIAN and TYPE=S):




```
.NET  N=3  2  1  3  ndR ...
```

	column:1	column:2	column:3	column:4	column:5	column:6
row:1	reS(2,2)	imS(2,2)	reS(2,1)	imS(2,1)	reS(2,3)	imS(2.3)
row:2	reS(1,2)	imS(1,2)	reS(1,1)	imS(1,1)	reS(1,3)	imS(1.3)
row:3	reS(3,2)	imS(3,2)	reS(3,1)	imS(3,1)	reS(3,3)	imS(3.3)

Location of the terminal parameters in the matrix corresponds to the position of the terminal name in the terminal list in .NET statement.

Note: Numbers in S(...,...) are not the row and column numbers, they are the node names.

S,Y-parameter Output Format for S Element Model

When you run network analysis to create an S element model with the parameter DATA=(npts d1 d2 ...), the .NET statement does not contain the parameter OUTFILE, and SmartSpice will print S element model card lines for model parameters N, FSTART, FSTOP, SPACING, MATRIX, VALTYPE and DATA into a standard output. After running, these text lines can be directly inserted in the statement:

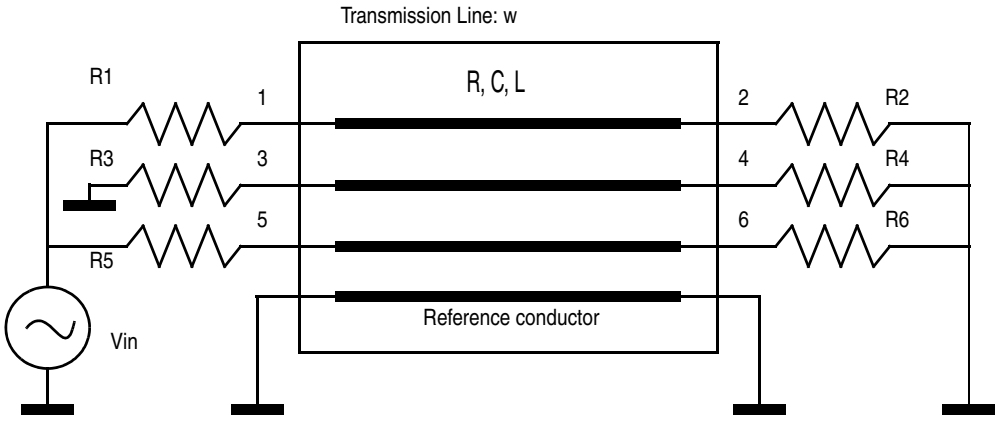
```
.MODEL  modname  SP
```

For the parameter DC=(...), you must run network analysis at frequency=0, using the statement:

```
.NET  N=val  nd1 nd2 ... ndN ndR  LIN 1 0 0
+ OUTTYPE=Y|S
+ <OUTVALTYPE=CARTESIAN|CARTESIAN_POI|POLAR|POLAR_POI|POLARDEG|
+ POLARDEG_POI|DBPOLAR|DBPOLARDEG|DBPOLAR_POI|DBPOLARDEG_POI>
+ <OUTMATRIX=SYMMETRIC|HERMITIAN|NONSYMMETRIC|BLOCKSYMMETRIC>
```

which will print text that contains S or Y matrix at frequency=0. These text lines can be directly inserted in the S model statement as well.

Example 1




```
* Example 1: Three Coupled Transmission Lines
* ------------------------------------------------
* from:
* D.B. Kuznetsov, J.E. Schutt-Aine,
* "Optimal transient Simulation of Transmission Lines",
* IEEE Transactions on Circuits and Systems - I:
* Fundamental Theory and Applications,
* vol. 43, No.2, 1996, pp. 110-121.
* ------------------------------------------------
.options Nomod

vin In 0 ac 1 DC 0
```

```
w1 n=3 1 3 5 0 2 4 6 0
+     umodel = IcWire  l=0.677

.MODEL IcWire W plev=1 NL=3 ELEV=2
+      Cr1=94p  C12=22p C13=22p
+               Cr2=94p C23=22p
+                       Cr3=94p

+      L11=418n  L12=125n  L13=125n
+                L22=418n  L23=125n
+                          L33=418n

+      R11=15 R22=15 R33=15

r1 In 1 50
r3 0 3 10meg
r5 In 5 50

r2 2 0 1k
r4 4 0 10meg
r6 6 0 1k

.AC LIN 100 100Meg 10G

.PROBE AC v(2) v(3) v(4) v(5) v(6)
.end
```

To extract Y-parameters for transmission line w1, the input deck will be as following:

```
  *  Example  1.1:  Y  parameter  extracting  for  Three  Coupled
Transmission Lines

.options Nomod dcgnode=1.e-15

w1 n=3 1 3 5 0 2 4 6 0   umodel = IcWire  l=0.677

.MODEL IcWire W plev=1 NL=3 ELEV=2
+      Cr1=94p  C12=22p C13=22p
+               Cr2=94p C23=22p
+                       Cr3=94p

+      L11=418n  L12=125n  L13=125n
+                L22=418n  L23=125n
+                          L33=418n

+      R11=15 R22=15 R33=15

*** NET analysis at f=0 for DC data point
***     with print in standard output

.NET N=6 5 3 1 2 4 6 0 LIN 1 0 0
+  OUTtype=Y OUTvalType=CARTESIAN OUTMatrix=SYMMETRIC

*** NET analysis with saving in file

.NET N=6 5 3 1 2 4 6 0 LIN 100 100Meg 10G
+  OUTfile=w1_ycart_sym.par
```

```
+  OUTtype=Y OUTvalType=CARTESIAN OUTMatrix=SYMMETRIC

.end
```

Data points are saved in file by statements:

```
.NET N=6 5 3 1 2 4 6 0 LIN 100 100Meg 10G
+  OUTfile=w1_ycart_sym.par
+  OUTtype=Y OUTvalType=CARTESIAN OUTMatrix=SYMMETRIC
```

are presented (only 1 column of each complex matrix, and only for first four frequencies) below:

```
*  N=6  FSTART=1e+08  FSTOP=1e+10
*  SPACING=LINEAR  MATRIX=SYMMETRIC  VALTYPE=CARTESIAN
***    *** Y - parameters DATAFILE ***
 100  $$$ - number of data points
***        RealData             ImaginaryData          ...
*** --- f[0]=1.000000e+08 ----------
    3.124280037913846e-02   3.807573944434364e-02     ...
   -8.762571488499651e-03  -4.262693066245417e-02     ...
   -8.686392999335567e-03  -4.258945205857109e-02     ...
   -8.374198332972770e-03  -4.394725950044670e-02     ...
   -8.374198332961132e-03  -4.394725950043575e-02     ...
    3.012771053275088e-02   3.934398794544804e-02     ...
*** --- f[1]=2.000000e+08 ----------
    9.600097590046522e-03   2.184162055342543e-02     ...
   -2.904792971476534e-03  -2.270232218348971e-02     ...
   -2.905894460224350e-03  -2.269422401750084e-02     ...
    2.500368582015801e-03   2.541921658697185e-02     ...
    2.500368582017091e-03   2.541921658697456e-02     ...
   -8.438340996008539e-03  -2.450706679215652e-02     ...
*** --- f[2]=3.000000e+08 ----------
    4.854573795175520e-03   1.364884667833906e-02     ...
   -1.505368565247766e-03  -1.397742154748597e-02     ...
   -1.496653202146515e-03  -1.397123864755817e-02     ...
   -1.074575541805148e-03  -1.815755681472123e-02     ...
   -1.074575541805350e-03  -1.815755681472184e-02     ...
    3.614296190288036e-03   1.776496740377678e-02     ...
*** --- f[3]=4.000000e+08 ----------
    3.173615090847965e-03   8.715615349774808e-03     ...
   -1.002069204647519e-03  -8.877698516035197e-03     ...
   -1.003232460199350e-03  -8.873636286244576e-03     ...
    5.304763372361797e-04   1.464291247683102e-02     ...
    5.304763372351252e-04   1.464291247682730e-02     ...
   -1.811436083898202e-03  -1.439888343526939e-02     ...
*** --- f[4]=5.000000e+08 ----------
       2.448489788784600e-03     5.180948125816843e-03         ...
...
```

Data points are printed in output (for using in S element model card) by the statements:

```
.NET N=6 5 3 1 2 4 6 0 LIN 1 0 0
+  OUTtype=Y OUTvalType=CARTESIAN OutMatrix=SYMMETRIC
```

and are presented (only 1 column of complex matrix) below:

```
***    *** Y - DC parameters   MATRIX=SYMMETRIC  VALTYPE=CARTESIAN
***
+  DC=(
***         RealData              ImaginaryData      ...
+   +9.847365929640578e-02   0.000000000000000e+00 ...
+    0.000000000000000e+00   0.000000000000000e+00 ...
+    0.000000000000000e+00   0.000000000000000e+00 ...
+    0.000000000000000e+00   0.000000000000000e+00 ...
+    0.000000000000000e+00   0.000000000000000e+00 ...
+   -9.847365729640575e-02   0.000000000000000e+00 ...
+ )
```

Data points are printed in output (for using in S element model card) by the statements:

```
.NET N=6 5 3 1 2 4 6 0 LIN 100 100Meg 10G
+  OUTtype=Y OUTvalType=CARTESIAN OutMatrix=SYMMETRIC
```

and are presented (only 1 column of each complex matrix and only for the first four frequencies) below:

```
+  N=6  FSTART=1e+08  FSTOP=1e+10
+  SPACING=LINEAR  MATRIX=SYMMETRIC  VALTYPE=CARTESIAN
+  DATA=(
***    *** Y - parameters DATAFILE ***
+  100  $$$ - number of data points
***         RealData              ImaginaryData          ...
*** --- f[0]=1.000000e+08 ----------
+    3.124280037913846e-02   3.807573944434364e-02     ...
+   -8.762571488499651e-03  -4.262693066245417e-02     ...
+   -8.686392999335567e-03  -4.258945205857109e-02     ...
+   -8.374198332972770e-03  -4.394725950044670e-02     ...
+   -8.374198332961132e-03  -4.394725950043575e-02     ...
+    3.012771053275088e-02   3.934398794544804e-02     ...
*** --- f[1]=2.000000e+08 ----------
+    9.600097590046522e-03   2.184162055342543e-02     ...
+   -2.904792971476534e-03  -2.270232218348971e-02     ...
+   -2.905894460224350e-03  -2.269422401750084e-02     ...
+    2.500368582015801e-03   2.541921658697185e-02     ...
+    2.500368582017091e-03   2.541921658697456e-02     ...
+   -8.438340996008539e-03  -2.450706679215652e-02     ...
*** --- f[2]=3.000000e+08 ----------
+    4.854573795175520e-03   1.364884667833906e-02     ...
+   -1.505368565247766e-03  -1.397742154748597e-02     ...
+   -1.496653202146515e-03  -1.397123864755817e-02     ...
+   -1.074575541805148e-03  -1.815755681472123e-02     ...
+   -1.074575541805350e-03  -1.815755681472184e-02     ...
+    3.614296190288036e-03   1.776496740377678e-02     ...
*** --- f[3]=4.000000e+08 ----------
+    3.173615090847965e-03   8.715615349774808e-03     ...
+   -1.002069204647519e-03  -8.877698516035197e-03     ...
+   -1.003232460199350e-03  -8.873636286244576e-03     ...
+    5.304763372361797e-04   1.464291247683102e-02     ...
+    5.304763372351252e-04   1.464291247682730e-02     ...
+   -1.811436083898202e-03  -1.439888343526939e-02     ...
*** --- f[4]=5.000000e+08 ----------
+       2.448489788784600e-03     5.180948125816843e-03        ...
...
+ )
```

Example 2

This example illustrates the calculation of S-parameters for three coupled conductor transmission lines, described by the frequency tabled RLGC file (`diap6g.rlc`) extracted by the tool QUEST.

```
*Test: 3 couple lines l=0.0001
.OPTIONS NOMOD

W1 n=3 1 3 5 0 2 4 6 0 RLGCfile=diap6g.rlc l=0.0001
+ rlgccheck=0

.NET N=6 5 3 1 2 4 6 0
+    LIN 401 0 10gHz
+    OutType=S
+    OutValType=POLAR
+    OutMatrix=NONSYMMETRIC
+    OutFile=QW1_Spolar_nonsym.par

.end
```

where `diap6g.rlc` file, which was created by QUEST in the W element `RLGCfile`, contains the following data:

```
* RLCG parameters for 3 signal conductors.
* lossy transmission line
*******************************************************
* N - number of signal conductors  NS - number of samples
*******************************************************
3        6
*******************************************************
* 1 - frequency (Hz)
1e+07
***************************************
* L - inductance (H/m)
***************************************
9.98485e-07
5.31687e-07   9.99572e-07
3.94458e-07   5.31687e-07   9.98485e-07
***************************************
* C - capacitance (F/m)
***************************************
8.51607e-11
-1.74082e-11   9.12922e-11
-1.65572e-12   -1.74083e-11   8.51607e-11
***************************************
* R - resistance (Ohm/m)
***************************************
33333.3
0.000478106   33333.3
0.000434762   0.000478106   33333.3
***************************************
* G - conductance (Ohm-1/m)
***************************************
7.00001e-08
-3.4e-08   6.9e-08
-3.6e-08   -3.4e-08   7.1e-08
*************************************************
```

```
* 2 - frequency (Hz)
4.31845e+09
**************************************
* L - inductance (H/m)
**************************************
9.98414e-07
5.31637e-07   9.99495e-07
3.94418e-07   5.31637e-07   9.98414e-07
**************************************
* C - capacitance (F/m)
**************************************
8.51565e-11
-1.74119e-11   9.12952e-11
-1.66022e-12   -1.74119e-11   8.51565e-11
**************************************
* R - resistance (Ohm/m)
**************************************
33447.5
88.7017   33450
82.1067   88.7017   33447.5
**************************************
* G - conductance (Ohm-1/m)
**************************************
0.0086818
-0.0064504   0.00605166
-0.00671612   -0.00645043   0.00868176
******************************************************
* 3 - frequency (Hz)
9.69906e+09
**************************************
* L - inductance (H/m)
**************************************
9.98126e-07
5.31438e-07   9.9919e-07
3.94255e-07   5.31438e-07   9.98126e-07
**************************************
* C - capacitance (F/m)
**************************************
8.51387e-11
-1.74259e-11   9.12767e-11
-1.67672e-12   -1.74259e-11   8.51387e-11
**************************************
* R - resistance (Ohm/m)
**************************************
33908.2
446.959   33920.8
413.909   446.959   33908.2
**************************************
* G - conductance (Ohm-1/m)
**************************************
0.0437563
-0.0328728   0.0298896
-0.0338437   -0.0328727   0.0437562
******************************************************
* 4 - frequency (Hz)
1.68687e+10
```

```
**************************************
* L - inductance (H/m)
**************************************
9.97401e-07
5.30936e-07   9.98423e-07
3.93838e-07   5.30936e-07   9.97401e-07
**************************************
* C - capacitance (F/m)
**************************************
8.50921e-11
-1.74627e-11   9.12491e-11
-1.71775e-12   -1.74626e-11   8.50921e-11
**************************************
* R - resistance (Ohm/m)
**************************************
35065.7
1348.42   35103
1250.29   1348.42   35065.7
**************************************
* G - conductance (Ohm-1/m)
**************************************
0.132095
-0.0980944   0.0920242
-0.102114   -0.0980949   0.132094
******************************************************
* 5 - frequency (Hz)
2.76466e+10
**************************************
* L - inductance (H/m)
**************************************
9.95625e-07
5.29703e-07   9.96547e-07
3.92804e-07   5.29703e-07   9.95625e-07
**************************************
* C - capacitance (F/m)
**************************************
8.49803e-11
-1.75567e-11   9.11665e-11
-1.8216e-12   -1.75571e-11   8.49803e-11
**************************************
* R - resistance (Ohm/m)
**************************************
37946.1
3597.81   38038.8
3346.59   3597.81   37946.1
**************************************
* G - conductance (Ohm-1/m)
**************************************
0.352989
-0.261929   0.245965
-0.272498   -0.261931   0.352987
************************************************
* 6 - frequency (Hz)
5e+10
**************************************
* L - inductance (H/m)
```

```
**************************************
9.89546e-07
5.25393e-07   9.90189e-07
3.89037e-07   5.25393e-07   9.89546e-07
**************************************
* C - capacitance (F/m)
**************************************
8.45859e-11
-1.78832e-11   9.08754e-11
-2.18697e-12   -1.78835e-11   8.45859e-11
**************************************
* R - resistance (Ohm/m)
**************************************
47967.5
11491.2   48223.7
10766.5   11491.2   47967.5
**************************************
* G - conductance (Ohm-1/m)
**************************************
1.13347
-0.838758   0.788976
-0.870707   -0.838759   1.13347
```

Magnitude of some calculated S-parameters (from example 2): s(1,5), s(1,3), s(1,4), s(1,6), s(3,5), s(3,1), s(3,2), s(3,6); and s(5,3), s(5,1), s(5,2), s(5,4) are presented below:

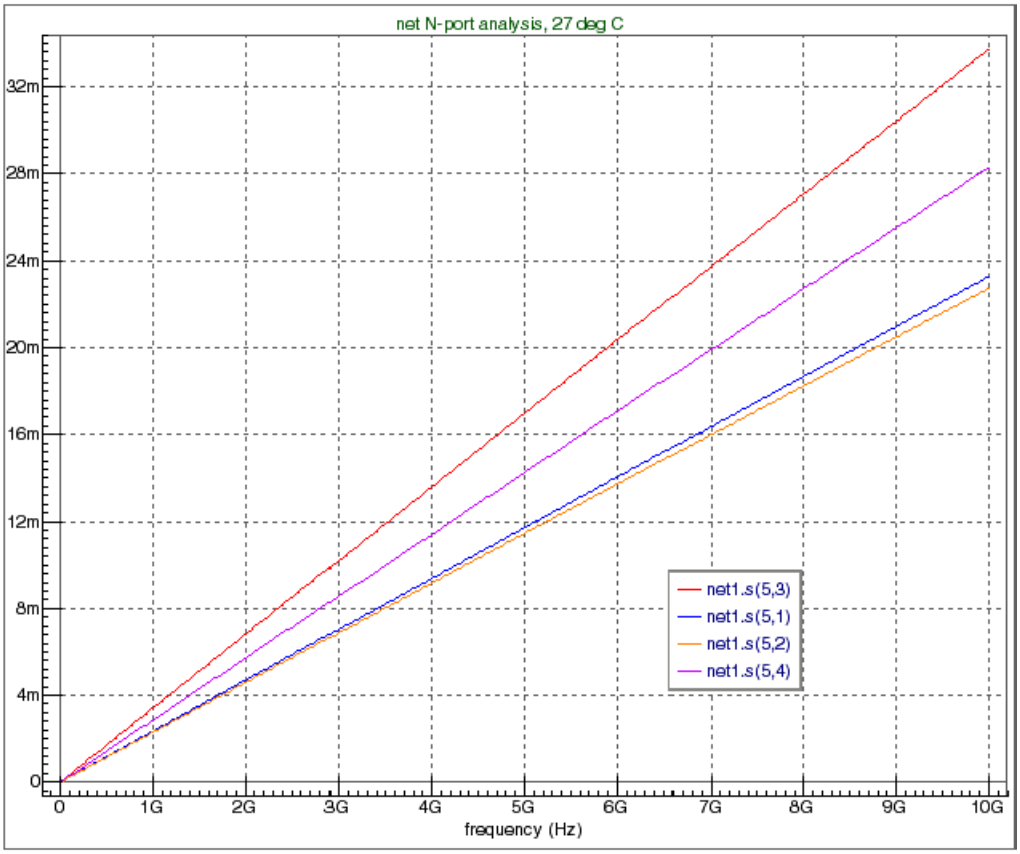


Figure 3-8 Calculated S-parameters (1)

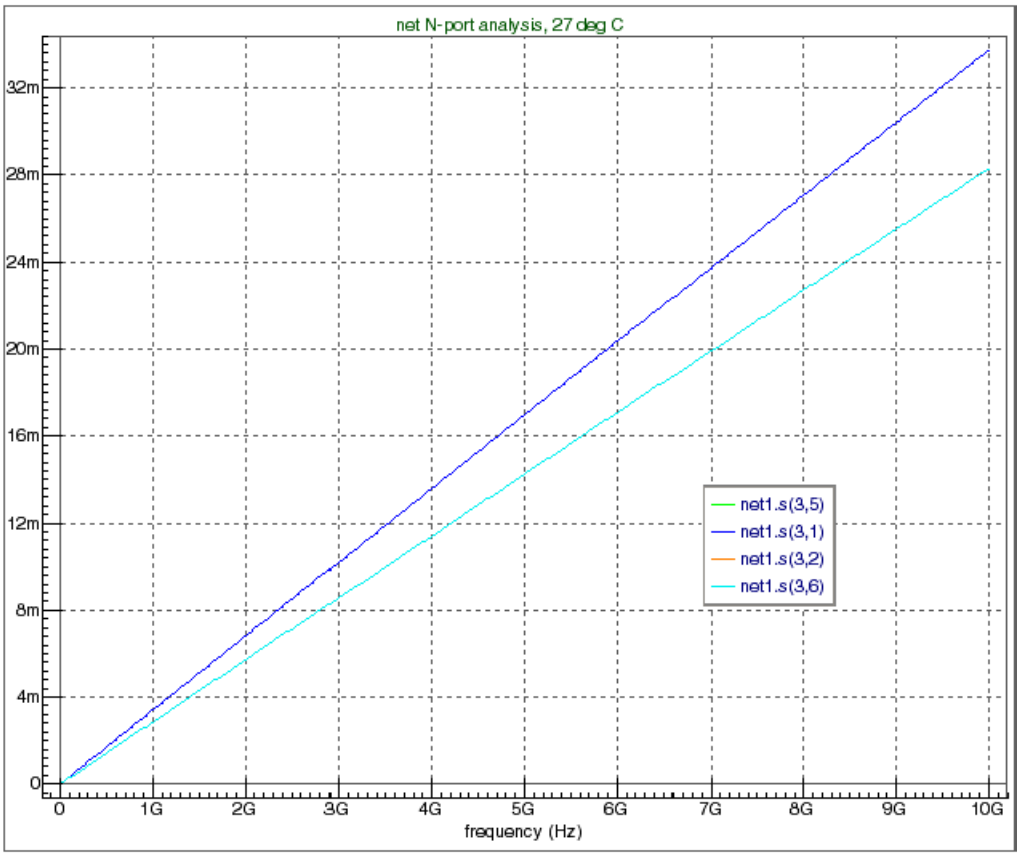


Figure 3-9 Calculated S-parameters (2)

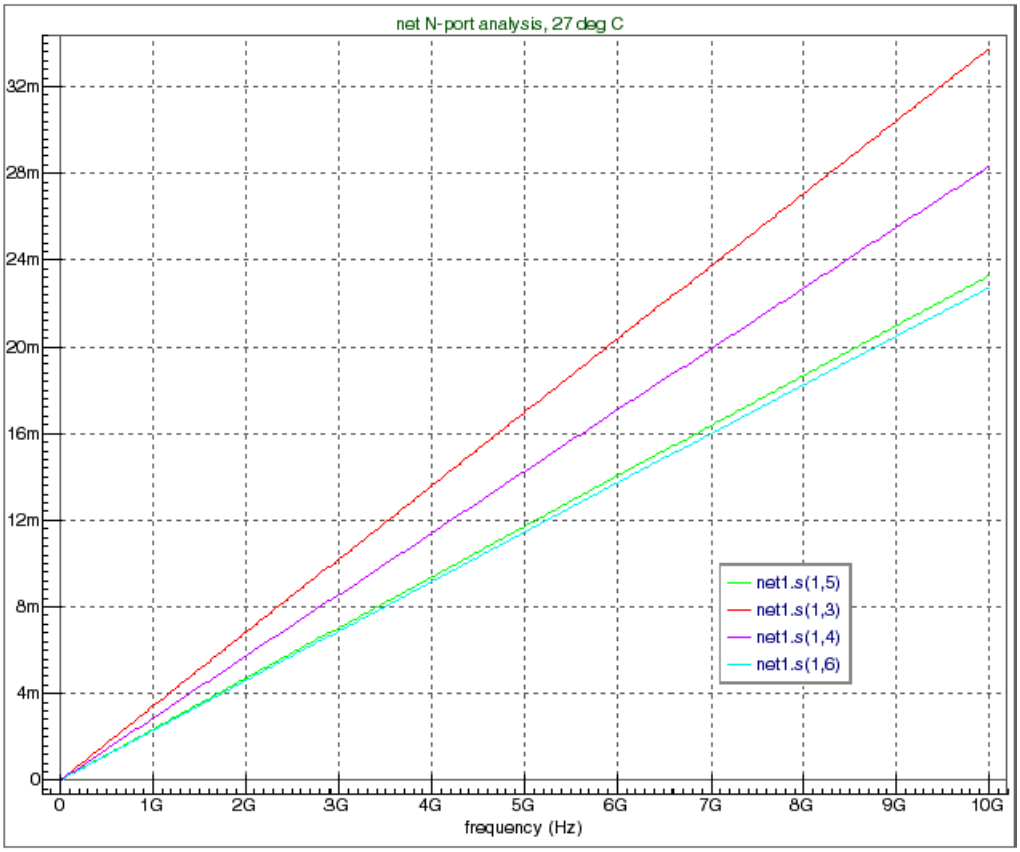


Figure 3-10 Calculated S-parameters (3)

Output Results

As a result of multi-terminal network analysis, SmartSpice calculates scattering (S), impedance (Z), admittance (Y), hybrid (H), terminal impedance (ZI), inverse Linvill C factor (KD), transducer power gain (Gt), maximum unilateral transducer power gain (GTUmax), maximum available power gain (Gamax), and so forth. The SmartSpice statements .PRINT, .PLOT, .PROBE and .MEASURE can be used to print, plot, save in rawfile and measure these parameters.

S, Y, Z and H-parameters are complex (*N x N*) matrices at set frequency points, where *N* is the number of signal terminals.

The output of all elements of these matrices is specified by the syntax:

```
S(ALL), Y(ALL), Z(ALL), H(ALL)
```

The output or separate measure matrix element is specified by the syntax:

```
s(nodename1,nodenam2),                    y(nodename1,nodenam2),
z(nodename1,nodenam2), h(nodename1,nodenam2)
```

off-diagonal matrix elements, which describes the dependence of the parameters s, z, y, h on the terminals nodename1 and nodename2;

```
s(nodename,nodename)  or  s(nodename),  y(nodename,nodename)  or
y(nodename),
z(nodename,nodename)  or  z(nodename),  h(nodename,nodename)  or
h(nodename)
```

diagonal matrix elements corresponding to terminal nodename.

These complex parameters are supported in the following predefined macros:

real part of complex vector, formed by matrix element at the set frequency points:

```
sr(nodename1,nodenam2) = re(s(nodename1,nodenam2));
sr(nodename) = re(s(nodename,nodenam));

yr(nodename1,nodenam2) = re(y(nodename1,nodenam2));
yr(nodename) = re(y(nodename,nodenam));
zr(nodename1,nodenam2) = re(z(nodename1,nodenam2));
zr(nodename) = re(z(nodename,nodenam));

hr(nodename1,nodenam2) = re(h(nodename1,nodenam2));
hr(nodename) = re(h(nodename,nodenam));
```

imaginary part of complex vector, formed by the matrix element at set the frequency points:

```
si(nodename1,nodenam2) = im(s(nodename1,nodenam2));
si(nodename) = im(s(nodename,nodenam));

yi(nodename1,nodenam2) = im(y(nodename1,nodenam2));
yi(nodename) = im(y(nodename,nodenam));

zim(nodename1,nodenam2) = im(z(nodename1,nodenam2));
zim(nodename) = im(z(nodename,nodenam));

hi(nodename1,nodenam2) = im(h(nodename1,nodenam2));
hi(nodename) = im(h(nodename,nodenam));
```

magnitude of complex vector, formed by the matrix element at the set frequency points:

```
sm(nodename1,nodenam2) = mag(s(nodename1,nodenam2));
sm(nodename) = mag(s(nodename,nodenam));

ym(nodename1,nodenam2) = mag(y(nodename1,nodenam2));
ym(nodename) = mag(y(nodename,nodenam));

zm(nodename1,nodenam2) = mag(z(nodename1,nodenam2));
zm(nodename) = re(z(nodename,nodenam));

hm(nodename1,nodenam2) = mag(h(nodename1,nodenam2));
hm(nodename) = mag(h(nodename,nodenam));
```

magnitude in decibels of complex vector, formed by the matrix element at the set frequency points:

```
sdb(nodename1,nodenam2) = db(s(nodename1,nodenam2));
sdb(nodename) = db(s(nodename,nodenam));

ydb(nodename1,nodenam2) = db(y(nodename1,nodenam2));
ydb(nodename) = db(y(nodename,nodenam));

zdb(nodename1,nodenam2) = db(z(nodename1,nodenam2));
zdb(nodename) = db(z(nodename,nodenam));

hdb(nodename1,nodenam2) = db(h(nodename1,nodenam2));
hdb(nodename) = db(h(nodename,nodenam));
```

phase in radians of complex vector, formed by the matrix element at the set frequency points:

```
sp(nodename1,nodenam2) = ph(s(nodename1,nodenam2));
sp(nodename) = ph(s(nodename,nodenam));
yp(nodename1,nodenam2) = ph(y(nodename1,nodenam2));
yp(nodename) = ph(y(nodename,nodenam));

zp(nodename1,nodenam2) = ph(z(nodename1,nodenam2));
zp(nodename) = ph(z(nodename,nodenam));
hp(nodename1,nodenam2) = ph(h(nodename1,nodenam2));
hp(nodename) = ph(h(nodename,nodenam));
```

group delay (*d(phase)/df*) of complex vector, formed by the matrix element at the set frequency points:

```
sgd(nodename1,nodenam2) = gdelay(s(nodename1,nodenam2));
sgd(nodename) = gdelay(s(nodename,nodenam));

ygd(nodename1,nodenam2) = gdelay(y(nodename1,nodenam2));
ygd(nodename) = gdelay(y(nodename,nodenam));

zgd(nodename1,nodenam2) = gdelay(z(nodename1,nodenam2));
zgd(nodename) = gdelay(z(nodename,nodenam));

hgd(nodename1,nodenam2) = gdelay(h(nodename1,nodenam2));
hgd(nodename) = gdelay(h(nodename,nodenam));
```

Output Example 1

```
.PRINT s(all) yr(in), yr(in,out) yr(out,in) yr(out,out)
```

This statement causes the printing of all S-parameters, and prints the real parts of Y-parameters, which describe terminals with the names `in`, `out` and the interaction between them.

Terminal impedance `ZI` is the N complex vectors at a set of frequency points.

The output of all vectors is specified by the syntax:

```
ZI(ALL)
```

The output of separate vectors to be measured is specified by the syntax:

```
zi(nodename)
```

Parameters `KD`, `GT`, `GMS`, `GTUMAX` and `GAMAX` are the *N x (N-1)* real vectors at a set of frequency points, each (which describe the interaction terminals) excludes self-interaction cases (`nodename1` cannot be the same as `nodename2`).

The output of all elements of these vectors is specified by the syntax:

```
KD(ALL), GT(ALL), GMS(ALL), GTUMAX(ALL), GAMAX(ALL)
```

The output or separate measure vector is specified by the syntax:

```
DS(nodename1,nodename2),                    GT(nodename1,nodename2),
GMS(nodename1,nodename2),
GTUMAX(nodename1,nodename2), GAMAX(nodename1,nodename2)
```

Output Example 2

```
.PROBE   NET   ZI(ALL)   Gt(in,out)   Gms(in,out)   GTUmax(in,out)
Gamax(in,out)
```

This statement saves in rawfile all `ZI` parameters, and the parameters `GT`, `GMS`, `GTUMAX` and `GAMAX` for terminals with names `in` and `out`.

.NODESET (Set Node Voltages)

Syntax

```
.NODESET V(node1)=val1 <V(node2)=val2 ...>
```

This statement helps the program find the DC or initial transient solution by making a preliminary pass with the specified nodes held at the given voltages. The restriction is then relaxed, and the iteration continues to the true solution. The .NODESET statement may be necessary for convergence on bistable or astable circuits. In most circumstances, this statement will not be necessary.

If a .NODESET statement is specified in a subcircuit definition, SmartSpice expands the nodenames that appear in the statement during subcircuit expansion.

The .NODSET statement supports templates and bus notations.

Note: If two .NODESET statements refer to the same node, the second .NODESET overwrites the first.

Examples

```
.NODESET V(6)=5.0 V(3)=0.2
.NODESET V(12)=4.0 V(4)=5.5
.SUBCKT SUB1 1 3
RSUB1 1 2 100
RSUB2 2 3 200
.NODESET V(2)=2.5
.ENDS
VIN 2 0 PULSE(0 5 1N 1N 2N 10N)
R1 2 10 1K
XSUB 10 30 SUB1
R2 30 0 10K
.NODESET V(XSUB.2)=3.5
```

In this example, two .NODESET statements refer to node 2 inside the subcircuit XSUB. The second .NODESET statement overwrites the first one.

```
.nodeset v(bus<1:2>)=0.3
.nodeset v(node*)=0.5
```

In the example above, the first .NODESET statement sets the 2 bus wires (bus1, bus2) to 0.3V. The second .NODESET statement sets all nodes with the prefix node to 0.5V.

.NOISE (Noise Analysis)

Syntax

```
.NOISE V(node1 <,node2>) src <interval|<INTER=>val>
+ <DEC|DEC_ENG|LIN|OCT nump fstart fstop>
+ <DEVCON=<val>> <RX<=val>>
+ <CALLV> <SAVEV> <UIC> <IC> <PRINTOP>
+ <<SWEEP> MONTE=val <PRMC>>
```

or

```
.NOISE V(node1 <,node2>) src <interval|<INTER=>val>
+ <FREQLIST|POI numsteps freq1 <freq2 <... <freqN>>>>
+ <DEVCON=<val>> <RX<=val>>
+ <CALLV> <SAVEV> <UIC> <IC> <PRINTOP>
+ <<SWEEP> MONTE=val <PRMC>>
```

This statement performs a noise analysis of the circuit. The analysis calculates noise spectral density at the output port due to noise sources of each device in the circuit, and the equivalent input noise. The noise contribution of a particular noise source to the output is calculated as:

```
(Spectral Noise Density) * RX,
```

where RX is a transfer function coefficient between the node where the noise source is situated, and the node of the defined output port.

All this is done for every point in a specified frequency range. After calculating the spectral densities, these values are integrated over the specified frequency range to output total integrated noise.

V(node1 <, node2>)	Output voltage that defines the port where total output noise is calculated. By default, node2 is assumed to be ground.
src	The name of an independent voltage or current source to which input noise is referred. src is not a noise generator; it only defines the point where the equivalent input noise is calculated.
interval|INTER<=val>	The optional integer parameter if specified: • the total output noise, • the noise contribution of each noise generator, • and if an additional RX<=val> parameter is specified, a transfer function coefficient for noise sources to the output, depending on val, are printed out every interval frequency point. If you omit interval or set it to 0 SmartSpice doesn't print a summary.
RX	If an additional RX<=val> parameter is specified, a transfer function coefficient for noise sources to the output, depending on val, are printed out every interval frequency point.
DEC	Sweep by decades.

DEC_ENG	Sweep by decades. This parameter causes the following frequency scale to be formed: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, ... and so on. Only 10 points per decade (parameter `nump`) should be defined.
OCT	Sweep by octaves.
LIN	Linear sweep.
nump	Number of points per decade or per octave, or the number of points in a linear sweep.
fstart	The starting frequency.
fstop	The final frequency.
FREQLIST|POI	Indicates the list of frequencies to be performed by Noise Analysis. It is possible to pass 0 (zero) as `numsteps` value, which has a meaning of `<read all points>`.
numsteps	Number of points of interest (frequency points). When set to 0 all points in the list will be used.

If `<DEC|DEC_ENG|LIN|OCT nump fstart fstop>` or `<FREQLIST|POI numsteps freq1 <freq2 <... <freqN>>>>` are not specified, they are extracted from `.AC` statement.

CALLV	Calls a previously saved operating point before starting the DC operating point calculation.
SAVEV	Saves the calculated DC operating point.
UIC	Causes SmartSpice to bypass the DC operating point calculation and to create a linearized model at the called operating point. In this case, AC analysis is free from any “No Path to Ground” problems.
IC	Causes SmartSpice to use `.IC` statements when computing the operating point.
PRINTOP	Prints OP information about computing the operating point or the called operating point (if `UIC` is specified).

The `.NOISE` statement produces two plots:

- one for the noise spectral density curves,
- and one for the total integrated noise over the specified frequency range.

All output noise voltages/currents (`onoise`) are in squared units (V^2/Hz and A^2/Hz for spectral density; V^2 and A^2 for integrated noise). Input noise voltages/currents (`inoise`) are normalized on a square magnitude of the independent voltage/current source.

Examples

```
.NOISE V(5) VCC DEC 10 1 10G
.NOISE V(5,3) V1 OCT 8 1.0 1.0E6 1 RX=3
.NOISE  v(2) vin
.AC DEC 10 1 1K
```

DEVcon<=val>	At every frequency point, SmartSpice outputs a table of the val, most noisy devices in the circuit and their names, the device's total noise, and percentage contribution to the total output noise onoise_s. Only devices with a contribution of more than 0.01% are taken into account.

If the parameter DEVcon has no value, the output table consists of only one noisy device (by default, val=1).

To output contributions of all noisy devices in the circuit negative parameter value (val<0) should be defined.

If DEVCON omitted and the INTER parameter is specified with a non zero value, DEVCON will be initialized with a default negative value.

Examples

```
.NOISE V(OUT) VINPUT DEC 10 1 100MegHz 1
+ DEVcon = 3
```

The output table will consist of the 3 most noisy devices.

```
.NOISE V(OUT) VINPUT DEC_ENG 10 1 100MegHz 1
+ DEVcon = -1
```

The output table consists of a list of all noisy devices at every frequency point, such as:

```
Spot  Noise  Summary  (in  V^2/Hz)  at  100  Hz  Sorted  by  Noise
Contributors
Element      Noise Contribution          % Of Total
   m         8.150e-016                  100.000


element     r1_tm
       fn  0.000e+000
    total  1.647e-025

 element     m
       rd  0.000e+000
       rs  0.000e+000
       id  7.933e-063
       fn  6.642e-031
    total  8.150e-016
```

```
**** total output noise voltage         = 8.150e-016 V^2/Hz
                                        = 2.855e-008 V/sqrt(Hz)

     transfer function value:
       v(nd)/vg                         = 8.060e-007
     equivalent input noise at vg
                                        = 3.542e-002 1/sqrt(Hz
```

.OP (Operating-Point Analysis)

Syntax

```
.OP <format1> <time1 ...> <CALLV> <SAVEV>
```

format1	Can be one of the following formats: • **ALL:** Prints a complete summary of voltage, current, conductance and capacitance values at a given operating point during Transient analysis. • **CURRENT:** Prints a short summary of voltage, current and power values. • **VOLTAGE:** Prints a summary of voltage values.
time1	Specifies a timepoint at which the corresponding summary must be printed.
CALLV	Calls a previously saved operating point before starting DC operating point calculation.
SAVEV	Saves the calculated DC operating point.

The inclusion of this statement in an input deck causes SmartSpice to determine the DC operating point of the circuit with inductors shorted and capacitors opened. When run in batch mode, this statement also causes all output variables to be printed immediately after the DC operating point calculation is done.

Note: A DC operating point analysis is automatically performed prior to a Transient analysis to determine the transient initial conditions. The DC operating point analysis is also run prior to an AC small-signal analysis to determine the linearized small-signal models for nonlinear devices.

If at least one format time pair is specified, the corresponding DC operating point calculation is not performed. In this case, at least one .TRAN statement must be specified. There is no limit to the number of format-time pairs that can be specified in one .OP statement. Any number of .OP statements can be specified in an input deck. During the corresponding Transient analysis, the format<i> summary is printed and saved in rawfile at exactly the timepoint t<i>. If the timepoint does not correspond to an actual timepoint, data will be interpolated prior to being output. The first letter is sufficient in a format specification.

If the input deck contains the .OP statement without ALL, VOLTAGE and CURRENT specifications, SmartSpice will always execute this statement and print outputs of type ALL. Printing is suppressed by specifying the option BRIEF.

If the input deck contains the .OP statement with the ALL, VOLTAGE and CURRENT specifications, and the .TRAN statement is not present in the input deck, then this .OP statement will be executed as a statement without ALL, VOLTAGE and CURRENT specifications.

If the input deck contains both the .TRAN and .OP statements, and the .OP statement contains one of the parameters ALL, VOLTAGE and CURRENT, then the DC operating point will be computed only once during the Transient analysis. The ALL, VOLTAGE and CURRENT specifications of the .OP will not be executed as an independent analysis command.

Device Type Information in the Operating Point Output

The device type information is now displayed in the Operating Point output.

For example, when using the HiSIM model:

```
****** model type: HISIM

subckt
element     msat            mlin            mcut
model       mosn            mosn            mosn

type        nmos            nmos            nmos
region      Saturation      Linear          Cut-off
power        4.6338e-03      2.2850e-06      6.4624e-08
id           2.3169e-03      5.7124e-05      2.6817e-08
ibd         -1.0100e-12     -1.0100e-12     -1.0100e-12
ibs         -3.0100e-12     -1.0500e-12     -6.0100e-12
```

The device type displayed depends on the technology:

- NMOS/PMOS: for MOSFETs and SOIs;
- NPN/PNP: for Bipolars (and additionaly LPNP for BJT level 1 & 2 and Mextram);
- NMF/PMF: for MESFETs;
- NJF/PJF: for JFETs;
- NTFT/PTFT: for TFTs.

Examples

```
.OP
.OP 1ns c 2ns vol 7ns all 5ns current 3ns
```

The first example calculates the DC operating point. In the second example, SmartSpice does not calculate the DC operating point. During Transient analysis, it prints and saves in rawfile complete operating point summaries at (approximately) `1ns` and `5ns`; current summaries at `2ns` and `3ns`; and a voltage summary at `7ns`.

Initial Point Algorithms

Convergence of the Newton-Raphson method depends very much on how close the initial point is to the final solution. In SmartSpice, the default initial point is set according to nonlinear device parameters, but independently of the circuit configuration. In some difficult cases, this initial point is too poor to help the Newton-Raphson method find a solution without a large number of iterations.

When the simulation is performed more than once on the circuit with the same configuration, the number of iterations needed can be significantly reduced using SmartSpice's ability to store and retrieve the previously computed operating point. In most cases, the Newton-Raphson method needs only a few iterations to converge starting from the stored operating point.

If an input deck contains two or more analysis commands, or a command for parametric analysis or optimization, such as `.TEMP`, `.MODIF`, `.ST`, nested `.DC` and `SWEEP`, then it is reasonable to specify the parameters `SAVEV` and `CALLV` in analysis command lines. The parameter `SAVEV` causes the computed operating point to be stored in the operating memory. The parameter `CALLV` causes the previously stored operating point to be retrieved from the operating memory, and is used as the initial point for Newton-Raphson iterations.

If it is necessary to simulate the same circuit using two or more different input files, it is reasonable to store the operating point calculated during the first simulation in a file using the `.SAVEBIAS` statement. This file has a `.NODESET` statement containing all of the node voltages in the circuit. The stored operating point can subsequently be retrieved from the file using the `.LOADBIAS` statement.

.OP Monte-Carlo

An operating point analysis supports Monte-Carlo.

Syntax

```
.OP MONTE=val <PRMC> <donotprintop>
```

MONTE=val	Performs Monte-Carlo statistical analysis.
val	The number of Monte-Carlo repetitions.
PRMC	Prints the value of modified and measured parameters on each Monte-Carlo iteration.
donotprintop	Skips operating point summary printing. Could be used without Monte-Carlo.

Parameters of Gaussian, Uniform or Limit function distributions are set by the .PARAM statements. The name and type of measurements, and names of output variables for Monte-Carlo analysis are set by .MEASURE statements.

Example

```
.PARAM Parameter1 = 'AGAUSS(1,1,1)'
.PARAM Parameter2 = 'AGAUSS(2,2,2)'
V1 1 0 1
R1 1 2 'Parameter1'
R1load 2 GND 1
R2 1 3 'Parameter2'
R2load 3 GND 1
.option nomod nodeck
.option post=2
.OP MONTE=3 prmc donotprintop
.print op v(2) v(3) @R1[res] @R2[res]
.measure op v2_max max v(2)
.measure op v3_max max v(3)
.end
```

.OPTIONS (Option Specification)

See section Section 3.14 .OPTIONS (Option Specification)

.OVERSHOOT (Check Node Voltages)

Syntax

```
.OVERSHOOT filename=”val” <vmin=val> <vmax=val> <duration=val>
+ <nodes=”nodename <nodename <...<nodename>>>”>
+ <excludenodes=”nodename <nodename <...<nodename>>>”>
```

This statement causes SmartSpice to check all nodes on every timepoint during the simulation, and report nodes with voltages exceeding limits specified by `vmin` and `vmax`. Minimum spike duration that will be detected can be controlled with duration variables. Therefore, it is possible to tell SmartSpice to skip short insignificant spikes. It is also possible to specify a nodes list with nodes that have to be included and/or excluded from checking. The output report is sorted by nodes. If a spike occurred at any certain node, first there’s a line stating that *v(x)* >`vmax` or *v(x)* <`vmin`, and then is followed by listing of spikes detected. This list contains a spike start time, spike end time, spike duration and peak voltage (maximum voltage in case of >`vmax` spike, or minimum voltage in <`vmin` spike case).

Node name patterns may be supplied to select nodes as well as patterns for node name exclusion. Together, these two pattern lists provide full control on node selection for overshoot checking. Use nodes parameter to supply patterns for node selection. If nodes parameter is not given, then all nodes are assumed (default).

It is possible to specify more than one statement for single analysis, but the only supported analysis is TRAN.

filename	The name of the file where overshoot information is saved. This file contains comment lines that identify the circuit name and type, the type and date of the simulation, specified parameters, and report on detected spikes.
vmin=val, vmax=val	These values specify voltage limits. At least one of them has to be present.
duration=val	Specifies minimum duration of spike to detect. If omitted, SmartSpice will detect spikes of any duration.
nodes	List of nodes to check. Wildcards can be used. If not specified, SmartSpice will check all nodes.
excludenodes	List of nodes to exclude from checking. Wildcards can be used. If not specified, SmartSpice will check all nodes.

Example

```
.OVERSHOOT filename="o.ost" vmin=-0.2 vmax=0.9 duration=1p
+ nodes=”v* x*.*”
+ excludenodes=”va* x3.*"

.overshoot vmax=0.1 nodes="*_in damped*" excludenodes="*x?.n_in"
+ filename="check.ost"
```

.PARAM (Parameter Definition)

Syntax

```
 Simple assignment and algebraic definition
.PAR/PARA/PARAM/PARAMS parname1=val1  <parname2=val2 ...>
.PAR/PARA/PARAM/PARAMS parname1=expr1 <parname2=expr2 ...>

or user-defined function
.PAR/PARA/PARAM/PARAMS  MyFunc(x,y)='expr(x,y)'
.PAR/PARA/PARAM/PARAMS  MyFunc='expr(x,y)'

or predefined analysis function
.PAR/PARA/PARAM/PARAMS parname = OPTxxx(initval lower upper)

.PAR/PARA/PARAM/PARAMS parname=UNIF(nomval, relvar <,mult>)
.PAR/PARA/PARAM/PARAMS parname=AUNIF(nomval, absvar <,mult>)
.PAR/PARA/PARAM/PARAMS parname=GAUSS(nomval, relvar, sigma <,mult>)
.PAR/PARA/PARAM/PARAMS parname=AGAUSS(nomval, absvar, sigma
+ <,mult>)
.PAR/PARA/PARAM/PARAMS parname=LIMIT(nomval, absvar)
```

The .PARAM statement defines one or more parameter labels by assigning the names parname1, parname2, either to constant values val1, val2, or to expressions expr1, expr2. Parameter labels must begin with alphabetic character(s) or the underscore character(s). Expressions must be enclosed in single quotes (’), and can contain numbers, previously specified parameter labels, operators, and functions. Function arguments can be expressions. Up to 9 levels of nesting can be used for the functions.

Redefinition of parameters causes re-evaluation. If a parameter is defined at some point in the deck, and then redefined later on, all parameters in the deck are recomputed to ensure that dependencies between parameters are accounted for correctly.

The .PARAM statement can be globally and locally (i.e., inside a subcircuit definition) specified. Once defined, a global parameter label or an expression containing global parameter labels can be used in the input deck when a numerical value is expected. A local parameter label can be used only inside the subcircuit.

.PARAM statement supports the ternary operator in the expressions defined functions.

Examples

```
RS1    1  2    'distr(5,5,6, 0.75)'
.param distr(b,n,w,skew1bm1w) ='n +  2. *shift* (w - n) * (b - n) /
(w == n
+ || n == b || w == b ? 1. : w - b + 0.75*shift * (w + b - 2. * n))'
```

Simple assignment and algebraic definition:

```
.param res1=10k bbb=1k bf=80 vdd=10 vdd2='vdd/2'
.param w2=2 L1=4
.param ddd=res1 eee='ddd+bbb'
.param a1='sin(res1*22)/100' a2='10*exp(bf/10)'
.param b1 = 'res1*33' b2 = 'bf/33'
RS2 3 0 res1
m1 1 2 3 4 L='L1*1e-6' w='W2*1e-6'
vdd 1 gnd vdd2
.model qnl npn(bf=bf rb=100 va=50)
```

In this example, resistor RS2 has a resistance defined by the parameter label res1, which is 10K. Parameter L of device m1 has the value L1*1e-6=4u, and parameter w of device m1 has the value W2*1e-6=2u. The voltage of the voltage source vdd is defined by the parameter label vdd2, which is 5. The value of the NPN model parameter for bf is given by the parameter label bf, which is 80.

```
.param ps=72
+   x14='max(0.0+min(60/2,((ps-64)/2-2)/2),0.002)'
r2 2 0 'max(0.1+min(0.9, 5), 1.02+max(-2,-10))'
```

In this example, the parameter x14 will be evaluated as [max(min(30,1), 0.002)=1 ?]. The resistor r2 will have the value max(0.1+0.9, 1.02-2)=1.

To see the values of the parameter labels contained in an input deck line, use the command LISTING PARAM.

User-defined function:

```
.param myfunc(x,y) = 'x*y'
```

Predefined analysis functions:

```
*Bisection input parameter definition
.PARAM DelayTime=Opt1(0.0n, 0.0n, 5.0n)

*MONTE CARLO PARAMETERS
.param nvt0 = 10e-3 $ V*um
.param pvt0 = 15e-3 $ V*um
.param nbeta = 2.3e-2 $ 100%*um
.param pbeta = 3.2e-2 $ 100%*um
.param delta_l = 0.0 $ m
.param sigma = 1.0 multiplier = 1

* Subcircuit Definitions
.subckt n d g s b chw=0 chl=0 mult=1 g=0
.param   dvt0   =   agauss(0.0,   'nvt0/sqrt(mult*chw*chl*1e12)',
sigma,multiplier)
.param   wdw   =   gauss('chw',   'nbeta/sqrt(mult*chw*chl*1e12)',
sigma,multiplier)
.param ldl = agauss('chl', delta_l, sigma, multiplier)
mn d g s b n w=wdw l=ldl m=mult geo=g delvto=dvt0 .
ends n
.subckt nl d g s b chw=0 chl=0 mult=1 g=0
.param dvt0 = agauss(0.0, 'nvt0/sqrt(mult*chw*chl*1e12)', sigma )
.param wdw = gauss('chw','nbeta/sqrt(mult*chw*chl*1e12)',sigma)
.param ldl = agauss('chl', delta_l, sigma)
mnl d g s b nl w=wdw l=ldl m=mult geo=g delvto=dvt0
.ends nl
```

Distribution Function

If f(x) is the probability density of a random variable x, then the cumulative distribution function is the integral of f(x). It can be approximated by a piecewise linear function.

Syntax

```
CDF(xyPairs)
```

Where:

- xyPairs: pairs of points [xi, yi]

The following rules apply:

- minimum two pairs are required
- y1 must be 0
- yn must be 1
- xi+1 must be greater than xi (monotonically increasing);
- yi+1 must be greater than or equal to yi (monotonically non-decreasing);

Example

```
.PARAM par1 = CDF(-2,0, -1,0.2, -0.5,0.5, 0.5,0.5, 1,0.8, 2,1)
```

String Parameter

String parameter is a parameter considered as character string. The keyword str('string') is used to define string parameter. When string parameter PAR is used, it is called as str(PAR).

The following netlist components support string parameters:

```
.PARAM
.DEFPARAM
.SUBCKT
Subcircuit call statements(Xcall)
m-device statements
```

Syntax

```
.param param_name=str('string')
.defparam param_name=str('string')
.SUBCKT subcktname <n1 n2 ...> <param_name=str('string')>
Xyyy n1 <n2 n3 ...> subcktname <param_name=str('string')>
Mxxx nd ng ns <nb> str(param_name)
```

Example

```
.MODEL NCH NMOS ...
.MODEL PCH PMOS ...
.SUBCKT   CELL       A   B   C   VDD   VSS   Y     PMOS_MODEL=str('PCH')
NMOS_MODEL=str('NCH')
lpp=2u
M1 Y C VDD VDD str(PMOS_MODEL) L=LpP W=WP
M2 Y A NET2 VSS str(NMOS_MODEL) L=LN W=WN
M3 NET2 B NET1 VSS str(NMOS_MODEL) L=LN W=WN
M4 Y A VDD VDD str(PMOS_MODEL) L=LP W=WP
M5 Y B VDD VDD str(PMOS_MODEL) L=LP W=WP
M6 NET1 C VSS VSS str(NMOS_MODEL) L=LN W=WN
.ENDS CELL
.....
.alter
.MODEL PCH_NEW PMOS ...
.param PMOS_MODEL=str('PCH_NEW')
```

In this example two default subcircuit parameters PMOS_MODEL and NMOS_MODEL are defined as string parameters and used in instance statements as model name references. The .param statement redefines the parameter PMOS_MODEL by new string 'PCH_NEW'. The .model PCH_NEW will be used in altered netlist for instances M1, M4 and M5 in subcircuit CELL.

.PAT (Pattern Definition for Voltage/Current Source)

SmartSpice provides pattern source function to include patterns from the .PAT statement.

Syntax

```
Vxxx n+ n- PAT <(> vhi vlo td tr tf tsample PatName <RB=val>
+ <R=repeat> <)>
Ixxx n+ n- PAT <(> vhi vlo td tr tf tsample PatName <RB=val>
+ <R=repeat> <)>

.PAT <PatName>=data <RB=val> <R=repeat>
.PAT <PatName>=[component 1 ... component n] <RB=val> <R=repeat>
```

PatName	The pattern name that has an associated b-string or nested structure.

Example

```
v1 1 0 pat (5 0 0n 1n 1n 5n a1 r=2 rb=2)
.PAT a1=b10mz r=1 rb=1
```

The final pattern source for this example:

```
b10mz r=1 rb=1 bzm01 r=2 rb=2
```

The final pattern function for this example:

```
‘10mz 10mz zm01 m01 m01’
```

Example

```
v1 1 0 pat (5 0 0n 1n 1n 5n [a1 b1100] r=1)
.pat a1=[b1m1 r=1 bm0m] r=2 rb=2
```

Final pattern function for this example:

```
‘1m1 1m1 m0m m0m m0m b1100 1m1 1m1 m0m m0m m0m b1100’
```

Example

```
* test .PAT functionality in Source
*
V1 1 0 pat (5 0 0n 1n 1n 5n a1 a2 r=2 rb=2)
.PAT a1=b10mz r=1 rb=1
.PAT a2=bzm01 r=1 rb=1
R1 1 0 19k
.end
```

.PLOT (Plot Vectors)

Syntax

```
.PLOT <anytype> outvar1 <(plo1, phi1)> <outvar2 <(plo2, phi2)> ...>
```

anytype	The type of analysis for which the specified outputs are desired. If anytype is unspecified, the .PLOT statement will be executed for all simulation types. anytype must be one of the following parameters: • **AC:** For AC analysis. • **DC:** For DC sweep. • **DISTO:** For distortion analysis. • **FFT:** For fast Fourier transform output. • **FOUR:** For Fourier analysis output. • **NET:** For network analysis output. • **NOISES:** For noise spectral density. • **NOISET:** For integrated noise. • **TRAN:** For Transient analysis.
outvar1...	Output variables or expressions to be plotted. For the description of output variables and expressions, see Chapter 6 Outputs and Chapter 4 Expressions.
plo1, phi1, ...	The optional lower and upper plot limits. They can be specified after any of the output variables. All output variables to the left of a pair of plot limits (plo, phi) are plotted using the same lower and upper plot limits. If plot limits are not specified, SmartSpice automatically determines the minimum and maximum values of all output variables being plotted, and scales the plot to fit. Even without plot limits, if the output variables have considerably different values, more than one y-scale will be used.

The input deck contains any number of .PLOT statements. Each .PLOT statement contains any number of output variables. The output variable all.vecname is used to plot all vectors with the name vecname.

In batch mode, the .PLOT statement instigates the program to plot simulation results in ASCII plot format, using ASCII characters to draw each waveform in the plot. This plot can be sent to any printer type. The overlap of two or more traces on a plot is indicated by the letter X. When more than one output variable appears on the same plot, the first specified variable will also be printed, unless the variable noasciiplotvalue has been set.

Examples

```
.PLOT AC VM(3) VR(4) VI(5,6)
.PLOT TRAN V(2,4) (10,100) I(VEE)
.PLOT V(12) IB(Q2) IS(M3)
.PLOT DISTO V(4)
.PLOT NOISES ALL
```

The first example produces an ASCII plot of the voltage magnitude at node 3, the real voltage part at node 4, and the imaginary voltage part between nodes 5 and 6 for all AC simulation results.

The second example plots using ASCII characters the transient voltage between nodes 2 and 4, with plot limits 10 and 100 in ASCII format. It also asciiplots the transient current across voltage source VEE in ASCII format.

The third example plots the voltage at node 12, the base current of BJT device Q2, and the source current of MOSFET device M3 in ASCII format. The plot of these signals is generated for all simulation types.

The fourth example plots signal V(4) for distortion analysis.

The fifth example plots all signals generated as a result of spectral density noise calculation.

.PRINT (Print Vectors)

Syntax

```
.PRINT <anytype> outvar1 <outvar2 ...>
```

anytype	The type of analysis for which the specified outputs are desired. If anytype is unspecified, the outputs of all specified simulation types will be printed. anytype must be one of the following parameters: • **AC:** For AC analysis outputs. • **DC:** For DC sweep outputs. • **DISTO:** For distortion analysis outputs. • **FFT:** For fast Fourier transform output. • **FOUR:** For Fourier analysis output. • **MEAS:** For saved results of .MEASURE statements. • **NET:** For network analysis outputs. • **NOISES:** For noise spectral density outputs. • **NOISET:** For integrated noise outputs. • **OP:** For operating-point analysis outputs. • **PZ:** For pole-zero analysis outputs. • **TF:** For transfer function outputs. • **TRAN:** For Transient analysis outputs.
outvar1...	Output variables or expressions to be printed. For the output variable and expression descriptions, see Chapter 6 Outputs (Section 6.2.4 MACROS) and Chapter 4 Expressions.

The output variable all.vecname is used to print all vectors with vecname.

In batch mode, the .PRINT statement causes the program to print simulation results. The input deck can contain any number of .PRINT statements. Each .PRINT statement contains any number of output variables. The values of output variables are printed in table form, with each column corresponding to one output variable. The variable width sets the table width. The number of digits printed is set by .OPTIONS NUMDGT or the variable numdgt.

Examples

```
.PRINT AC V(3) VR(7) VI(9)
.PRINT DC V(2) I(Vcc) ic(q1) ib(q2)
.PRINT TRAN v(2,9) i(r3) i(L1) ig(m2)
.PRINT NOISET INOISE_T_V ONOISE_T
.PRINT NOISES INOISE_S_V ONOISE_S
.PRINT DISTO v(3) v(4,5)
.PRINT TF zin_at_vin zout_at_4 tf
.PRINT PZ ALL
.PRINT NET S11 S12 S21 S22
```

.PROBE (Write Results in Rawfile)

Syntax

```
.PROBE <anytype> <outvar1 <outvar2 ...> | <MACROS> >
```

or

```
.PROBE/CSDF
```

anytype	The type of analysis for which a rawfile is created. If unspecified, the rawfile contains output for all analyses in the input deck. anytype must be one of the following parameters: • **AC:** For AC analysis outputs. • **DC:** For DC sweep outputs. • **DISTO:** For distortion analysis outputs. • **FFT:** For fast Fourier transform output. • **FOUR:** For Fourier analysis output. • **NET:** For network analysis outputs. • **NOISES:** For noise spectral density outputs. • **NOISET:** For integrated noise outputs. • **OP:** For operating-point analysis outputs. • **PZ:** For pole-zero analysis outputs. • **TF:** For transfer function outputs. • **TRAN:** For Transient analysis outputs.
outvar1...	Output variables or expressions to be stored. For the output variable and expression descriptions, see Chapter 6 Outputs (Section 6.2.4 MACROS) and Chapter 4 Expressions. It is also possible to use the following macros:

The .PROBE statement writes simulation results into a rawfile named basename.raw. basename is the name (without an extension) of the input file. When no output vectors or expressions are specified, all node voltages and device currents are written.

Examples

```
.PROBE
.PROBE V(10) V(11,12) I(VIN) I(RS1) IV(Q2)
.PROBE TRAN
.PROBE DC V(1) I(VDS)
```

The first example saves the results of all analyses in the input deck in one rawfile. All voltages and device currents are stored.

The second example saves the specified signals for all analyses in the input deck in one rawfile.

The third example saves all Transient analysis voltages and currents in one rawfile.

The fourth example saves the specified signals for DC analysis in a rawfile.

SmartSpice will now generate .dat files with data specified in .PROBE statements only in the option -pspice.

The macros TOP(V) and TOP(I) causes SmartSpice to save node voltages and branch currents only for top-level circuit nodes and branches (ignoring subcircuit levels).

```
.SAVE TOP(V) TOP(I)
.PROBE TOP(I)
```

.PROTECT (Make Deck invisible)

Syntax

```
.PROTECT | .PROT
```

Equivalent to `.OPTIONS BRIEF=1`.

.PZ (Pole-Zero Analysis)

Syntax

```
.PZ node1 node2 node3 node4 CUR|VOL|POL|ZER|PZ <CALLV> <SAVEV>
+ <UIC> < <SWEEP> MONTE=val <PRMC> >
```

The .PZ statement causes a pole-zero analysis to be performed on the circuit. SmartSpice creates a linearized small-signal model at the operating point of the circuit, and computes the poles and zeros of the transfer function. The SmartSpice input deck can contain more than one .PZ statement.

node1, node2	Input nodes.
node3, node4	Output nodes.
CUR	Calculates the (output voltage) / (input current) transfer function.
VOL	Calculates the (output voltage) / (input voltage) transfer function.
POL	Calculates poles.
ZER	Calculates zeros.
PZ	Calculates poles and zeros.
CALLV	Calls a previously saved operating point before beginning a DC operating point calculation.
SAVEV	Saves the DC operating point.
UIC	Bypass the DC operating point calculation and create a linearized model at the called operating point. The PZ analysis is free from any “No Path to Ground” problem.

Examples

```
.PZ 1 0 3 0 VOL POL
.PZ 1 0 3 0 VOL PZ CALLV SAVEV
.PZ 1 0 3 0 CUR ZER CALLV SAVEV
```

In the first example, SmartSpice calculates the DC operating point, linearizes the circuit, and computes poles for the transfer function of the type (output voltage) / (input voltage).

In the second example SmartSpice calls a previously saved operating point, computes and saves the DC operation point, and calculates the poles and zeros of the circuit.

In the third example, SmartSpice linearizes the circuit immediately around the called operating point, and calculates the zeros for the transfer function of the type (output voltage) / (input current).

.RTTEMP (Change Temperature During Transient Simulation)

Syntax

```
.RTTEMP temp time <<temp1 time1> ....<tempN timeN>>
.RTTEMP PWL (time temp << time1 temp1> ....<timeN tempN>>)
```

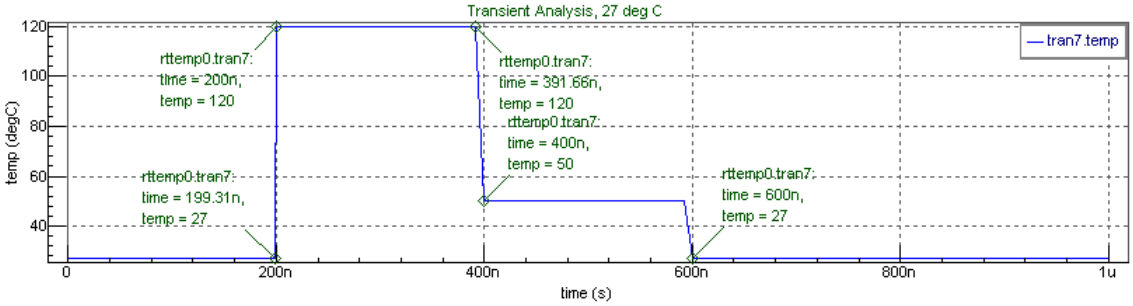
The .RTTEMP statement specifies temperature change during transient analysis, which will start with the default temperature or from the temperature specified in the .TEMP statement.

Switching pairs (temp, time) will change the temperature of the circuit which will stay constant until the next switching time. This statement is additive.

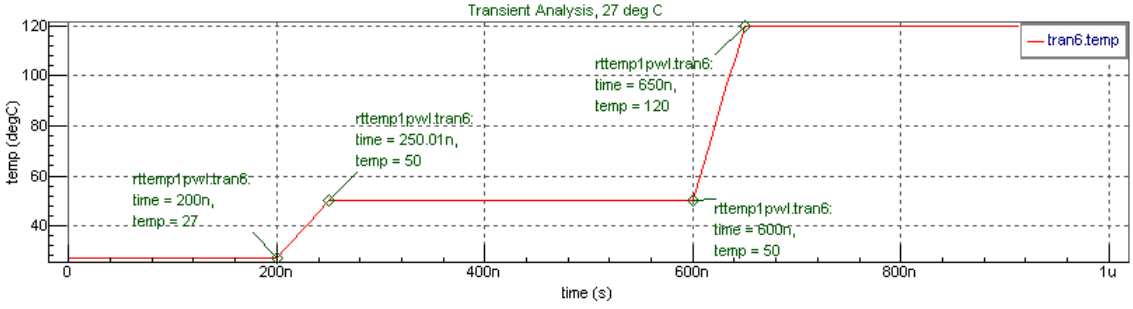
PWL (Piecewise linear function parameter) will change temperature gradually. This statement replaces previous .RTTEMP statements.

Example

```
.PARAM t1=120 time1=200n
.RTTEMP t1 time1
.RTTEMP 50 400n 27 600n
```




```
.PARAM t1=27 t2=50 t3=120 time1=200n time2=600n time3=900n time4=1u
dt=50n
.RTTEMP PWL (0 t1 time1 t1 'time1+dt' t2 'time2' t2 'time2+dt' t3
time3 t3)
```



SmartSpice automatically saves a parametric vector temp in transient plot when the .RTTEMP statement is specified in the netlist. The vector temp contains temperature value during a transient analysis.

The .RTTEMP statement can be used for dynamic control of model parameters during transient simulation.

Example

```
.rttemp pwl 0 27 10n 27 10.1n 40 20n 40 20.1n 27 40n 27
.tran 10n 50n
.probe temp @m1[bt]

.MODEL PTFT PTFT (
…
+BT        = '-0.316824E-6*(TEMP-80)+4.41017E-5'
…
```

The model parameter BT will be recalculated for (time, temperature) pairs. The temperature model of the transistor will be recalculated and the simulation will move on.

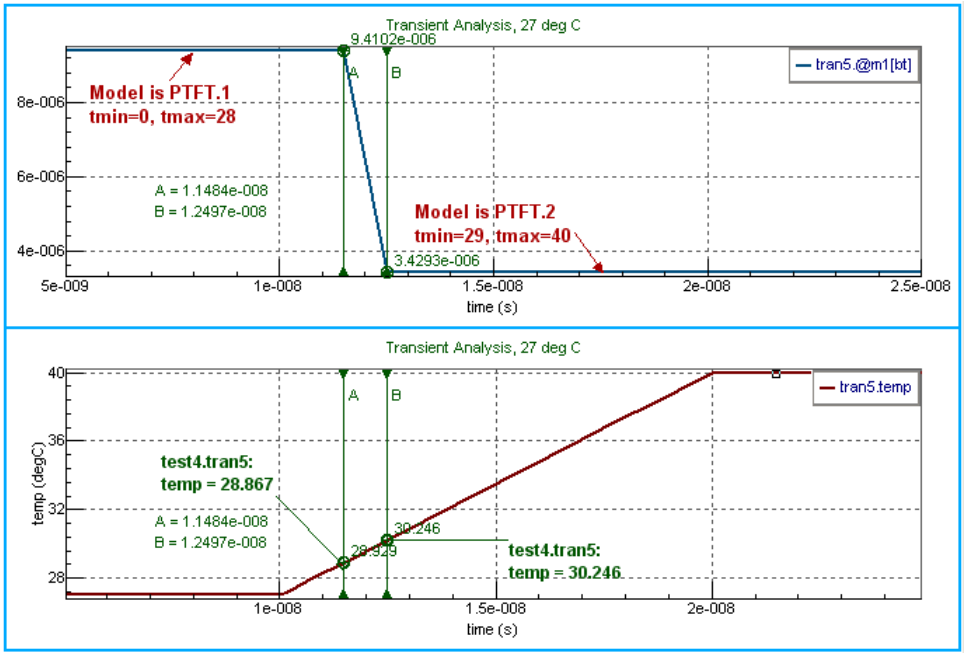
.RTTEMP is compatible with SmartSpice model binning algorithm. When the temperature is changed, the algorithm will check if the current model is still valid for the simulation. If temperature model binning is violated, the algorithm will look for the correct model and will replace the model for all transistors associated with this mode during simulation.

Example

```
.rttemp pwl 0 27  10n 27  20n 40
.MODEL PTFT.1 PTFT (... TMIN = 0  TMAX = 28 )
.MODEL PTFT.2 PTFT (... TMIN = 29 TMAX = 40 )
```

For time points A and B SmartSpice will call a binning algorithm to check that the correct temperature bins are used. If a new bin is found, the model PTFT.1 is replaced with PTFT.2 otherwise SmartSpice uses a model from the previous time point.

During the transition from 27°C to 40°C, model PTFT.1 stays active until the point A marker is reached. At point A, model PTFT.1 is out of temperature bin, but the model PTFT.1 is still active because the temperature binning algorithm did not find any model which satisfied the temperature binning parameters. At the next step, point B, temperature binning algorithm changes the active model to PTFT.2 model which satisfies the current temperature 30.246C.



.RTTEMP Thermal

The thermal equation extension for `.RTTEMP` is intended to adjust a chip temperature (T_j) during transient analysis and calculate the device contributions to the total chip power dissipation (P_d).

$$T_j = P_d \cdot Q_{jA} + T_A \quad (1)$$

- T_j: chip temperature at times t_{j1}, $t_{j2, \ldots}$,
- P_d: chip power dissipation (W),
- Q_{jA}: thermal resistance (constant parameter),
- T_A: ambient temperature (constant parameter).

$$P_d = \Sigma_m \{ f_m [I(device_m), V(node)] \} \quad (2)$$

where

- f_m[**I**(device$_m$),**V**(nodes)]: user-defined runtime expression for power dissipation in device$_m$,
- **I**(device$_m$): average current through device$_m$,
- **V**(node): average node voltage,
- m: device index (up to the maximum number of devices specified in `.RTTEMP` statement),
- t_D and t_O: duration and overlap time parameters for calculation **I**(device$_m$) and **V**(node). See the algorithm description below.

Algorithm Description

This algorithm is illustrated for the case of a single two-terminal device. The grey box (see Figure 3-11) is timeframe [t_{start}=0; t_{j1}= t_{start}+ t_D=150u] for calculation $\mathbf{I}_1$ (average current) for given device, $\mathbf{V}_1$(nodes) (average voltage) for given nodes and duration time t_D=150u, t_O=25u, which helps calculate and update temperature T_{j1} at time t_{j1}. The next timeframe [t_{j1}-t_O=125u; t_{j2}=275u] for calculation $\mathbf{I}_2$, $\mathbf{V}_2$(nodes) and update T_{j2} at t_{j2} are shown on the black framed box.

At time t_{j1}= t_{start}+ t_D the calculated $\mathbf{I}_1$ and $\mathbf{V}_1$(nodes) help to calculate T_{j1} using Equation 2 and chip temperature is updated to T_{j1} and remains constant until t_{j2} where it will be updated to T_{j2}. P_d will be calculated on the every timepoint t_j as expression of average current **I**(device$_m$) and average voltage $\mathbf{V}_1$(node).

P_d will be calculated for every timepoint t_j as expression of average (by default or if keyword `AVG` is specified in `.RTTEMP`) or `RMS` (if keyword `RMS` is specified) current I(device$_m$) and voltage V_1(nodes).

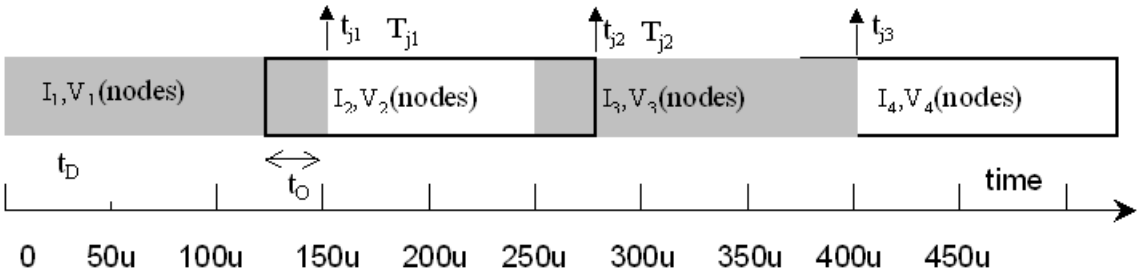


Figure 3-11 .RTTEMP THERMAL Algorithm

Note: t_{j1}, t_{j2}, t_{j3} ,... – timepoints for calculation and updating chip temperature to T_{j1}, T_{j2}, T_{j3},... . At these timepoints new T_j will be applied to the circuit.

Note: $\mathbf{I}_1$ and $\mathbf{V}_1$(nodes) are calculated as average current and average node voltage (or RMS current and RMS node voltage) for time [t_{start}=0; t_{j1}= t_{start}+ t_D=150u] with duration t_D. In case of `INTEGRAL` SmartSpice calculates the expression first and than integrates it over the period t_D.

Syntax

```
.rttemp thermal [AVG|RMS|INTEGRAL] TA=val TR=val TD=val TO=val
+ 'expression'
```

where

`thermal`	keyword to activate thermal equation extension.
`AVG`	(Average) SmartSpice will use average currents and voltages during TD while calculating P_d as described above. Default is `AVG`.
`RMS`	(Root Mean Square) SmartSpice will use `RMS` currents and voltages during TD while calculating P_d.
`INTEGRAL`	SmartSpice will integrate expression `'expression'` over period TD while calculating P_d
`TA`	ambient temperature T_A (real parameter).
`TR`	thermal resistance Q_{jA} (real parameter).
`TD`	duration t_D (positive real parameter).
`TO`	overlap time t_O (positive real parameter less than `TD`).
`'expression'`	user defined runtime expression, e.g., `` `I(R1)*(V(node1)-V(node2))+ I(C2)*(V(node3)-V(node4))` ``.

Note: You should specify duration t_D, overlap time t_O, ambient temperature TA, thermal resistance TR and runtime expression f_m[**I**(device$_m$),**V**(nodes)] for all necessary devices.

Note: Parameters TO and TD should be positive and TD>TO. Avoid situations when TD and TO are too close. In such cases performance could degrade significantly.

Output from .RTTEMP with Thermal Equation Extension

This statement will save calculated temperatures T_j in the vector temp during the transient analysis.

Example 1

```
VIN  1 0  DC 1  SIN(0 10 0.125K)
R1   1 2  1K
R2   1 2  2K
C1   2 0  100PF
.TRAN 1n 1m
.par Tmp_TA=27 Tmp_TR=50
.rttemp thermal TA=Tmp_TA TR=Tmp_TR TD=150u TO=50u
+ 'I(R1)*V(1,2)+I(R2)*I(R2)*2e3+I(C1)*V(2)'
* Actual value
.let  currenttemp='(I(R1)*v(1,2)+I(R2)*I(R2)*2e3+I(C1)*v(2))*Tmp_TR
+ Tmp_TA'
```

In this case, SmartSpice will save the calculated temperatures Tj1 in vector temp. For example, at point tj1=150u the terms I(R1)*V(1,2), I(R2)*I(R2)*2e3 and I(C1)*V(2) are calculated (by default, average currents and voltages) and summed according to Equation 2, Tj1 is calculated according to Equation 1, and circuit temperature is updated to Tj1. After that simulation continues to the timepoint tj2=250u where steps are repeated and circuit temperature is updated to Tj2 (see Figure 3-12).

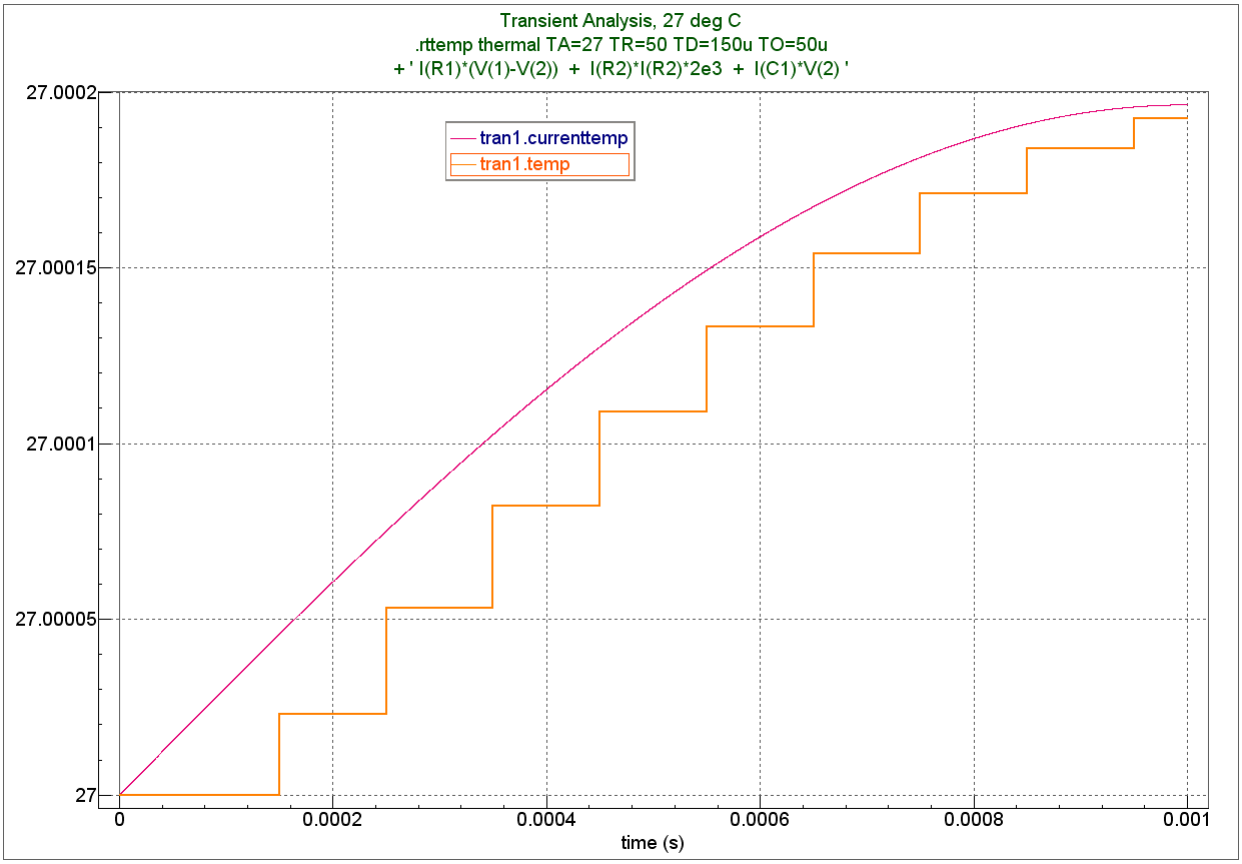


Figure 3-12 Vectors 'temp', 'currenttemp' From Example 1

Example 2

```
VIN  1 0  DC 0  SIN(0 0.1 5MEG)  AC 1
VCC  8 0  DC 10
VEE  9 0  DC -12
RS1  1 2  1K
RS2  5 0  1K
RC1  3 8  10K
RC2  4 8  10K
RBIAS  7 8  20K
CLOAD  3 4  5PF
Q1  3 2 6  QNL
Q2  4 5 6  QNL
Q3  6 7 9  QNL
Q4  7 7 9  QNL
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF CJC=2PF
+   VA=50)

.TRAN 1NS 171NS

* Parameters
.par Tmp_TA=27
.par Tmp_TR=100
.par Tmp_TD=50n
.par Tmp_TO=10n
```

```
.rttemp thermal  RMS TA=Tmp_TA TR=Tmp_TR TD=Tmp_TD TO=Tmp_TO
+ ' i(RS1)*(V(1)-V(2)) + i(Q1)*(V(2)-V(6)) + i(RC1)*(V(8)-V(3)) +
i(CLOAD)*(V(4)-V(3))'
```

In this case, SmartSpice will use `RMS` values for currents `i(RS1)`, `i(Q1)`, `i(RC1)`, `i(CLOAD)` and voltages `V(1)`, `V(2)`, `V(3)`, `V(4)`, `V(6)`, `V(8)` calculated during `TD`=50n because the keyword RMS is specified in `.RTTEMP` statement.

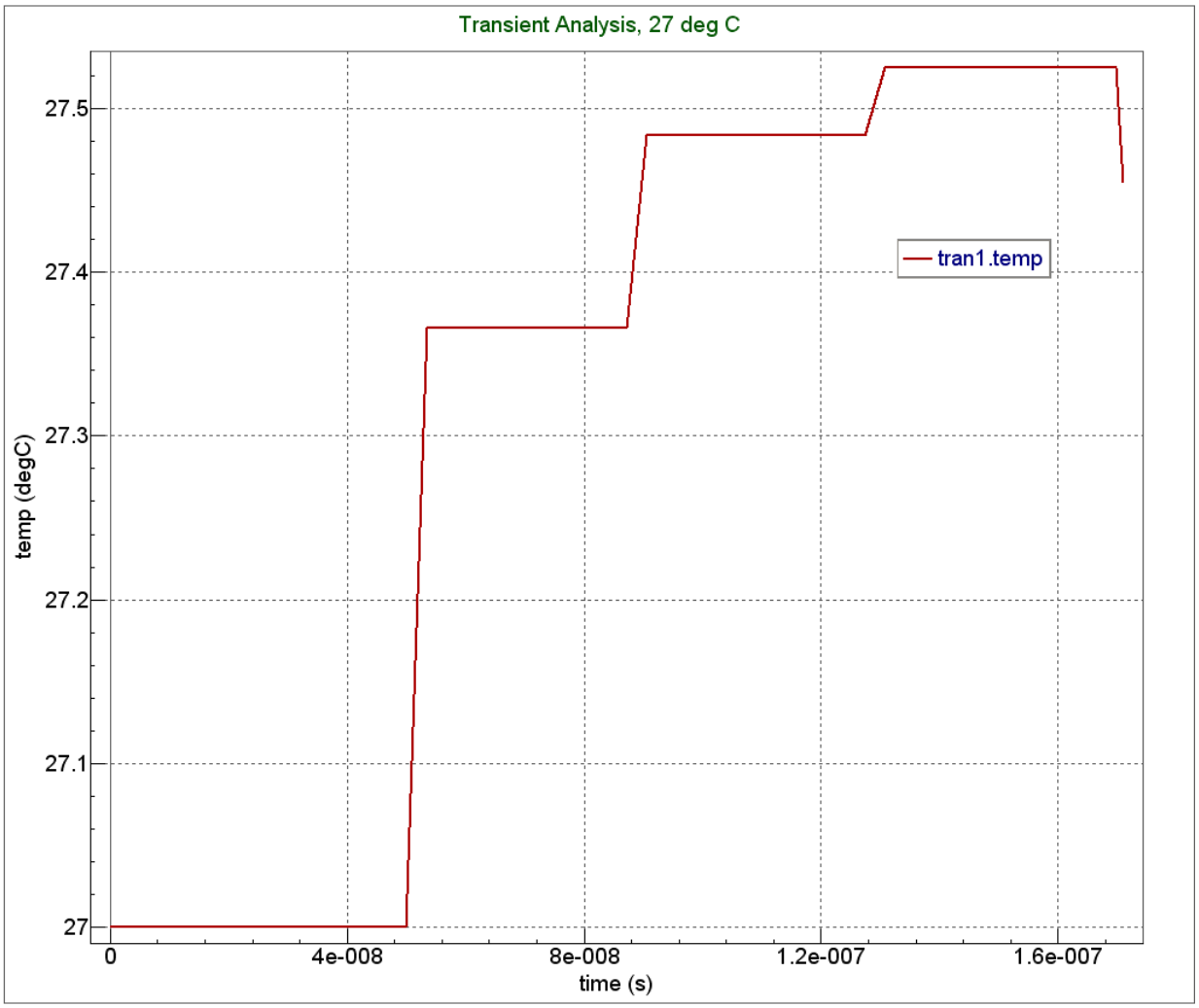


Figure 3-13 Vector 'temp' from Example 2

.SAVE (Save Vectors During Simulation)

Syntax

```
.SAVE <outvar1 <outvar2 ...> | <MACROS> >
```

This statement causes SmartSpice to save output variables `outvar1, outvar2, ...`, or a group of vectors defined by the `MACRO`. For the output variable descriptions, see Chapter 6 Outputs (Section 6.2.4 MACROS).

SmartSpice now supports the following:

Syntax

```
.SAVE <analysis> VEC=EXPRESSION
```

analysis	The type of analysis for which a new vector `VEC` is created. It must be one of the following: AC, DC, DISTO, NET, NOISES, NOISET, OP, PZ, TF, TRAN. If `analysis` is omitted, the `VEC` vector is created for all analyses specified in the input deck.
VEC	The name of new vector.
EXPRESSION	A standard output variable expression.

Examples

```
.SAVE v(1) v(2) I(VIN) IC(Q1)
.SAVE IB(Q3) ID(M22) I(D1)
```

In the first example, SmartSpice adds the following signals to the list of signals to be saved during the simulation: voltages at nodes `1` and `2`, currents through voltage source `VIN`, and the collector current of bipolar device `Q1`.

In the second example, SmartSpice saves the base current of bipolar device `Q3`, the drain current of MOSFET device `M22`, and the current through diode `D1`.

The macros `TOP(V)` and `TOP(I)` causes SmartSpice to save node voltages and branch currents only for top-level circuit nodes and branches (ignoring subcircuit levels).

Example

```
.SAVE TOP(V) TOP(I)
.PROBE TOP(I)
```

For MOSFET devices:

```
vd(devicename), vg(devicename), vs(devicename), vb(devicename)
```

the voltage difference between the terminal voltage of a MOSFET device (`vd`-device, `vg`-gate, `vs`-source, `vb`-bulk) and ground.

.SAVEBIAS (Save Bias Point to File)

Syntax

```
.SAVEBIAS filename anytype <NOSUBCKT> <TIME=val <REPEAT>>
+ <TEMP=val>
+ <STEP=val>
+ <DC=val> <DC1=val> <DC2=val> <ALL>
```

This statement causes the bias-point node voltages for the specified analysis (OP, TRAN or DC) to be saved to file `filename`.

`filename`	The name of the file where the bias-point node voltage is saved. This file contains comment lines that identify the circuit name and title, the type and date of the simulation, and one `.NODESET` statement with the bias point voltages.
`anytype`	The analysis type for which bias-point node voltages are saved. This can be one of the following: OP, TRAN or DC.
`NOSUBCKT`	If this parameter is specified, no subcircuit node voltages will be saved.
`TIME=val <REPEAT>`	The parameter followed by a value that specifies the time when the Transient analysis bias point is to be saved. If the optional parameter `REPEAT` is omitted, then the next bias point greater or equal to `val` will be saved. If `REPEAT` is specified, then `val` is the interval at which the bias point will be saved. The file `filename` is written the first time at `time=val`, then overwritten at `time=2*val`, `3*val`, and so on until the transient simulation terminates. Specifying `0` for `val (TIME=0 REPEAT)` causes the bias point to be stored after each time point.
`TEMP=val`	Specifies the temperature at which bias-point information will be saved.
`STEP=val`	Specifies a value for the step variable at which the bias point will be saved.
`DC=val`	The DC sweep variable value at which the bias point will be saved. This can be used for single-variable (non-nested) DC sweeps.
`DC1=val, DC2=val`	Values for first and second DC sweep variables, respectively, where the bias point will be saved. This can be used for nested sweeps.
`ALL`	When this flag is specified, the bias file will contain all node voltages.

If the flag `ALL` is not specified, the bias file will contain the node voltages, as they were specified in output statements (`.PRINT`, `.PROBE/...`).

The .SAVEBIAS statement causes bias-point node voltages for the specified simulation type and specified simulation conditions to be saved to the file filename. This file can later be used by the .LOADBIAS statement to speed up the simulation of the same circuit.

Note: A bias file is generated under CONV=-1(TRANOP) condition.

Examples

```
.SAVEBIAS "SAVED.BIAS" OP
.SAVEBIAS /main/examples/sbias1.in tran
.SAVEBIAS "sb3.tran" tran time=1n repeat
.SAVEBIAS "sb2.in" dc dc=3
.SAVEBIAS "sb4.dc" dc dc1=1.2 dc2=2.5
.SAVEBIAS "sb5.dc" dc step=80
```

The first example causes the small-signal operating point bias point to be saved in the file SAVED.BIAS.

The second example causes the transient bias, at the first time point greater than or equal to 0 seconds, to be saved to the file sbias1.in in the directory /main/examples.

The third example causes the bias point to be stored in the file sb3.tran every 1n, overwriting the previous contents of the file.

The fourth example stores the DC bias point when the DC sweep variable value is 3.

The fifth example stores the DC bias point when a nested sweep is run, and when the first and second DC variables are equal to 1.2 and 2.5, respectively.

The sixth example causes the bias point to be stored in the file sb5.dc when the DC simulation is run inside an .ST sweep, and the first DC point is stored when the step variable value is 80.

The standard SmartSpice .SAVEBIAS statement is used to store some or all node voltages of the circuit in a file in the .NODESET format. This statement can also be used to store the transient operating point in a special .IC format that allows a new Transient analysis to be started at this point.

Syntax

```
.SAVEBIAS filename TRAN TIME = val <REPEAT> IC|NIC
```

This statement causes the transient operating point to be saved to the file filename. The operating point is stored in the form of a continuation .IC statement.

filename	The name of the file where the operating point is to be stored. filename will be stored in a directory where the input deck is located.
TRAN	This mandatory parameter indicates that the .SAVEBIAS statement is applied to the Transient analysis.
TIME=val	This mandatory parameter defines the time point (or time points) where the transient operating point is to be saved. If REPEAT is specified, val must be positive (val > 0).

REPEAT	This optional parameter causes the transient operating point to be stored at the time points: `0`, `val`, `2*val`, and so forth until the time point at which the Transient analysis is terminated.
IC|NIC	These parameters define the continuation `.IC` statement formats. One of them must be specified in the `.SAVEBIAS` statement. `IC` causes a compact format to be used. `NIC` causes a format with the variable names to be used (not recommended for large circuits).

.SAVEBIAS behavior together with .ALTER statement

If the `.ALTER` statement is specified in the netlist, `.SAVEBIAS` will save the bias-point node voltages or the operating point for every altered circuit with the name filename-alternumber.

.SAVEBIASI (Save Current Bias Value to File)

Syntax

```
.SAVEBIASI filename anytype <TIME=val <REPEAT>> <TEMP=val>
+ <STEP=val> <DC=val> <DC1=val> <DC2=val> <ALL> <NOSUBCKT>
```

This statement causes the bias-point device currents for the specified analysis (OP, TRAN, or DC) to be saved to file `filename`.

filename	The name of the file where the bias-point device current is saved. This file contains comment lines that identify the circuit name and title, the type and date of the simulation, and the bias point currents.
anytype	The analysis type for which bias-point device currents are saved. This can be one of the following: OP, TRAN, or DC.
TIME=val <REPEAT>	The parameter followed by a value that specifies the time when the transient analysis bias point is to be saved. If the optional parameter `REPEAT` is omitted, then the next bias point greater or equal to `val` will be saved. If `REPEAT` is specified, then `val` is the interval at which the bias point will be saved; the file `filename` is written the first time at `time=val`, then overwritten at `time=2*val`, `3*val`, and so forth until the transient simulation terminates. Specifying `0` for `val` (`TIME=0 REPEAT`) causes the bias point to be stored after each time point.
TEMP=val	Specifies the temperature at which bias-point information will be saved.
STEP=val	Specifies a value for the step variable at which the bias point will be saved.
DC=val	The DC sweep variable value at which the bias point will be saved. This can be used for single-variable (non-nested) DC sweeps.
DC1=val, DC2=val	Values for first and second DC sweep variables, respectively, where the bias point will be saved. This can be used for nested sweeps.
ALL	When this flag is specified, the bias file will contain all device currents and the `NOSUBCKT` keyword will be ignored. If the flag `ALL` is not specified, the bias file will contain the device currents as they were specified in output statements (`.PRINT`, `.PROBE`, `.SAVE`, `.PLOT`...).
NOSUBCKT	If this parameter is specified, no subcircuit device currents will be saved.

The option `NUMDGT` defines the output format (number of digits) to save into file.

Example

```
.SAVEBIASI op_current_bias OP
```

```
.SAVEBIASI tran_current_bias tran time=2n repeat
.SAVEBIASI dc_current_bias dc dc=0.4
.SAVEBIASI dc_current_bias dc dc1=2 dc2=5
.SAVEBIASI dc_current_bias dc step=5
```

The first example causes the OP current bias point to be saved in the file op_current_bias.

The second example causes the bias point to be stored in the file tran_current_bias every 2n, overwriting the previous contents of the file.

The third example stores the dc bias point when the DC sweep variable value is 0.4.

The fourth example stores the dc bias point when a nested sweep is run, and when the first and second DC variables are equal to 2 and 5, respectively.

The fifth example causes the bias point to be stored in the file dc_current_bias when the DC simulation is run inside an .ST sweep, and the first DC point is stored when the step variable value is 5.

Example of .SAVEBIASI file for Transient analysis (NUMDGT=4)

```
*--------------------------------------------------------------------
*
* SmartSpice .SAVEBIASI file from:
*
* Input file: ram.in
* Title: ram.sp SPICE FILE
* Name: tran1 (Transient Analysis, 70 deg C)
* Date: Mon Aug  1 11:23:18 2005
* Time: 0
*
*--------------------------------------------------------------------
id(m13877)    -9.5080e-12
ig(m13877)    -0.0000e+00
is(m13877)     8.9028e-12
ib(m13877)     6.0517e-13
i(vresetbar)      0.0000e+00
i(vid7)      6.3723e-12
i(vid6)      6.3723e-12
i(vid5)      6.3723e-12
i(vid4)      6.3723e-12
```

.SHOW (Print Device/Model Parameters)

Syntax

```
.SHOW filename=param.inf time=time1 <time2 <...<timeN>>> <repeat>
+ object=<-l|-ll> <<devname1 ...> : <paramname1 ...>>
```

filename	This parameter is followed by the name of the file where the .SHOW information is saved.
time	This parameter starts the timepoint list when the .SHOW information must be saved.
repeat	This parameter defines the time period for saving. Default is the largest timepoint value.
object	This parameter starts the device and model parameters list.

The .SHOW statement prints device or model parameter names of the current circuit during the Transient analysis.

The options <-l|-ll> specifies the format of the output: -l specifies the short list format (one parameter per line); and -ll specifies the long list format (one parameter per line, including parameter descriptions). The default format is -l.

Examples

```
.show filename= param.info time= 10ns 20ns object= R* : all
.show filename= param1.info time= 10ns repeat object= all
.show filename= param2.info time= 10ns object= C1 V1
```

.SNS (Sensitivity Analysis)

Syntax

```
.SNS DC|TRAN outvar1 <outvar2 ...> TO param1 <param2 ...>
+ <ARG <=val>> <ARG2=val> <ARG3=val> <MAXSNS> <MAXNORM> <WCASE>
```

This statement calculates the sensitivity of any basic output variable with respect to most input parameters at any DC or Transient analysis point. An input deck can have only one `.SNS` statement. If the `.SNS` statement is specified, then SmartSpice performs sensitivity analysis for each `.DC` or `.TRAN` statement of the input deck.

SNS analysis uses the .option numdgt to control output precision.

DC|TRAN	Types of analysis.
outvar1, outvar2	Names of basic output variables (node voltages and branch currents).
TO	This parameter follows the list of output variables and is followed by the list of parameters and control options.
param1, param2	Names of parameters. The following parameter types are accepted: • the name of a device parameter. • the name of a model parameter. • the parameter `TEMP`. • the parameter `GMIN`.

The statement "`.SNS DC ... TO GMIN`" means the sensitivity output variables with respect to `DCGMIN`, and "`.SNS TRAN ... TO GMIN`" with respect to `GMIN`.

ARG=val	DC sweep argument value, or timepoint of the Transient analysis, where the sensitivity is calculated (the default value is `swstart` of the corresponding `.DC` statement, or the first timepoint of the Transient analysis).
ARG2, ARG3	Two additional argument values, where the sensitivity is calculated.
MAXSNS	This parameter is used to print the maximum sensitivity for `outvar1` and `outvar2`.
MAXNORM	This parameter is used to print the maximum of normalized sensitivity for `outvar1` and `outvar2`.
WCASE	This parameter is used to print the worst case for `outvar1` and `outvar2`.

The basic variables of the circuit are:

- Node voltages of the circuit.
- Currents through inductors.

- Currents through independent voltage sources.
- Currents through controlled voltage sources of all types.

Examples

```
.SNS DC V(1) I(VIN) TO ARG=-0.25 R1(RES) QNL(BF) TEMP
.SNS TRAN V(5) TO M*(W) ARG=6NS ARG2=26NS
```

In the first example, SmartSpice calculates the sensitivity of the output variables `V(1)` and `I(VIN)` with respect to the parameter `RES` of the device `R1`, to the parameter `BF` of the model QNL, and to temperature at the argument value `-0.25` of the DC sweep.

In the second example, SmartSpice calculates the sensitivity of output variable `V(5)` during the Transient analysis, with respect to the parameter `W` of all circuit MOSFET transistors, at the time points `6ns` and `26ns`.

The formula for normalized sensitivity of the output variable has been changed from

$$\delta = \left(\frac{\partial V}{\partial P}\right) \times (P \times 100)$$

to

$$\delta = \left(\frac{\partial V}{\partial P}\right) \times (P \times 100)/V,$$

where V is output variable, and P is a varied parameter.

Example

```
Example: Buffer
* Sensitivity of output waveform to circuit parameter changes
*************************************************************cms
.PARAM  parm7w=500u parm8w=500u vccdc=4.5

***** Main Circuit
vin  in1  0  DC 1.5  PULSE( 0.3 3.2 2N 5N 5N 15N )
vcc  vdd  0  DC vccdc
x1 in1  vdd  0 out1 stage1
r1 out1 in2 1
x2 in2  vdd  0 out2 stage2
r2 out2 in3 1
x3 in3  vdd  0 out3 stage3
r3 out3 in4 1
x4 in4  vdd  0 out4 stage4
m9  0  out4     0     0     nch   W=380U     L=1.2U
COUT  out4  0  700p

***** Sub Circuit Blocks

.subckt stage1 in vdd vss out
m1  vdd  in   out   vdd   pch   W=5.067U   L=1.6U
m2  out  in   vss   vss   nch   W=34.82U   L=2.0U
.ends stage1

.subckt stage2 in vdd vss out
m3  vdd  in   out   vdd   pch   W= 50u     L=1.1U
m4  out  in   vss   vss   nch   W=70.U     L=1.0U
.ends stage2
```

```
.subckt stage3 in vdd vss out
m5  vdd  in   out   vdd    pch   W=320.U    L=1.1U
m6  out  in   vss   vss    nch   W=440.U    L=1.1U
.ends stage3

.subckt stage4 in vdd vss out
m7  vdd  in   out   vdd   pch   W=parm7w   L=1.1U
m8  out  in   vss   vss   nch   W=parm8w   L=1.1U
.ends stage4

***** Models
.MODEL   pch   PMOS  (  level=3  tox=.02e-6  phi=0.576  gamma=0  vto=0
alpha=0 kappa=0 is=1e-14 )
.MODEL   nch   NMOS  (  level=3  tox=.02e-6  phi=0.576  gamma=0  vto=0
alpha=0 kappa=0 is=1e-14 )

.TRAN 0.2NS 40NS CALLV SAVEV
.SNS tran v(out4) TO r1(res) pch(is) m.x3.m6(W)

.MEASURE  TRAN  del_R  DELAY   v(in1)  RISE=1  VAL=1.5  TARG=v(out4)
RISE=1  VAL= 'vccdc/2'
.MEASURE  TRAN  del_F  DELAY   v(in1)  FALL=1  VAL=1.5  TARG=v(out4)
FALL=1 VAL='vccdc/2'

.END
```

.ST (Parametric Analysis)

Syntax

```
.ST <LIN> swname swstart swstop swincr
```

or

```
.ST OCT|DEC swname swstart swstop np
```

or

```
.ST LIST swname value1 <value2 ...>
```

The .ST statement performs a parametric sweep on the sweep variable swname for all analyses specified on the input deck. As with the .TEMP statement, all ordinary analyses (.DC, .AC, .TRAN, etc.) and corresponding output statements (.PRINT, .PLOT and .MEASURE) are executed for each value of swname.

LIN	Linear sweep (the default sweep type).
OCT	Sweep by octaves.
DEC	Sweep by decades.
LIST	Sweep over a list of values.
swname	The name of the variable to be swept, and can be: • the name of an independent voltage or current source. • the name of a device parameter. • the name of a model parameter. • the parameter TEMP, for a temperature sweep. • a parameter label from a global .PARAM statement.
swstart	The starting sweep variable value for LIN, DEC and OCT sweeps.
swstop	The final sweep variable value for LIN, DEC and OCT sweeps.
swincr	The increment value of the sweep variable for LIN sweep.
np	The number of points per decade or per octave.
value1, value2,...	The values of the sweep variable for the LIST sweep.

Examples

```
.ST VIN -2.0 5.0 0.25
.ST LIN VDS 0 10 .5
.ST DEC QNL(BF) 10 100 10
.ST LIST TEMP 0 20 27 50 -20
```

In the first example, the voltage of voltage source VIN is swept linearly from -2.0 to 5.0, with a 0.25 step.

In the second example, the voltage of voltage source VDS is swept linearly from 0 to 10 with a 0.5 step.

In the third example, parameter `BF` of the bipolar model `QN1` is swept logarithmically from `10` to `100` with a step of `10` points per decade.

In the fourth example, all specified simulations are performed at five temperatures: 0, 20, 27, 50, and -20.

.SUBCKT (Subcircuit Definition)

Syntax

```
.SUBCKT subcktname <n1 n2 ...>
+ <OPTIONAL: optn1=defval1 <optn2=defval2 ...>>
+ <<PARAMS|PARAM:> parname1=val1 <parname2=val2 ...>>
```

subcktname	The subcircuit name. It is used by an X element statement to reference the subcircuit.
n1 ...	The optional list of external nodes. Ground nodes (zero) are not allowed.
OPTIONAL	The parameter that specifies the start of optional nodes list.
optn1, optn2, ...	The names of optional nodes with their default values set to defval1, defval2, ... An optional node is a node that can be optionally specified in a subcircuit call. When it is not specified in a subcircuit call, its default value will be used.
PARAMS|PARAM	The parameter preceding the list of parameters and their values. This parameter is only mandatory when optional nodes are specified.
parname1, parname2, ...	The name of a parameter with value set to val1, val2, ... that will be local to the subcircuit. val is either a numerical value or an expression enclosed in single quotes containing previously defined parameters. If the same parameter is assigned, a value on more than one statement in the input deck (.PARAM, .SUBCKT and X statements) will be: • a value assigned in a global .PARAM statement (outside subcircuits) is used if it exists, otherwise, • a value assigned in a corresponding X statement is used if it exists, otherwise, • a value assigned in a local .PARAM statement (inside the subcircuit) is used if it exists. • a value assigned in a corresponding .SUBCKT statement is used if it exists, otherwise,

A subcircuit definition is initiated by a .SUBCKT statement. The group of element statements which immediately follow the .SUBCKT statement define the subcircuit. The last statement in a subcircuit definition is the .ENDS statement. Control statements can not appear within a subcircuit definition; however, subcircuit definitions may contain anything else, including other subcircuit definitions, device models, and subcircuit calls.

Note: Any model definition or element included as part of a subcircuit definition is local, and must be referenced outside of the subcircuit definition using its expanded name (see Section X (Subcircuit Call)). All internal nodes (if not included on the .SUBCKT statement) are local, with the exception of the global node 0 and any nodes defined in the .GLOBAL statement.

Examples

```
.SUBCKT OPAMP 1 2 3 4
.SUBCKT SUB1 10 11 12 par1=1.5k par2=1e-11
.SUBCKT SUB2 vcc vout PARAMS: a=2Meg b='sqrt(a*2)/10'
.SUBCKT SUB3 ad14 ad15 OPTIONAL: on1=vcc on2=vee
```

The first example starts the definition of subcircuit `OPAMP` with four external nodes `1`, `2`, `3` and `4`.

The second example starts the definition of subcircuit `SUB1`, with external nodes `10`, `11` and `12`, and with parameters `par1` and `par2`.

The third example starts the definition of subcircuit `SUB2`, with external nodes `vcc` and `vout`, and with parameters `a` and `b`.

The fourth example starts the definition of subcircuit `SUB3`, with external nodes `ad14` and `ad15`, and with optional nodes `on1` and `on2`. The optional nodes default to `vcc` and `vee`, respectively.

The operating temperature can be set as the default subcircuit parameter `TEMP` in the `.subckt` statement. The temperature will be applied for all instances of specific subcircuit.

Example

```
.subckt cell n1 <n2 n3 ...> temp=50
```

Note: A hierarchical temperature feature is active when the `.option parhier` is local.

Circuit Design Language (CDL) Support

SmartSpice supports the following CDL dialects.

Example

```
.subckt subcktname N1 N2 / N3 N4
```

In this example a slash (`/`) character distinguishes input ports (following the slash) from output ports (preceding the slash).

Multiply Parameter

In common case the `M` (multiply) parameter is used to simulate subcircuits and/or devices in parallel. This parameter multiplies the internal component values.

Example

```
XR1 1 0 a  m=2
v1 1 0 1
.subckt a n1 n2
x1 n1 n2 b
.ends
.subckt b n1 n2
x2 n1 n2 c
.ends
.subckt c n1 n2
.param m=7
r1 n1 n2 1 m=m
.ends
```

There are two places where the multiply parameter can be specified: subcircuit and device instance statements. In previous example, the subcircuit XR1 has a multiply parameter that equals 2. The resistor R1 multiply parameter is specified from the m parameter (m=m means that the multiply parameter is equal to the parameter m which is defined in the .PARAM statement and equals 7).

The multiply parameter is hierarchical parameter (i.e., for a subcircuit within a subcircuit), and the multiplier is the product of both levels.

The multiply parameter of the device instance is the product of the hierarchical multiply parameter and the device multiplier. In the previous example, the multiplier for R1 is equal 14 (2*7; 2- the hierarchical multiply parameter, 7 - device multiply parameter).

Default Subcircuit Parameters

Parameters specified in the .SUBCKT statement are default subcircuit parameters.

Example

```
XR1 1 0 a
v1 1 0 1
.subckt a n1 n2 q1=2  q2='q1+3'
.param q1=4
.param q3 = q1
r1 n1 n21  q2
r2 n21 n2  q3
.ends
```

In this example, the .subckt statement defines default value for parameters q1 and q2 as 2 and 'q1+3'. There are two .param statements for parameters q1 and q2 inside subcircuit definition. Previously, SmartSpice has used the default parameter values. The values of R1 and R2 were specified as 5 and 2. In this version SmartSpice redefines the default parameter value using subcircuit .param statements. The values of R1 and R2 are specified as 7 and 4.

.TEMP (Temperature Statement)

Syntax

```
.TEMP t1 <t2 ...>
```

t1 t2 ...	Temperature in °C, at which the simulation will be performed.

The .TEMP statement sets the temperature at which all analyses are to be performed. This is the circuit operating temperature. When more than one temperature is given, all analyses are performed for each temperature.

The element operating temperature and the operating temperature of individual circuit elements are changed by setting the parameter TEMP in the element statement.

The nominal temperature and the temperature at which model parameters are measured is set through the parameter TNOM. Default is 27 °C.

Model nominal temperature and the nominal temperature of an individual model is changed by setting the parameter TNOM in the .MODEL statement.

Examples

```
.TEMP -20
.TEMP 20 40 60 80 100
```

If the second example is included in the input deck, the circuit will be simulated at five temperatures: 20, 40, 60, 80 and 100 °C.

The expression in .TEMP statement can be re-evaluated during SWEEP/.MODIF.

Example

```
.param R=1 TempParameter=0
.temp 'TempParameter'

.data DataTable
R TempParameter
1    0
5   30
10 125
.enddata

.tran 1n 5n sweep DATA=DataTable
```

.TF (Transfer Function)

Syntax

```
.TF outvar insrc
```

This statement defines the small-signal output and input for the DC small-signal analysis. `outvar` is the small-signal output variable, and `insrc` is the small-signal input source. If this statement is included, SmartSpice computes the DC small-signal value of the transfer function (output/input), input resistance, and output resistance.

Examples

```
.TF V(5,3) VIN
.TF I(VLOAD) VIN
```

In the first example, SmartSpice computes the ratio of `V(5,3)` to `VIN`, the small-signal input resistance at `VIN`, and the small-signal output resistance measured across nodes `5` and `3`.

.TRAN (Transient Analysis)

This statement causes a Transient analysis to be performed on the circuit. It calculates circuit behavior over a specified time interval, starting from time zero.

Syntax

```
.TRAN</OP> <STEP=>tstep <STOP=>tstop <<START=>tstart> <tmax>
+ <UIC> <SKIPDC=no|yes> <CALLV> <SAVEV < =tsave>> <TRANOP < =top>>
+ <STORE=num> <FSTEPS=val>
+ <sw2_spcf>  < <SWEEP> MONTE=val <PRMC<=val>> >
+ <SWEEP OPTIMIZE=OPTxxx RESULT=measname MODEL=modname>
+ <NOISE<=val> <FREQ=val <FREQH=val>>> <NOISE_RELTOL=val>
+ <NOISE_IC=ZERO|AUTO>
+ <DELF=val> <MAXF=val> <DELT=val> <NMAX=val>
+ <LAPLACE_ACCURATE=val> <CKPTPERIOD=val> <speedplot>
```

or

```
.TRAN</OP> tstep1 tstop1 <tstep2 tstop2 ...  tstepN tstopN>
+ <START=tstart> <UIC> <SKIPDC=no|yes> <CALLV> <SAVEV < =tsave>>
+ <TRANOP < =top>>  <STORE=num>
+ <sw2_spcf> <FSTEPS=val> < <SWEEP>  MONTE=val  <PRMC<=val>> >
+ <SWEEP OPTIMIZE=OPTxxx RESULT=measname MODEL=modname>
+ <NOISE<=val> <FREQ=val <FREQH=val>>> <NOISE_RELTOL=val>
+ <NOISE_IC=ZERO|AUTO>
+ <DELF=val> <MAXF=val> <DELT=val> <NMAX=val>
+ <LAPLACE_ACCURATE=val> <CKPTPERIOD=val>
+ <write="FILENAME"> <writefinal="FILENAME2"> <speedplot>
+ <reltol=  { reltol1  , reltol2 , …, reltolN  }>
+ <abstol=  { abstol1  , abstol2 , …, abstolN  }>
+ <rmax=    { rmax1    , rmax2   , …, rmaxN    }>
+ <vntol=   { vntol1   , vntol2  , …, vntolN   }>
+ <trtol=   { trtol1   , trtol2  , …, trtolN   }>
+ <chrgtol= { chrgtol1 , chrgtol2, …, chrgtolN }>
+ <bytol=   { bytol1   , bytol2  , …, bytolN   }>
```

tstep	Time interval used to control the printing and plotting of Transient analysis results.
tstop	The end time of the Transient analysis.
tstart	Causes SmartSpice to store outputs for printing, plotting, and measuring from time `tstart`. Default is `0`.
tmax	The maximum internal time step that SmartSpice can use during Transient analysis. The choice of this parameter is critical to a speedy and accurate simulation. Large values of `tmax` speed up the simulation, but the danger exists of missing fast changes in the solutions. Default is: `tmax` $= min\left(RMAX \times tstep, \frac{tstop - tstart}{50}\right)$ where: *RMAX* is specified in the `.OPTIONS` statement, and the default value is equal to 10.

Note: During Transient analysis, under certain conditions, internal time step can be 10 times more than the default `tmax`, calculated with default value *RMAX=10*. In such cases internal time step can reach a value of $tstep \times 100$. If the parameters `tmax` or RMAX are specified (not the default value) in the `.TRAN` or `.OPTIONS` statements, then the internal time step is always less than or equal to `tmax`.

UIC	Causes SmartSpice to bypass DC operating point calculation. The Transient analysis starts immediately from initial conditions specified in the `.IC` statement or by `IC` parameters on the various elements.
SKIPDC	To allow for syntax from a Spectre netlist.
CALLV	Causes SmartSpice to call a previously saved operating point before starting Transient analysis.
SAVEV	Causes SmartSpice to save the operating point calculated at the time `tsave`.
tsave	The time when the calculated operating point is saved. Default is `0`.
TRANOP=top	Causes SmartSpice to compute the transient operating point over the time interval from `0` to `top`, starting from zero initial conditions. SmartSpice saves the operating point automatically, and performs the Transient analysis over the time interval from `0` to `tstop`. By default, `top=tstop`.
STORE=num	Saves data at one time point for each `num` timesteps during the transient simulation. By default, data is saved at each `timestep`. For example, `STORE=10` saves every 10th calculated time point.
FSTEPS=val	Minimum number of time steps per rise or fall region of circuit controlling signals (PULSE, PWL, ...) or between the next breakpoints. Default is `10`. If a circuit contains Laplace elements, the default is `4`.
/OP	This suffix on the `.TRAN` statement causes DC operating point information to be printed exactly as in the `.OP` statement.

sw2_spcf	The second (nested) sweep specification. It can be specified in one of the following ways:
	
	`SWEEP swname swstart swstop swincr`
	`SWEEP swname LIST nplist swval1 swval2 ...`
	`SWEEP swname swtype np swstart swstop`
	`SWEEP MODIF = dataname <PRTBL>`
	`<SWEEP> DATA = dataname`
	
	where
	`swname` is the name of the parameter to be swept. It can be:
	• a parameter label defined in a global `.PARAM` statement.
	• the name of an independent voltage or current source.
	• the full-path name of a device parameter.
	• the full-path name of a model parameter.
	• the parameter `TEMP` (temperature sweep).

swstart	The starting value of the swept parameter.
swstop	The stop value of the swept parameter.
swstep	The increment value of the swept parameter.
swincr	The increment value for a linear sweep.
nplist	Number of parameter values `Swval1, swval2, ...` specified for a sweep of the type LIST. If `nplist` is less than the number of parameter values given, the remaining parameter values will be ignored.
swtype	The parameter that defines the type of sweep: • **DEC:** Sweep by decades. • **OCT:** Sweep by octaves. • **LIN:** Linear sweep.
np	The number of points per decade or per octave, or number of points for LIN sweep.
MODIF=dataname	This references the parametric `.DATA` statement.
DATA=dataneme	This references the parametric `.DATA` statement.
PRTBL	If this flag is present, then the final table of all modified parameter labels and calculated `.MEASURE` results is printed. Default is `OFF`.

<SWEEP> MONTE=val	Causes SmartSpice to perform Monte-Carlo statistical analysis, where `val` is the number of Monte-Carlo repetitions. Parameters of Gaussian, Uniform or Limit function distributions are set by `.PARAM` statements:
	
	`.PARAM parname=UNIF(nomval, relvar <,mult>)`
	`.PARAM parname=AUNIF(nomval, absvar <,mult>)`
	`.PARAM parname=GAUSS(nomval, relvar, sigma <,mult>)`
	`.PARAM parname=AGAUSS(nomval, absvar, sigma <,mult>)`
	`.PARAM parname=LIMIT(nomval absvar)`
	
	where:

parname	Parameter name whose value is calculated by distribution function.
UNIF	Uniform distribution function with relative variation.
AUNIF	Uniform distribution function with absolute variation.
GAUSS	Gaussian distribution function with relative variation.
AGAUSS	Gaussian distribution function with absolute variation.
LIMIT	Random limit distribution function with absolute variation. `+/-absvar` is added to `nomval` based on whether the random outcome of a `-1` to `1` distribution is greater or less then `0`.
nomval	Nominal value for Monte-Carlo analysis.
absvar	The absolute variation: the `AUNIF` and `AGAUSS` vary `nomval` by `+/-absvar`.
relvar	The relative variation: the `UNIF` and `GAUSS` vary `nomval` by `+/-nomval x relvar`.
sigma	The parameter for Gaussian distribution. The `absvar` or `relvar` is specified at the `sigma` level. The effective standard deviation of a random sample will be `absvar/sigma`.
mult	The multiplier repeats function calculation `mult` times, saving only the largest deviation. Default is `mult 1`.
speedplot	Allows you to speed up simulation by reusing previously created SmartSpice structures (plot). Only one plot will be created, the waveforms will not be preserved. All measure results will be printed and preserved in the `.meas` plot. The raw files will contain only the data from the last sweep step and all measure results, if there is no `.probe` statements in the netlist. Default is off.

reltol **abstol** **rmax** **vntol** **trtol** **chrgtol** **bytol**	These options all share a common format of values: `x={x1, x2, x3,...xN}` where `x` is one of the options named, and `x1, x2...`are successive values of this option. Each value of the option parameter corresponds to the relevent time interval. Example: • 1st time interval `[0; tstop1]` uses the first option value `x1` • 2nd time interval `[tstop1:tstop2]` uses the second option value `x2` and so forth.

The name and type of measurements and names of output variables for Monte-Carlo analysis are set by `.MEASURE` statements.

PRMC<=val>	Causes SmartSpice to print the values of modified and measured parameters on each Monte Carlo iteration. In HSPICE mode. Flag `PRMC` saves distributed parameters in the measure file, if `.MEASURE` statement is used. The `PRMC=2` saves all parameters in the raw file, if the option `sweepmonte` is present. The feature doesn't work in HSPICE compatible mode.
OPTIMIZE	Causes SmartSpice to perform Bisection Optimization. This parameter is followed by the bisection optimization parameter.
OPTxxx	Optimization parameter reference name must agree with the `OPTxxx` name given in the `.PARAM` statement.
RESULT	This parameter is followed by the target for optimization.
measname	Name of the measure calculated by a `.MEASURE` statement. It can be a circuit performance measure like delay, rise/fall time, maximum/minimum value, or difference between a desired and a simulated curve calculated by the `.MEASURE` statement.
MODEL	This parameter is followed by the optimization model.
modname	The model name. It is used by bisection optimization to refer to a particular model.
NOISE<=val>	Causes SmartSpice to perform Transient Noise Analysis with noise sources depending on `val` to account for: • **`val=1, 2`:** Thermal noise only. • **`val=3`** (default)**:** Thermal, shot, and flicker (1/f) noise. • **`val=4`:** Shot noise only. • **`val=5`:** Flicker (1/f) noise only.

To account for flicker (1/f) noise in the simulation, additional parameters must be defined:

FREQ=val	Single frequency/low frequency of the band [FREQ, FREQH].
FREQH=val	High frequency of the band [FREQ, FREQH].

If NOISE or NOISE=3 is given in a .TRAN statement, but the parameter FREQ is not specified, flicker noise is not taken into account.

If NOISE=5 is given in a .TRAN statement, and the frequency parameter FREQ is not specified, SmartSpice uses the following default value: FREQ=20 Hz.

NOISE_RELTOL=val	Relative tolerance for LTE algorithm applied to correlation matrix integration. If the calculated timestep is smaller than the current timestep, then the timepoint is set back (timestep reversal) and the calculated timestep is used to increment the time. Default is 0.05. NOISE_RELTOL=0 turns off this additional timestep control algorithm.
NOISE_IC=ZERO	Sets Initial Condition for the noise voltage variance during Transient Noise Analysis at time=0 to 0. Default is AUTO. Forces Initial Condition calculation at time=0.

See Chapter 15 Transient Noise Analysis.

write	Writes a state file "FILENAME" for the initial transient analysis point.
writefinal	Writes a state file "FILENAME2" for the final transient analysis point.

Transfer Function Transform Parameters

When the circuit contains the devices, which are described by transfer function H(s), SmartSpice performs the conversion of these transfer functions to impulse response h(t). The conversion parameters (time and frequency resolutions) depend on circuit and Transient analysis mode, and must be the same for all such devices. You can specify conversion parameters directly in the .TRAN statement.

MAXF=val	*Nyquist critical frequency* - f_c. $2 \times f_c$ is the frequency window over which transfer function H(s) is calculated from LAPLACE, POLE or FREQ description.
DELF=val	*Frequency resolution* - Δf. $1/\Delta f$ is the time window over which transfer characteristic h(t) is calculated from H(s) by inverse FFT.
DELT=val	*Time resolution* - Δ, which is the uniform time interval between h(t) samples.
NMAX=val	Maximum number of transfer characteristic h(t) samples. Default is 8,388,608.

These parameter values affect accurate transient results, and the CPU time.

Parameters f_c and Δ are tied hard by relation $f_c = 1/(2 \times \Delta)$, and you can specify only one. If f_c and Δ are both specified, the Δ has a higher priority. The following table shows parameter calculation flow (N- number of h(t) samples):

Δf	f_c or Δ if f_c: $\Delta=1/(2 \times f_c)$ if Δ: $f_c=1/(2 \times \Delta)$	
Default	Default	$\Delta f=1/Tstop$; $f_c=N \times \Delta f$; $\Delta=1/(2 \times f_c)$; N=1024
Yes	Default	$f_c=N \times \Delta f$; $\Delta=1/(2 \times f_c)$; N=1024
Default	Yes	$\Delta f=1/Tstop$; $N=(2 \times f_c)/\Delta f$; match N as power 2; $\Delta f=(2 \times f_c)/N$
Yes	Yes	$N=(2 \times f_c)/\Delta f$; match N as power 2; $\Delta f=(2 \times f_c)/N$

In last case, when Δf and Δ are both specified, $N=(2 \times f_c)/\Delta f$ is calculated, and $\Delta \times N$ can be smaller than Tstop or N can be larger than `NMAX`. In such situations SmartSpice prints the warning message:

```
Warning: LAPLACE: required number points N=... > Nmax=...
                  finalTime was DECREASED to ...
                  frequency resolution DELF=...
```

and decreases $Tstop=\Delta \times min(N,Nmax)$.

Default value of parameter Δ (`TDELTA`) will be set to `minTimeSourceInterval` (minimum changing time (rise or fall) of all controlling signals (PULSE and PWL sources) with factor `val`, and specified in the `.TRAN` statement option parameter `LAPLACE_ACCURATE=val`. Default is `0`.

If the `MAXF`, `DELF` or `DELT` parameters are not specified, the corresponding device parameters are used.

Time-Varied Tstep and Tmax

The second `.TRAN` statement syntax allows you to vary parameters `tstep` and `tmax` during transient simulation. At each specified time interval $[tstop_{k-1}, tstop_k]$, $tstep = tstep_k$ and $tmax = RMAX \times tstep_k$, where $k=1$, .., N and default is $RMAX=10$.

Note: The statement `.PRINT` prints outputs as a table with the table step equal to the `.TRAN` parameter `tstep` for the first `.TRAN` statement syntax. In the case of the second syntax, with variable `tstep`, SmartSpice will print outputs at internal integration method time points. To print outputs at time points according to `.TRAN` variable `tstep`, use `.OPTIONS INTERP=1`.

`START=tstart`	Causes SmartSpice to store data from the Transient analysis starting at the particular time `tstart` in the simulation.

During an entire transient simulation, all voltage and current sources of the circuit maintain their Transient analysis values. If the parameter TRANOP or UIC is specified, then the SmartSpice Transient analysis is free of any “No Path to Ground” problems.

Examples

```
.TRAN 5NS 80NS 1NS 0.5NS
.TRAN 5NS 80NS CALLV SAVEV SWEEP WIDTH 10u 15u 1u
.TRAN 5NS 80NS TRANOP=50NS
.TRAN 0.1n 100n 1n 500n START=50n STORE=3
.TRAN 0.6n 23u 0.6n 26u 0.6n 50u reltol = { , 1e-4, }
```

In the first example, SmartSpice calculates the DC operating point and then performs a Transient analysis on the interval from 0 to 80ns, starting from the calculated operating point. It stores the calculated results on the interval from 1ns to 80ns. The maximum value of the internal time step is 0.5ns.

In the second example, SmartSpice calculates the DC operating point starting from the called operating point. SmartSpice then saves the newly calculated DC operating point and performs the Transient analysis. The maximum time step is 80ns/50. The analysis is performed for six values of the parameter label width: 10, 11, 12, 13, 14 and 15 microns.

In the third example, SmartSpice computes the transient operating point at 50ns, saves it, and performs the Transient analysis, storing outputs on the interval from 0 to 80ns.

In the fourth example, SmartSpice performs a Transient analysis to 100ns with the parameter, step=0.1ns, and a maximum timestep of 0.2ns, and from 100ns to 500ns with the parameter, step=1ns, and the maximum time step of 2ns. SmartSpice stores data from 50ns to the end of the simulation only, and stores each third data point.

In the fifth example, SmartSpice performs a Transient analysis to 23u with the default reltol, from 23u to 26u with reltol=1e-4 and from 26u to 50u with the default reltol.

When .TRAN and .OP (no arguments) Appear in the Same Deck

When the .OP statement is used on its own, its purpose is to perform an operating point analysis, and then print the results. When used in conjunction with a .TRAN statement, it is used in a slightly different way. Since the .TRAN statement already performs its own built-in operating point analysis, the .OP statement serves only to specify the time at which the operating-point information is to be printed.

Support for Parameters in .TRAN Statement Modified by .ST or .MODIF Statements

In previous versions of SmartSpice, parameters in .TRAN statement defined as expressions were not reevaluated if modified by .ST or .MODIF statements. Now tstart, tstop, tstep and tmax parameters can be defined as parameters/expressions, and easily modified by .ST or .MODIF statements.

Example

```
.PARAM trp=1n
.TRAN 1n 'trp*2'
.ST list trp 1n 2n 3n
```

In this example Transient analysis will be launched 3 times, each time with reevaluated analysis end time (2n, 4n and 6n).

Transient Ramping Algorithm

When SmartSpice fails to achieve convergence using the Newton-Raphson method, it is reasonable to use the Transient analysis method to compute steady state. The parameter TRANOP causes the steady state to be computed prior to the Transient analysis itself.

Syntax

```
.TRAN tstep tstop TRANOP= <top>
```

Where tstep is the time interval used to control the printing and plotting Transient analysis results, and the maximum step size; tstop is the end time of the steady state computation. By default, top=tstop.

SmartSpice calculates the steady state over the time interval from 0 to top and then performs the Transient analysis over the time interval from 0 to tstop.

The transient ramping is similar to the DC source stepping. At time point zero, SmartSpice sets all power supplies and all node voltages specified in .IC statements to zero. It then performs the Transient analysis. The power supplies and initial conditions are increased over the time interval from 0 to TR, which is less than tstop, and then are held constant from TR to top. SmartSpice calculates a proper TR value automatically.

The transient operating point calculated as a result of the steady state computation over the time interval from 0 to top is used as the initial point for the Transient analysis independent of how stable “steady state” is. It means that the DC operating point calculation is not performed at all.

Transient ramping is a more reliable way to solve the initial value problem (i.e., more reliable than the DC operating point calculation). However, it is usually more time consuming. The number of iterations needed for steady state computation depends on top, but not too much. In contrast, the stability of the steady state depends on top very much. To provide a stable steady state, it is reasonable to specify a large top value, which must be significantly greater than the largest time constant of the simulated circuit.

.UNPROBE (Save Unprobed Vectors For Transient Analysis)

SmartSpice provides a feature to save un-probed node waveforms with a reduced number of points for transient analysis. To activate this feature, specify .UNPROBE statement in the input deck. If .UNPROBE is specified without arguments, SmartSpice saves all nodes with fewer data points except .PROBE statement nodes. If nodes are specified in the .UNPROBE statement, SmartSpice will save only these nodes.

Syntax

```
.UNPROBE outvar1 <outvar2 ...>
```

outvar1...	Output variables.

Example

```
.unprobe
.probe v(1) v(2) v(3)
```

SmartSpice saves 1/100 data points for all nodes excluding v(1), v(2) and v(3). All data points for v(1), v(2) and v(3) will be stored in plot.

Note: The option RAWPTS will only be effective on vectors not specified in .UNPROBE statements.

.UNPROTECT (Make Deck Visible)

Syntax

```
.UNPROTECT|.UNPROT
```

Equivalent to `.OPTIONS BRIEF=0`.

.UNSAVE (Unsave Vectors)

Syntax

```
.UNSAVE outvar1 <outvar2 ...>
```

This statement causes SmartSpice to remove output variables `outvar1, outvar2, ...` from the list of signals that are saved during the simulation. For the output variable descriptions, see Chapter 6 Outputs.

Examples

```
.UNSAVE V(2) V(3) I(D1) I(L22)
.UNSAVE ID(M3) IB(Q4)
```

The first example instructs the program not to save the following signals during simulation: `V(2)`, `V(3)`, `I(D1)` and `I(L22)`.

In the second example, the drain current of MOSFET device `M3` and the base current of bipolar device `Q4` is removed from the list of signals saved during simulation.

.VEC (Digital Vector File)

The .VEC statement is used to specify all types of digital vectors (inputs, outputs and bidirectional vectors).

Syntax

```
.VEC 'vec_file'
```

vec_file	Digital vector file name.

The .VEC file must be a text file.

Example

```
.VEC 'digital.vec'
```

In this example, SmartSpice generates digital voltage sources for a digital.vec file from the current directory.

The digital vector file consists of three sections:

- Vector Pattern Definition
- Waveform Characteristics
- Tabular Data

Vector Pattern Definition

The Vector Pattern Definition section describes the digital signals by names, sizes and time parameters. All statements from this section can appear in any order (except the RADIX statement, which must be the first). All keywords are case insensitive.

The typical vector pattern definition section is:

```
RADIX  11
VNAME sermux1 sermux0
IO     ii
TUNIT  ns
```

The first line with RADIX defines two single-bit signals or vectors.

The second line with VNAME statement defines the signal names.

The third line with IO statement defines that this is the input signal.

The fourth line with TUNIT defines time unit for tabular data and waveform characteristics; in this case it's nanoseconds (ns).

Note: The statement with a RADIX keyword should be the first line in a vector file.

RADIX Statement

The RADIX statement defines the number of bits in the signal. The RADIX statement is intended to work with the values of the number of bits ranging from 1 to 4 (i.e., 1-Binary, 4-Hex).

Syntax

```
RADIX  valid_digits
```

valid_digits	Number of bits in signal.

# bits	Radix	Number system	Valid digits
1	2	Binary	0, 1
2	4	-	0 - 3
3	8	Octal	0 - 7
4	16	Hexadecimal	0 - F

Examples

```
RADIX  11111111 11111111
RADIX  41
```

The first line defines two 8-bit signals.

The second line defines two signals: the first is a 4-bit signal and the second is a 1-bit signal.

VNAME Statement

The VNAME statement defines the name of each signal.

Syntax

```
VNAME  name1 <name2 ...>
```

name1	Signal name.

Example

```
VNAME v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12 v13 v14 v15 v16
```

If VNAME statement is not defined, SmartSpice assigns a default name to each signal: v1, v2, v3 and so on.

If VNAME statement is defined twice or more, the last statement overrules the previous statement.

SmartSpice also supports bus notation. This notation looks as follows:

Syntax

```
VNAME  bus_name[start_index:end_index]
```

bus_name	Bus name.

start_index, end_index	Starting and ending index.

The opening and closing square brackets ([ ]) and the colon (:) are required.

The range of the bus must correlate with the RADIX statement.

Example

```
Bus[15:0]
```

This example specifies the bus with the name Bus, and the signals: Bus15, Bus14, Bus13, ... Bus0.

SmartSpice supports the nested grouping symbols such as: {}, <>, ().

Examples

```
Bus1<[3:0]>
Bus2[[3:0]]
```

The first example describes the bus with the name Bus1, and the signals: Bus1<3>, Bus1<2>, Bus1<1> and Bus1<0>.

The second example describes the bus with the name Bus2, and the signals: Bus2[3], Bus2[2], Bus2[1] and Bus2[0].

Example

```
RADIX  41
VNAME Bus[3:0] Signal
```

This example generates signals with the names Bus3, Bus2, Bus1, Bus0 and Signal.

Example

```
RADIX  44441
VNAME Bus[[15:0]] Signal[0:0].
```

This example generates signals with the name Bus[15], Bus[14], ..., Bus[1], Bus[0] and Signal0.

In case if there are duplicates in the input sources from .vec statement, SmartSpice outputs the next error message:

```
Error on line:
linenumberilename
.VEC internal error while creating sources from .vec.
Vector 'vectorname' is duplicated.
```

Example

1. A spice netlist contains the following statements:

```
...
.vec 'InputSourceDuplicate1.vec'
.vec 'InputSourceDuplicate2.vec'
...
```

2. .Vec file 'InputSourceDuplicate1.vec' contains the voltage source vname:

```
...
vname in<1>
...
```

3. .Vec file 'InputSourceDuplicate2.vec' contains the voltage source vname:

```
...
vname in<1>
...
```

In the previous example SmartSpice generates the following error message:

```
Error on line:
6 : .vec 'InputSourceDuplicate2.vec'
.VEC internal error while creating sources from .vec.
Vector 'in<1>' is duplicated.
```

IO Statement

Defines the type of each vector (signal). The line starts with the keyword IO, followed by a some vector's type definitions.

Syntax

```
IO  I|O|B|U <I|O|B|U...>
```

Command Argument	Description
I	Input signal
O	Output signal
B	Bidirectional signal
U	Unused signal

I	Input signal, which SmartSpice uses to stimulate the circuit.
O	Expected output vector, which SmartSpice compares with the simulated results.
B	Bidirectional vector, which can be set to the input or output vector during the simulation due to the controlling signal.
U	Unused vector, which SmartSpice ignores during simulation.

By default, SmartSpice assumes that all signals are the input signals.

Example

```
IO iiii i bbbb oo iii
```

ENABLE Statement

Specifies the controlling signal for bidirectional signals. All bidirectional signals require an ENABLE statement. The bidirectional vector becomes output when the controlling signal is at state 1 (or High). An error message will be displayed when the statement ENABLE is missed to activate the bidirectional signal.

Syntax

```
ENABLE  <~>Signal_Name Mask
```

Signal_Name	Controlling signal name, must be an input signal. To reverse the default control logic, start the control signal name with a tilde (~).
Mask	Defines the mask (and thereby signals to which the statement applies).

Example

```
radix 1111 1
io    iiii b
vname  IOPAD4  IOPAD3  IOPAD2  IOPAD1  IOPAD0
enable  IOPAD4   0000  1
```

In this example, the IOPAD0 is a bidirectional signal, as defined by IO statement. The ENABLE statement indicates that the IOPAD0 vector becomes output vector when IOPAD4 signal is 1.

OUT, OUTZ or RESISTANCE Statement

The OUT, OUTZ or RESISTANCE statements are the same and specify the output resistance for each input signal for which the mask applies.

Syntax

```
OUT <Output_resistance> Mask
```

Output_resistance	Output resistance for an input signal, default is 0.
Mask	Defines the mask (and thereby signals to which the statement applies). If you do not specify a mask, the output resistance value applies to all input signals.

By default, if you do not specify the output resistance of a signal in the OUT/OUTZ statement, SmartSpice assumes that the output resistance is zero.

Example

```
radix 1111
io    iiii
vname  IOPAD4  IOPAD3  IOPAD2  IOPAD1
OUT 30
OUT 250 1000
```

In this example, the first OUT statement creates a 30 ohms resistor, to place in series with all input signals: IOPAD4, IOPAD3, IOPAD2, IOPAD1. The second OUT statement sets the resistance to 250 ohms for vector IOPAD4.

TRIZ Statement

Specifies the output impedance, when the signal is in tristate, for each input signal for which the mask applies.

Syntax

```
TRIZ <Output_impedance> Mask
```

Output_impedance	Output impedance for an output signal, default is 1000M.
Mask	Defines the mask (and thereby signals to which the statement applies).

By default, if you do not specify the tristate impedance of a signal in the TRIZ statement, SmartSpice assumes 1000M.

TRIZ statements have no effect on the expected output signals.

Example

```
radix 1111
io    iiii
vname  IOPAD4  IOPAD3  IOPAD2  IOPAD1
TRIZ 30
TRIZ 250 1000
```

In this example, the first TRIZ statement sets the high impedance resistance globally to 30 ohms. The second statement increases the value to 250 ohms for the vector IOPAD4.

Waveform Characteristics

Defines various parameters for signal like the rise or fall time, the thresholds for high or low logic levels, delay time of signal.

ODELAY Statement

Defines the delay time of the output signal relative to the absolute time of each row in Tabular Data section.

Syntax

```
ODELAY X <Mask>
```

X	Defines the delay time for output signal.
Mask	Defines the mask (and thereby signals to which the statement applies).

If you do not specify the signal delays in a TDELAY, IDELAY or ODELAY statement, SmartSpice assumes zero.

VOH Statement

The VOH statement defines the logic-high voltage for each output signal to which the mask applies.

Syntax

```
VOH X <Mask>
```

X	Defines the logic-high voltage,

Mask	Defines the mask (and thereby signals to which the statement applies).

Default VOH value is 3.3 V.

Example

```
VOH 5.0
VOH 14.0 1100 1100
```

In this example, the first VOH statement sets the logic-high voltage to 5.0 V for all output signals, and the second statement sets the logic-high voltage to 14.0 V for 1, 2, 5 and 6 signals.

VOL Statement

Defines the logic-low voltage for each output signal to which the mask applies.

Syntax

```
VOL X <Mask>
```

X	Defines the logic-low voltage,
Mask	Defines the mask (and thereby signals to which the statement applies).

Default VOL value is 0.0 V.

Example

```
VOL 1.0
VOL 1.2 1100 1100
```

In this example the first VOL statement sets the logic-low voltage to 1.0 V for all output signals, and the second statement sets the logic-low voltage to 1.2 V for 1, 2, 5 and 6 signals.

VTH Statement

Specifies the logic threshold voltage for each output signal to which the mask applies. The threshold voltage determines the logic state of output signals for comparison with the expected output results.

Syntax

```
VTH X <Mask>
```

X	Defines the logic threshold voltage.
Mask	Defines the mask (and there signal to which the statement applies).

Default VTH value is 1.65 V.

VTH statement has no effect on the input signals.

Example

```
VTH 1.75
VTH 1.8 1100 1100
```

In this example the first statement VTH sets the logic threshold voltage to 1.75 V for all output signals, and the second statement sets the logic threshold voltage to 1.8 V for 1, 2, 5 and 6 signals.

TUNIT Statement

The TUNIT statement defines the time unit in the .VEC file for PERIOD, TDELAY, SLOPE, TRISE, TFALL and absolute time.

Syntax

```
TUNIT  <fs|ps|ns|us|ms>
```

Unit	Description
fs	femtosecond
ps	picosecond
ns	nanosecond
us	microsecond
ms	millisecond

Default is ns.

Example

```
TUNIT ns
5.0 0111 1101
20.0 0110 1100
35.0 0010 0101
```

TUNIT statement in this example describes the absolute times in the Tabular Data section as 5.0, 20.0 and 35.0 ns.

PERIOD Statement

The PERIOD statement defines the time interval for the Tabular Data section, and is not needed to specify the absolute time at every time point. In this case, the Tabular Data section contains only signal values without absolute time. The TUNIT statement defines the time unit of the PERIOD.

Syntax

```
PERIOD x
```

x	time interval.

Example

```
RADIX  11
PERIOD 10
```

```
0111 1101
0110 1100
0010 0101
```

In this example, the first line defines the tabular data (`0111 1101`) at time `0 ns`, the second line (`0110 1100`) at `10 ns`, and the third line (`0010 0101`) at `20 ns`.

Waveform Characteristics

Waveform Characteristics section defines various parameters for signals, such as the rise or fall time, the thresholds for high or low logic levels, and delay time of signal.

The typical Waveform Characteristics section is:

```
TRISE 0.8 1011
TFALL 0.5 0011
VIH   3.3 1111
VIL   0.0 0110
```

The Waveform Characteristics are based on a bit-mask. In this example, `TRISE` statement sets the signal rise time to `0.8 ns` for signals `1`, `3` and `4`; `TFALL` statement sets the signal fall time to `0.5 ns` for signals `3` and `4`; `VIH` statement sets the voltage for logic-high level for all signals; and `VIL` statement sets the voltage for logic-low level for signals `2` and `3`.

The next section describes all waveform characteristics and corresponding keywords.

TDELAY and IDELAY Statements

The `TDELAY` and `IDELAY` statements define the delay time of the input signal relative to the absolute time of each row in the Tabular Data section.

Syntax

```
TDELAY x <mask>
```

x	Defines the delay time.
mask	Defines the mask (and thereby signals to which the delay applies).

```
IDELAY x <mask>
```

x	Defines the delay time for the input signal.
mask	Defines the mask (and the signals to which the delay applies).

The statements start with a keyword, followed by a delay value (`x`) and then the `mask`. The `mask` defines the signals to which the delay applies. If the `mask` is omitted, the delay value applies to all signals.

Note: If you specify more than one `TDELAY` or `IDELAY` statement, the last statement overrules the previous statements. If you do not specify the mask in a `TDELAY` or `IDELAY` statements, the delay value applies to all signals.

Example

```
RADIX  11
IO     ii
VNAME v1 v2
TDELAY 1.2 10
IDELAY 3.6 01
```

In this example, the first TDELAY statement sets the delay time at 1.2 for v1, and the second statement IDELAY sets the delay time at 3.6 for v2.

SLOPE Statement

The SLOPE statement defines the rise/fall time for the input signal.

Syntax

```
SLOPE x <mask>
```

x	Defines the rise/fall time.
mask	Defines the mask (and the signals to which the rise/fall time applies).

Default is 0.1 ns.

Example

```
SLOPE  3.6
SLOPE  2.2 1100 1100
```

In this example, the first SLOPE statement sets the rise/fall time at 3.6 for all signals, and the second statement sets the rise/fall time at 2.2 for signals 1, 2, 5 and 6.

Note: If you specify more than one SLOPE statement, the last statement overrules the previous statements. If you do not specify the mask in the statement, the x value applies to all signals.

TRISE Statement

The TRISE statement defines the rise time for the input signal.

Syntax

```
TRISE x <mask>
```

x	Defines the rise time.
mask	Defines the mask (and the signals to which the rise time applies).

Default is 0.1 ns.

Example

```
TRISE  1.6
TRISE  5.2 1100 1100
```

In this example, the first TRISE statement sets the rise time at 1.6 for all signals, and the second statement sets the rise time at 5.2 for signals 1, 2, 5 and 6.

Note: If you specify more than one TRISE statement, the last statement overrules the previous statements. If you do not specify the mask in the statement, the x value applies to all signals.

TFALL Statement

The TFALL statement defines the fall time for the input signal.

Syntax

```
TFALL x <mask>
```

x	Defines the fall time.
mask	Defines the mask (and the signal to which the fall time applies).

Default is 0.1 ns.

Example

```
TFALL  7.6
TFALL  0.2 1100 1100
```

In this example, the first statement TFALL sets the fall time at 7.6 for all signals, and the second statement sets the fall time at 0.2 for signals 1, 2, 5 and 6.

Note: If you specify more than one TFALL statement, the last statement overrules the previous statements. If you do not specify the mask in the statement, the x value applies to all signals.

VIH Statement

The VIH statement defines the logic-high voltage for each input signal to which the mask applies.

Syntax

```
VIH x <mask>
```

x	Defines the logic-high voltage.
mask	Defines the mask (and the signal to which the statement applies).

Default is 3.3 V.

Example

```
VIH 5.0
VIH 14.0 1100 1100
```

In this example, the first VIH statement sets the logic-high voltage at 5.0 V for all signals, and the second statement sets the logic-high voltage at 14.0 V for signals 1, 2, 5 and 6.

Note: If you specify more than one VIH statement, the last statement overrules the previous statements. If you do not specify the mask in the statement, the x value applies to all signals.

VIL Statement

The VIL statement defines the logic-low voltage for each input signal to which the mask applies.

Syntax

```
VIL x <mask>
```

x	Defines the logic-low voltage.
mask	Defines the mask (and the signal to which the statement applies).

Default is 0.0 V.

Example

```
VIL 0.5
VIL 1.2 1100 1100
```

In this example, the first statement VIL sets the logic-low voltage at 0.5 V for all signals, and the second statement sets the logic-low voltage at 1.2 V for signals 1, 2, 5 and 6.

Note: If you specify more than one VIL statement, the last statement overrules the previous statements. If you do not specify the mask in the statement, the x value applies to all signals.

VREF Statement

The VREF statement defines the name of the second node for each voltage source to which the mask applies. The first node has the corresponding name, as in the VNAME statement.

Syntax

```
VREF nodename <mask>
```

nodename	Defines the name of the second node for voltage source.
mask	Defines the mask (and the signal to which the statement applies).

Default is 0.

Example

```
RADIX 41
VNAME BUS[[3:0]] Signal[0:0]
VREF  0
VREF a 01
```

In this example, SmartSpice creates the voltage source Signal0 with two nodes: Signal0 and a. For example, BUS[0] voltage source has two nodes: BUS[0] and 0.

Note: If you specify more than one VREF statement, the last statement overrules the previous statements. If you do not specify the mask in the statement, the x value applies to all signals.

TSKIP Statement

The TSKIP statement forces SmartSpice to ignore the first row (with absolute time) from the tablula data section.

Syntax

```
TSKIP
```

Example

```
radix 11
period 20
tskip
15.9 10
```

In this example, SmartSpice ignores the first field with the absolute time (15.9) and uses time from the period statement

Tabular Data

The Tabular Data section must be last in a .VEC file; it defines the values of the digital signals at specified times.

Syntax

```
<time1> signal1_value1 signal2_value1 ...
<time2> signal1_value1 signal2_value1 ...
...
```

timex	Defines the timepoint.
signaly_valuex	Defines the value for the specific signal at the specific timepoint. Signal values can have any of the legal states, described below.

Rows in the Tabular Data section must appear in chronological order.

If the PERIOD statement is defined in a .VEC file, the Tabular Data section omits the absolute times, like time1, time2, ...

Examples

```
5.0  1010
12.1 0100
23.8 1111
43.9 0010
```

In this example, SmartSpice creates four signals. The first signal (from the left) is 1 at 5.0ns, 0 at 12.1 ns, 1 at 23.8 ns, and 0 at 43.9 ns; the second signal is 0 at 5.0 ns, 1 at 12.1 ns,1 at 23.8 ns, 0 at 43.9 ns, etc (the third signal - 1011, the fourth signal - 0010).

```
RADIX 11
```

```
IO ii
VNAME A B
TUNIT ns
PERIOD 5
; start data section
11
00
10
01
```

In the second example, SmartSpice creates two signals. The first signal A (from the left) is 1 at 0 ns, 0 at 5 ns, 1 at 10 ns, and 0 at 15 ns; the second signal B is 1 at 0 ns, 0 at 5 ns, 0 at 10 ns, and 1 at 15 ns.

Input Stimuli

SmartSpice converts each input signal into a PWL voltage source and a series resistance.

The table below describes the legal states for input signals.

Input States	Description
0	Drive to ZERO (gnd)
1	Drive to ONE (vdd)
Z	Floating to HIGH impedance
X	Drive to ZERO (gnd)
L	Resistive drive to ZERO
H	Resistive drive to ONE
U	Drive to ZERO (gnd)

For 0, 1, X, U states, SmartSpice sets the resistance to zero.

For L or H states, the OUT/OUTZ statements define the resistance value.

For Z state, the TRIZ statement defines the resistance.

Comment Lines and Continuing a Line

Any line in a .VEC file that begins with a semicolon (;) or an asterisk (*) is a comment line.

Any line in a .VEC file that begins with plus sign (+) is a continuation from the previous line.

check_window

The check_window statement defines a time window over which output signals are checked.

Syntax

```
check_window start_offset stop_offset steady <mask>
```

or

```
check_window start_offset stop_offset steady period_time start_time
<mask>
```

Two syntax forms are controlled by the steady parameter. The only difference between statements behavior is how the window center is selected.

In statement 1 above, the window center is at each vector stop time. In statement 2 above, the first window center is at the `start_time` and checking is repeated every `period_time`. The unit of time for `start_offset`, `stop_offset`, `period_time` and `start_time` is nanosecond.

start_offset	Defines window start as t-start_offset, t is window center defined depending on syntax form
stop_offset	Defines window stop as t+stop_offset, t is window center defined depending on syntax form
steady	Steady flag. When specified as 0 or 2, comparison check passes as long as the output state ever reaches the expected state at any time within time window. When specified as 1 or 3, comparison check passes if the output state matches the expected state throughout the time window period. Values of 0 and 1 select first syntax form of the statement. Values 2 and 3 select second syntax form of the statement.
period_time	2nd syntax form, defines time period when checking is repeated.
start_time	2nd syntax form, defines first time window center.
mask	Optional parameter, defines mask to apply statement for select output signals. By default, statement applies to all output signals.

Checking is performed during transient analysis. In addition to the regular `.vec` output, if `check_window` fails, a warning is issued.

This warning contains info on the type of check, signal name and time window definition that failed the check.

Use `.OPTION CHECKWINDOWWARN` to print all warnings after the simulation (see CHECKWINDOWWARN=[0|1]).

Examples

```
check_window 5 5 0
```

Checks all output signals; the check passes if the output state ever reaches the expected state. Window duration is 10ns with center at vector stop time.

```
check_window 25 25 3 100 2000 00001111
```

Check masked output signals; the window duration is 50ns. The first window center is at 2us, checking is repeated every 100ns. Steady check is performed; the check passes if the output state matches the expected state throughout the whole window.

Digital Vector File (.VEC) Example

An example of a `.VEC` file:

```
******* RADIX statement NON-BINARY format *****
RADIX 123
****** VNAME FOR SIGNALS **********************
vname sermux_in3_7 sermux_in3_6 sermux_in3_5
+sermux_in3_4 sermux_in3_3 sermux_in3_2
```

```
;******* INPUT SIGNAL ************************
IO i ii iii

;********* TIME UNIT  ***********************
tunit ns

******** SIGNAL FORM *************************
slope  0.6  116 ; - 101110 -
trise  0.2  024 ; - 010100 -
tfall  0.7  127 ; - 110111 -

********** PERIOD ******************************
period 3

********** TDELAY ******************************
tdelay  10  015 ; - 001101--

********** VOLTAGE LEVEL ***********************
vih     1.2 111 ; - 101001--
vil     0.5 033 ; - 011011--

****** INITIAL PATTERN FOR SIGNALS **************
*                  765432
***********************************************
1 1 4        ; --- 101100 ---
0 3 6        ; --- 011110 ---
1 2 4        ; --- 110100 ---
0 0 6        ; --- 000110 ---
1 1 1        ; --- 101001 ---
```

Expected Output

SmartSpice converts each output signal into a `.DOUT` statement in the netlist. During simulation, SmartSpice compares the actual results with the expected output vector's value. If the states are different, an error message will be printed.

The table below describes the legal states for output signals.

Input states	Description
0	Expect ZERO
1	Expect ONE
Z	Expect HIGH impedance (don't care)
X	Don't care
U	Don't care

SmartSpice generates errors report for `.VEC` output signals in following manner:

```
Time           Signal     Simulated   Expected
====           ======     =========   ========
0.000000e+00   out1           x          1
0.000000e+00   out2           x          1
5.000000e-09   out1           x          0
5.000000e-09   out2           x          1
```

```
1.000000e-08   out1          0          1
1.000000e-08   out2          0          1
1.500000e-08   out1          x          0
```

SmartSpice supports parameters with some statements in the digital vector files (VEC).

Parameter Usage

1. In the following statements, the unit is `tunit`: `PERIOD`, `TDELAY`, `IDELAY`, `ODELAY`, `SLOPE`, `TRISE`, `TFALL`. You might not need to attach units to the parameter.

Example

```
In netlist:
  .param periodn = 10   ; Means 10 ns
In VEC file:
  tunit ns
  PERIOD periodn
```

2. In the following statements, the unit is ohms: `OUT`/`OUTZ`, `TRIZ`. The parameter definition for these statements should follow SmartSpice syntax.

Example

```
In netlist:
  .param myout = 11Meg
In VEC file:
  OUT myout
```

3. In the next statements, the unit is volts: `VIH`, `VIL`, `VTH`, `VOH`, `VOL`.

Example

```
In netlist:
  .param vhigh = 5 vlow = 1     ; Means 5V and 1V
In VEC file:
  VIH vhigh
  VIL vlow
```

Option CBC (Context Based Control)

If the option CBC is specified in the `.VEC` file, SmartSpice will define the direction of bidirectional signal depending on its values. A bidirectional signal is an input if its value is 0, 1 or Z. A bidirectional signal is an otput if its value is L, H, X or U .

Example

```
io    b

vih 3.0
vil 0

vname  IOPAD0

option cbc

; begin vectors
0    H     ; output  H
1500 H     ; output  H
1600 0     ; input   0
1700 0     ; input   0
1800 L     ; output  L
1900 L     ; output  L
```

```
2000 1     ; input   1
2100 1     ; input   1
```

.VERILOG (Include Verilog-A File)

Syntax

```
.verilog "filename" <module_name> <VerilogA-options>
```

module_name	Optional Verilog-A module name. Name shouldn't begin with the hyphen symbol '-'. If specified, the only that module is loaded to SmartSpice from Verilog-A source file 'filename'.
VerilogA-options	A concationation of one or more options starting with a hyphen '-' symbol and separated by spaces. For more details see Section 3.14 .OPTIONS (Option Specification) Verilog-A - Options for Verilog-A.

Use this statement to specify Verilog-A source file "filename". Several .verilog cards can be used in the same netlist.

The files with directory path in the filename are supported.

Note: For HSpice compatibility .HDL statement can be used instead of .verilog.

Example

```
.verilog "resistor.va"
.hdl "/temp/inverter.va"
.verilog "capacitor.va" capacitor cap
```

In the first case the file "resistor.va" from the current (netlist specific) directory will be used.

In the second case the file "inverter.va" from the directory "/temp" will be used.

.WCASE (Worst-Case Analysis)

Syntax

```
.WCASE DC|TRAN|AC out_var measure <RANGE (from, at)> <LIST>
+ <OUTPUT ALL> <HI|LOW> <VARY EACH|ALL|BOTH> <BY value%>
+ <DEVICES filter>
```

This statement performs a worst case analysis of the circuit. As a first step, the measurement is calculated for a nominal run. During the nominal run, all analyses specified in the input deck (.TRAN, .AC, .DC, .SNS, etc.) are performed. Then sensitivity is calculated for the measurement of each parameter that has a tolerance definition. Worst case is then calculated in accordance with the sensitivity. During all subsequent runs, only the analysis specified in the .WCASE statement is performed. An input deck can use only one .WCASE, .MC or .SNS statement.

DC|TRAN|AC	Type of analysis to be performed.
out_var	Name of output variable for measure.
measure	Type of measurement. Can be one of the following: • **YMAX:** Finds the greatest difference between the current waveform and the nominal waveform. • **MAX:** Finds the maximum value of the current waveform. • **MIN:** Finds the minimum value of the current waveform. • **RISE_CROSS (value):** Finds the argument at the first point above the threshold value. • **FALL_CROSS (value):** Finds the argument at the first point below the threshold value.
LIST	Prints the real parameter values that are changed by the worst case analysis.
RANGE (from, at)	Defines the interval where the measure value will be evaluated. An asterisk (*) can be used in place of low value or high value. In this case only one boundary will be effective.
OUTPUT ALL	Parameter for output control. When specified, all output generated by .PRINT, .PLOT, .MEAS, .PROBE, and so forth is permitted.
HI|LOW	Defines the direction for worst case calculation. By default, for YMAX and MAX the direction is HI, otherwise the default is LOW.

VARY EACH|ALL|BOTH	Defines which parameters will be changed in worst case analysis as specified in the .MODEL statement. If VARY EACH is used, then only parameters specified with EACH in the .MODEL statement will be changed. If VARY ALL is used, only parameters specified with ALL in the .MODEL statement will be changed. If VARY BOTH is used, all parameters with tolerance specifications will be changed.
BY value%	Defines sensitivity calculation. The model parameter values will vary by the specified percentage. By default, value% is equal to RELTOL.
DEVICES filter	Defines the filter used to determine what devices are included in worst case analysis. The variable filter is a character string that should match a device type. For example, to perform worst case analysis only on BJT devices and capacitors, define the filter as DEVICES QC. The filter can use alphabetical characters only.

Note: In worst case calculations, tolerance values are used. Default is VARY BOTH.

Examples

```
.WCASE TRAN V(1) FALL_CROSS (0.001) LIST OUTPUT ALL
.WCASE DC V(1,2) MAX LIST RANGE (*,5) VARY ALL BY 2 LOW DEVICES QR
```

The first example performs a nominal run, all runs for sensitivity calculation, and a final run to compute the worst case. For each analysis, the value of the first point below 0.001 is found. A list of model parameters that have been changed will be printed before each run. All output produced by other statements will be printed.

The second example performs the nominal run, then performs sensitivity analysis for MAX(V(1,2)) with respect to model parameters with the ALL tolerance in the .MODEL statement. For each run, the parameter value is increased by 5%. The parameters are changed for BJT devices and resistors only. For every run, the maximum of V(1,2) is calculated during the interval from swname (defined in the .DC statement) to 5. After sensitivity results have been obtained, SmartSpice performs a final run to calculate the worst case. The step occurs in a lower direction for MAX(V(1,2)).

.WIDTH (Set Width for .PRINT and .PLOT)

Syntax

```
.WIDTH OUT=outwidth
```

or

```
.WIDTH CO=outwidth
```

The `.WIDTH` statement sets width (in columns) of the output from `.PRINT` and `.PLOT` statements. The default width is `80`, and the minimum width is `40`.

Examples

```
.WIDTH OUT=60
.WIDTH OUT=120
.WIDTH CO=60
```

3.14 .OPTIONS (Option Specification)

Syntax

```
.OPT | .OPTION | .OPTIONS opt1 <=val> <opt2 <=val>...>
```

This statement allows the control and user options to be reset for specific simulation purposes. Any combination of the following options can be included in any order. If an option is specified with no value, then it will acquire a value of 1 (true, ON). The options divided into sense groups are described below.

Table 3-5 Control Options

Control Options		
General	**Output**	**Third Party Interface**
CFLFLAG	ACCT	CDS
CKPTCLOCK	ACCT_INTERVAL	CSDF
DISTRIBUTION	ACOUT	EPIC
EXPERT	BRIEF	PSF
EXPMAX	CHECKWINDOWWARN	SDA
MAXVOLT	CONTEXTCMDS	WSF
MAXAMP	DONTPRINTTOP	
MFILEFORMAT	FILTERCURRENT	
MMSMOOTH	FORMAT	
MMSMOOTHEPS	FSDB_VPRBTOL	
NORELOAD	FSDB_IPRBTOL	
OPTIMIZER	FSDB_VERSION	
PARHIER	INGOLD	
PREVIEWSTIMULUS	INTERP	
PROBELET	IPLOT_ONE	
RESMIN	LIST	
RUNMODE	MATRSTAT	
SINGULARSUPPLYRES	MEASDGT	
SEARCH	MEASRAW	
STOPERR	MERGE_AC_NET	
STOPONFATALERRORS	NODE	
TEMP	NODECK	
TNOM	NOMOD	
USEDEGREES	NOPAGE	

Control Options		
ZRES	NUMDGT	
SPEEDPLOT	OPLST	
	OPTS	
	POST	
	POSTRAWPTS	
	POST_VERSION	
	PRINTBISTABLENODES	
	PROBE	
	PRINT_MAGNITUDE	
	RAWPTS	
	SAVEMEASUREOUT	
	SAVEMEXCEL	
	SAVEMFILES	
	SPLITMEASURE	
	SUPPRESSOUTRAW	
	UNPROBERATIO	
	UNWRAP	
	VIEWER	
	WIDTH	
	WILDCARDSTAT	
Specific Cases	**SOA - Safe Operating Area**	
FLATTENED_DCMONTE	MAXWARNS	
IGNOREMCSTEPERROR	WARN	
LIS_NEW		
MODMONTE		
NPORTALG		
RMNODELESS2		
SR_ABSTOL		
SR_RELTOL		
SR_VNTOL		
SR_LTERATIO		
SR_LVLTIM		

Control Options		
SWEEPMONTE		

Control Options

General

CKPTCLOCK	Controls the creation of checkpoint files. An additional option can be set in the input deck:

Example

```
.OPTION CKPTCLOCK=period,
```

Where `period` is the regular time interval in seconds when the checkpoint files will be saved. This option will work for all transient analysis statements, except those containing the `ckptperiod` variable in the `.TRAN` statement.

DISTRIBUTION=val	Resets the default distribution for Monte-Carlo deviation (either `GAUSS`, `UNIFORM`, or the name of a user-defined distribution). Default is `UNIFORM`.
EXPERT<=val>	Provides information about nonconvergent nodes and devices; activates warning messages about NaN, and nonphysical negative values in active device models; and creates convergency plots. `EXPERT=777` makes this information more detailed. `EXPERT=779` activates warning messages about BSIM3v3 and BSIM3H model parameters, which are out of range. Default is `OFF`.
EXPMAX=val	Use this option to specify the largest exponent that you can use for an exponential function. Default value is 80.
MAXVOLT, MAXAMP	Protects Newton-Raphson method convergency to a “non-realistic” solution. If the converged solution has a maximum of absolute node voltage values greater than specified in `MAXVOLT`, or has a maximum of absolute branch current values greater than specified in `MAXAMP`, then the converged solution is not accept. Default is `0`. Additionally, when nonzero `MAXVOLT` or `MAXAMP` values are specified, the option parameter `NEWTOL` has the value of `ON`. It means that additional iterations will be performed while convergency will be reached at two consecutive iterations.
MFILEFORMAT	`mfileformat=1` - SmartSpice generates measure file with 7 columns per line (default). `mfileformat=2` - SmartSpice generates HSPICE-compatible measure file with 4 columns per line. Default is `1`.

MMSMOOTH	This option makes the functions ABS, SGN, MIN and MAX similar to the functions SMABS, SMSGN, SMMIN and SMMAX with the smoothing coefficient eps=MMSMOOTHEPS.
MMSMOOTHEPS=val	Specifies the smoothing coefficient eps, used to make ABS, SGN, MIN, MAX functions derivable. When the option MMSMOOTH is specified, by default MMSMOOTH-EPS is 1.0e-3, otherwise MMSMOOTHEPS is 0. When this option val is not equivalent to 0, the option MMSMOOTH will be set.
NORELOAD	Suppresses unnecessary re-evaluation (re-loading) of parameters between run. Default is OFF.
OPTIMIZER	Selects the algorithm to be used in .MODIF optimization. The default is LM (Levenberg-Marquardt). The other settings include new global optimization algorithms: HJ (Hooke-Jeeves), SA (simulated annealing), DE (differential evolution), PT (parallel tempering) and GA (genetic algorithm). See Chapter 18 Optimizer for further details.
PARAMETRIC_DATA_IN_SAVEBIAS	When this option is set 1 (ON), the .SAVEBIAS statement will save expanded savebias files. This option is not set by default.
PARHIER	The PARHIER option allows you to specify which parameter value prevails when parameters with the same name are defined at different levels of the design hierarchy.

Syntax

```
.OPTIONS PARHIER = < GLOBAL | LOCAL | CALIBRE | DRACULA >
```

If .OPTIONS PARHIER = GLOBAL we have the next parameter passing order:

- Global parameters of the circuit.
- Local arguments of the subcircuit (X call line parameters).
- Local parameters of the subcircuit (.PARAM).
- Local arguments of the .SUBCKT statement.

If .OPTIONS PARHIER = LOCAL we have the next parameter passing order:

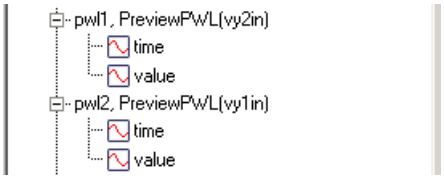
- Local arguments of the subcircuit (X call line parameters).
- Local parameters of the subcircuit (.PARAM).
- Local arguments of the .SUBCKT statement.
- Global parameters of the circuit.

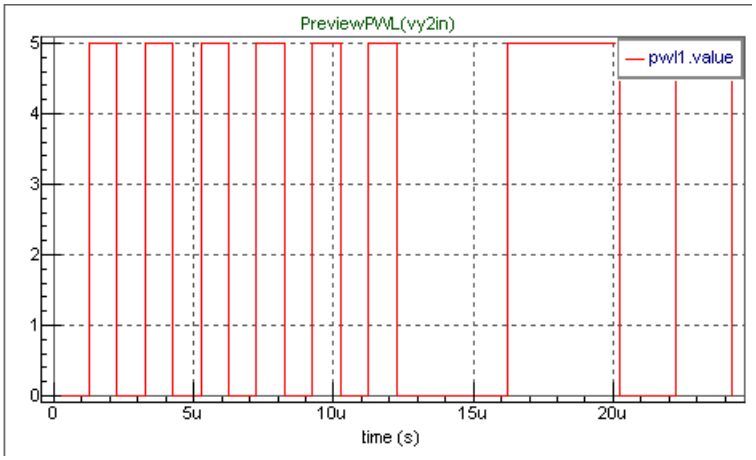
The default setting is GLOBAL.

PARHIER=CALIBRE (DRACULA)	Calibre/Dracula parameter scoping rules. SmartSpice implements the next parameter passing order when one of these options is specified: • Global parameters of the circuit. • Local arguments of the subcircuit call. These arguments will not be overwritten by any parameters with same names at higher level of hierarchy. • Local parameters of the subcircuit (.PARAM). • Local arguments of the .SUBCKT statement.
PREVIEWSTIMULUS	Enables SmartSpice to plot the piecewise linear pairs information (PWL) for a V-source and display the corresponding time dependant waveforms. SmartView will be invoked right after the input deck is sourced. This is useful to check circuit input stimulus before doing a long simulation.

Example

```
.OPTIONS previewstimulus
Vy2in 14 0 pwl (0 0 1.25e-06 0 1.252e-06 5 2.25e-06 5
+ 2.252e-06 0 3.25e-06 0 3.252e-06 5 4.25e-06 5
+ 4.252e-06 0 5.25e-06 0 5.252e-06 5 6.25e-06 5
+ 6.252e-06 0 7.25e-06 0 7.252e-06 5 8.25e-06 5
+ 8.252e-06 0 9.25e-06 0 9.252e-06 5 1.025e-05 5
+ 1.0252e-05 0 1.125e-05 0 1.1252e-05 5 1.225e-05 5
+ 1.2252e-05 0 1.625e-05 0 1.6252e-05 5 2.025e-05 5
+ 2.0252e-05 0 2.225e-05 0 2.2252e-05 5 2.425e-05 5
+ 2.4252e-05 0 )
```





PROBELET	Implemented to disable the writing of .LET statement vectors into a raw file.

Syntax

```
.option probelet=value
```

- value=1: SmartSpice will write .LET statement vectors into a raw file.
- value=0: .LET statement vectors will not be written into a raw file.

RESMIN=val	Specifies the minimum resistance value for all resistors, including parasitic and inductive resistances. Default is 1e-5 (ohm) (default=1e-3 in TMI mode with SmartSpice version ≥ 4.9.17). Range is from 1e-15 to 10 ohm. If the resistance is less than the option RESMIN, it will be reset to RESMIN value and a warning message will be displayed.
RUNMODE	Sets the group of related options.
SINGULARSUPPLYRES	The option SINGULARSUPPLYRES specifies the resistor value which terminates the floating node of a voltage source to ground. Default is 1e6.

Example

```
.option SINGULARSUPPLYRES=1e7
```

SEARCH	The string (which should be enclosed in quotes if it contains more than one word) following this option will be interpreted as a list of directories to be searched by SmartSpice. The existing search path is overwritten unless the string begins with a plus sign (+), in which case the new values are prepended to the existing list. Default is ~.

When including a file through the .INCLUDE statement, or loading a library of model parameters through a .LIB statement, by default, SmartSpice will look for these files in the current working directory (./), or in your home directory.

To modify this behavior, or extend it to more directories, the .OPTIONS SEARCH is appropriate. Depending on the syntax used, you can either override the previously set search path, or extend it to include more directories. For the new path to take effect, it must be specified before the .INCLUDE or .LIB statements.

The syntax for the .OPTIONS SEARCH is:

```
.OPT | OPTION | OPTIONS  <other_options>
+       SEARCH<=><+><path | “<+> <path1> <path2> ... “>
+       <other_options>
```

If the plus sign (+) is specified at the beginning of a path, the path is appended to the current search path. Otherwise, the new path overrides the previously set search path. If you prefer

that the path from .OPTIONS SEARCH is always appended to the current search path, with or without the plus sign, the you need to set the SmartSpice variable search_always_prepend to the value true in the file SmartSpice.ini. Note that it cannot be reliably set within an input deck, since its value must be known as the desk is being read, and options are not generally evaluated until afterwards.

A path specified with .OPTIONS SEARCH is used not only to look for files included with .INCLUDE or .LIB statements, but also to load subcircuit definitions that have been either explicitly defined by you, or included from a library (with .INCLUDE or .LIB statements). In the case of undefined subcircuits, SmartSpice will make up a filename from the subcircuit name with a .inc extension. It will then try to locate such a file in the search path specified as an option (default search path is /). If such a file is located, SmartSpice will load the subcircuit definition from it. Note that only the subcircuit being sought is loaded. Any other definition in the same file is ignored. In the case of a model definition within the subcircuit definition, the model is also loaded at the same time. This behavior is recursive; if the newly loaded subcircuit definition uses an undefined subcircuit, SmartSpice will also try to locate its definition in a separate file. SmartSpice will not attempt to use a subcircuit definition provided by the user in the case of a subcircuit call from within an undefined subcircuit. Instead, it will always look for a definition in a separate file. This is to reduce the potential conflict or name clash between user-defined subcircuits and library subcircuits.

STOPONFATALERRORS (STOPERR)	Stops SmartSpice from performing a simulation if errors are found while parsing the input deck, and overrides the stoponfatalerrors variable (see Chapter 5 Commands) for the circuit which contains this parameter. The following is the integer value behavior: • 0: SmartSpice will not perform any checks. • 1: All checks will be performed: netlist, post-processor commands, floating nodes[a]. • 2: Only netlist and floating node checks will be performed.

a.Checks for floating nodes could be suppressed with the floatingnodeiserror command.

TEMP=val	Temperature in degrees C, at which the simulation will be performed. TEMP is the temperature at which the circuit is to be simulated. If this is not specified, then the default value of 27°C is used, which happens to be the same as the default value for TNOM. The effect of the device at the circuit temperature is then calculated as the combination of the model at it's TNOM temperature, plus the effect of the temperature difference between these 2 values of TNOM and TEMP.

TNOM=val	Resets the nominal temperature. Default is 27 °C. TNOM is the temperature at which the device model was extracted. This can be set in the .MODEL card or in the .OPTIONS statement. If TNOM is set in the .MODEL card, then any circuit instance that calls this model card will use this temperature setting. For any .MODEL card that does not contain a TNOM setting, the .OPTIONS value will be used across all the remaining device models. If TNOM is not set in either of these two cases, then a default value of 300K is used.
USEDEGREES	Enables SmartSpice to overrride the value set by variable USEDEGREES. • USEDEGREES=true or 1: the angles and phases are output in degrees. • USEDEGREES=false or 0: the angles and phases are output in radians. Default is true or 1.

Examples

```
.OPTIONS  EXPERT  SEARCH  MyDirectory TNOM=25
.OPTIONS  SEARCH +LibDir  NORELOAD  USEDEGREES
.OPT  EXPERT=777  SEARCH=”+Dir1  Dir2  Dir3”  DISTRIBUTION=MyDis-
trib
```

ZRES=val	Defines the resistance value of internal resistors in hi-impedance state for PWLZ independent voltage sources. Default is 1e+12 ohm(=1/GMIN).
SPEEDPLOT	Allows you to speed up simulation of the DC, AC or transient analyses by reusing previously created SmartSpice structures (plot). Only one plot will be created, the waveforms will not be preserved. All measure results will be printed and preserved in the .meas plot. The raw files will contain only the data from the last sweep step and all measure results, if there is no .probe statements in the netlist. Default is off.
CFLFLAG=[0|1]	Allow you to speedup simulation by using precompiled binaries for the instance runtime expressions. SmartSpice will automatically generate source files, compile them and load precompiled binaries library.

Output

ACCT<=val>	Causes SmartSpice to print runtime statistics at the end of all analyses. The option ACCT=2 turns on a more detailed performance reporting mechanism, which prints the report containing two sections. In the first section, called **Total Analysis**, information is provided on the total time spent in several key areas of simulation. Following that is a section called **Device**, in which the following information is given for each device type: total number of devices in the deck, total time spent in setting up the devices, computing their temperature-related parameters, and simulating them. If the option ACCT=3 is used in -sb mode, the information about starting/finishing DCOP, DC, AC or TRAN analyses will be printed immediately. These numbers are expressed in terms of total time taken (seconds), and also as a percentage of the time taken to perform the simulation. Default is OFF.
ACCT_INTERVAL	Reports on an interval as a percentage of total simulation time. Default is 10. For example: `.OPTION ACCT_INTERVAL=val`

ACOUT=0	Set for HSPICE compatibility. When SmartSpice is running in hspice mode and the option .OPTIONS ACOUT=0 is set, vdb macro is redefined.
	`hspice mode: vdb(x,y) = vdb(x) - vdb(y)`
	`hspice mode (.OPTION ACOUT=0): vdb(x,y) = db(v(x) - v(y))`
	If ACOUT=1, then in regular SmartSpice mode vdb(x,y) macro is redefined to the default HSpice mode definition:
	`vdb(x,y) = vdb(x) - vdb(y)`

BRIEF<=val>	When BRIEF=1, the input deck will not be echoed to the output file (default behavior is to echo) or in the file listing (interactive mode only). All lines up to and including the .OPTIONS statement setting the option BRIEF=1 will be printed; all lines following that statement will be omitted. Default is 0 (OFF).
CHECKWINDOWWARN=[0|1]	Allows print warning messages from the check_window statement after the simulation. Default is 0 (OFF). See Section .VEC (Digital Vector File).
CONTEXTCMDS<=val>	Causes SmartSpice to use context-dependent syntax for output and post-processing statements. In the case of context-dependent syntax, the particular output statements are associated with only the above located analysis statement. Default is OFF. If the -hspice flag is specified on the command line, the default is ON, and it specifies CONTEXTCMDS=0 so you can turn off the context-dependent syntax.
DONTPRINTOP	Disables printing OP information.

FILTERCURRENT=val	When FILTERCURRENT=1, SmartSpice applies the filter on the postprocessing of any current. When absolute value of the current is less than ABSTOL * 1e-5 then the current value is considered to be equal to 0. The corresponding variable FILTERCURRENT has also been implemented. Default is 1.
FORMAT	Provides a compact format for .MEASURE output results. Default is ON. To provide an extended format, specify FORMAT=0.
FSDB_VPRBTOL=val	Voltage plot resolution control (tolerance) during writing to FSDB binary raw file. Due to the fact that FSDB format stores value changes of signal, specified value serves as a tolerance below which value changes are ignored hence controlling the resolution. Using FSDB_VPRBTOL=val (where val>=0) causes voltage plot data to be stored only if signal difference between previously written point and currently accessed one is equal to or greater than specified tolerance val, otherwise ignored. For example: FSDB_VPRBTOL=1.2e-6 Reducing tolerance ends up in increased final file size and vice versa.
FSDB_IPRBTOL=val	Current plot resolution control (tolerance) during writing to FSDB binary raw file. Due to the fact that FSDB format stores value changes of signal, specified value serves as a tolerance below which value changes are ignored hence controlling the resolution. Using FSDB_IPRBTOL=val (where val>=0) causes current plot data to be stored only if signal difference between previously written point and currently accessed one is equal to or greater than specified tolerance val, otherwise ignored. For example: FSDB_IPRBTOL=2.3e-9 Reducing tolerance ends up in increased final file size and vice versa.
FSDB_VERSION=val	Version of SpringSoft FSDB library to be loaded for FSDB file generation. Possible values are: FSDB_VERSION=1 (FSDB lib v.4.1) FSDB_VERSION=2 (FSDB lib v.4.2) FSDB_VERSION=3 (FSDB lib v.4.3) FSDB_VERSION=5 (FSDB lib v.5.0)

INGOLD=val	Resets the printout format. INGOLD=0 sets engineering format, exponents are expressed as a single character. INGOLD=1 sets combined fixed and exponential format. Fixed format for numbers 0.1-999. Exponential for 0.1 > number < 999. INGOLD=2 sets an exclusive exponential format (SmartSpice style). The default value for INGOLD is used as the value of the option print_format. Default is 2 in SmartSpice, and 0 in HSPICE (sets by -hspice flag).

Table 3-6 Single Character Expressions

Character	Multiplier
G	*1.e+9*
X	*1.e+6*
K	*1.e+3*
M	*1.e-3*
U	*1.e-6*
N	*1.e-9*
P	*1.e-12*
F	*1.e-15*

INTERP=val	Using INTERP=n (where *n>1*) causes data to be stored every n-th timepoint for Transient analysis (plus at every breakpoint at which input voltage and current sources have discontinuities). Using INTERP=1 in conjunction with .TRAN tstep tstop <tstart>... causes the vectors to be stored starting at time *tstart*, with time resolution *tstep*, and vector interpolation will be performed at time points *tstart*, *tstart+tstep*, *tstart+2tstep*, ... Using INTERP=1 in conjunction with .TRAN tstep1 tstop1 <tstep2 tstop2 ... tstepN tstopN> <START=tstart> causes the vectors to be stored starting at time *tstart*, with time resolution $tstep_k$ at each specified time interval $[tstop_{k-1}, tstop_k]$, where *k=1*, ..., N. Default is OFF.
IPLOT_ONE	Interactive plots of different sweep steps, and different analyses all appear on one graph. Default is OFF.
LIST	This option in conjunction with the .TRAN statement causes the summary of circuit elements (devices) to be printed exactly as in the .OP statement. Default is OFF.
MATRSTAT	Prints matrix solution statistic information. Default is OFF.

MEASDGT=val	Allows specification of the number of digits in the output from .MEASURE statements. Default is to use the value of options NUMDGT.
MEASRAW=val	Turn ON/OFF to save the .MEASURE results in the output rawfile. Default is 1(ON).
MERGE_AC_NET	Combines AC and NET analyses result vectors into one AC plot in -hspice mode. Default is OFF.
NODE	Prints summary of connections (node table). Default is OFF.

SmartSpice saves .MEASURE results in the output rawfile depending on how the option SAVEMEASURES is set, which has default value TRUE (i.e. measurement results are always saved by default). The option MEASRAW allows you to turn OFF this feature by specifying the statement .OPTIONS MEASRAW=0. If the option SAVEMEASURES is set to FALSE, the .OPTION MEASRAW=1 statement saves the .MEASURE results in the output rawfile.

The Element Node Table contains the number of nodes (column #), node names (column **Nodes**) and connections (column **Elements**) when using the NODE option or nodelist command.

Element Node Table

```
-------------------------------
 #      Nodes   Elements
 -----  -------  ------------------------------------------------------------------------------------
 1      0        q.x3.q4:S     q.x3.q3:S     q.x2.q2:S   q.x1.q2:S    vgoal3:-
                 vcc:-         vee:-         vin:-       rgoal3:-     r.x2.rs2:-
 2      1        vin:+         r.x1.rs2:-
 3      3        q.x1.q2:C     load:+        rc1:+
 4      8        vcc:+         rc2:-         rc1:-       rbias:-
 5      vee      q.x3.q4:E     q.x3.q3:E     vee:+
 6      vgoal3   vgoal3:+      rgoal3:+
 7      x1.25    q.x1.q2:B     r.x1.rs2:+
 8      x2.25    q.x2.q2:B     r.x2.rs2:+
```

NODECK	Suppresses listing of netlist in out-file for the batchmode. It has a higher priority than .OPTIONS BRIEF=0 and the .PROTECT statement. Default is OFF.
NOMOD	Suppresses model parameters and temperature update values printing. Default is OFF.
NOPAGE	Suppresses paging and banner for each major section of the output. Default is OFF.

NUMDGT=val	Resets the number of digits printed for output variable values and .MEASURE output results. Default is 4.
OPTLST<=1>	Prints additional bisection result information. Default is OPTLST=0.
OPTS	Prints all option values. Default is OFF.

SmartSpice supports three modes for .OPTIONS OPTS statement.

Syntax

```
OPTS or OPTS=1
OPTS=2
OPTS=10
```

- If OPTS or OPTS=1 are present in the input deck, SmartSpice prints all option variable values.
- If OPTS=2, SmartSpice prints all changed (from default) option values. Exception: RELTOL.
- If OPTS=10, SmartSpice prints all option values from the parsed input deck.

Example

```
.OPTIONS OPTS=2
.OPTIONS ACCT ACCURATE=2

------------------------------------------------------------------------

        ***  CHANGED OPTIONS  ***

------------------------------------------------------------------------

 GENERAL OPTIONS -

acct              =            1

opts              =            2

TRANSIENT ANALYSIS OPTIONS -

abstol            =         1e-12

accurate          =            2

fft_accurate

trtol             =          3.5

vntol             =         1e-06

itl41             =           10

method            =         GEAR

------------------------------------------------------------------------
```

PARAMETRIC_DATA_IN_SAVEBIAS	When this option is set (1), the .SAVEBIAS statement will save expanded savebias files. This option is not set by default.

POST<=val>	Automatically generates a rawfile without using the -r command-line option, and also causes the results to appear as vectors in the **Plot Vectors** window. If the POST option is used (and the -r option is not used on the command line) and the PROBE option is omitted, all node voltages and independent currents will be saved in rawfile and will appear as vectors in the **Plot Vectors** window, in addition to any variables appearing in output statements. The list below shows what values generate the corresponding rawfile format:

The following values (val) generate the corresponding rawfile format:

- **POST=1:** binary
- **POST=2:** ascii
- **POST=3:** single precision binary
- **POST=10:** csdf
- **POST=11:** excel
- **POST=12:** hspice_binary
- **POST=13:** hspice_ascii
- **POST=14:** ssf (Silvaco Standard Structure File)
- **POST=15:** data (Silvaco Data Format)
- **POST=16:** tabular
- **POST=17:** isdb (SimWave ISDB Format)
- **POST=18:** psf (Cadence PSF ASCII Format)
- **POST=19:** psf_binary (Cadence PSF binary Format)
- **POST=20:** fsdb (SpringSoft FSDB binary Format
- **POST=21:** wsf (Cadence WSF ASCII Format)
- **POST=22:** wsf_binary (Cadence WSF binary Format).

Note: FSDB format is available for transient analysis data only. Time scale has been changed from 1ps to 1fs. Waveform current saving has been implemented.

Binary data takes up less memory than ASCII and is the preferred type. This format has been made portable between all supported platforms, and is transparent.

If the POST option is used without a value, then a value of 1 is assumed. If the POST option is not specified, the rawfile has a binary format. The name of the rawfile is derived from the name of the current input deck with a -.raw extension.

The .MEASURE results rawfile is always generated in ASCII format, independent of the POST value.

POSTRAWPTS(PRPTS)=0	Disable (OFF) post-processing when RAWPTS=val is specified. Default is 1 (ON).

POST_VERSION=val	Sets the post-processing output version. Default is POST_VERSION=9601. The following are alternatives for this value: • val=9007 truncates the node name in post-processor output file to be no longer than 16 characters. • val=9601 sets the node name length for the output file to be consistent with the input restrictions (1024 characters). • val=2001 sets the post-processing output version with new output file header, which includes the right number of output variables rather than **** when the number exceeds 9999, and outputs more accurate ASCII (13 digits) and binary (double) results.
PRINTBISTABLENODES	Forces printing of bistable nodes during the Operating Point. Bistable nodes may cause unnecessary excessive Newton iterations and incorrect results. SmartSpice automatically uses .nodset on detected nodes. During transient analysis, .nodeset is removed.
PRINT_MAGNITUDE	Prints complex vector as magnitude in -hspice mode. Default is OFF.

Example

```
.ac lin 1 1000k 1001k sweep DATA=wl
.net v(gate) vds rout=50 rin=50

.print ac vg=par('g') cgx=par('-y21(i)/1000000/2/3.14')
.print ac S11(r) S11(i)
```

By deafult SmartSpice prints out real and imaginary parts of complex vector.

```
-------------------------------------------------------------------
Index     frequency          vg
-------------------------------------------------------------------
0        1.0000000e+006     1.1000000e+000     0.0000000e+000
```

In -hspice mode when option print_magnitude is specified vector magnitude will be printed.

```
 frequency          vg                cgx

 1.0000000e+006     1.1000000e+000    1.5041386e-014
```

PROBE	Limits the output variables saved in the rawfile to those specified in output statements (.SAVE, .PRINT, .PROBE, .LET, etc). This option is only active if the POST option is used, and the -r option is not used on the command line. Default is OFF. SmartSpice sets the option PROBE to 1 (ON) under TURBO mode to limit the output variables saved in rawfile. To disable this feature, .option PROBE=0 should be directly specified in the input deck.
RAWPTS=val	Affects the way Transient analysis results are dumped into rawfile in batch mode. SmartSpice dumps data into the file every RAWPTS timepoint. In this case, SmartSpice will save only the last RAWPTS timepoint in internal memory. The RAWPTS option is typically used when a large amount of output data is expected. Default is infinite. SmartSpice sets the option RAWPTS by default to 500 if the option POST is given.
SAVEMEASUREOUT	Moves a .MEASURE statement result from the common output stream into a separate file. The measurement results are saved in a file with 'measure.out' extension. The results are saved in the same format as they are displayed in GUI mode.
SAVEMEXCEL	Works with the SAVEMFILES option. If both SAVEMFILES and SAVEMEXCEL are specified, the .MEASURE statement result is output into a separate file with 'measure.csv' extension. Data in the file is saved in CSV (comma separated values) format. If the option SAVEMEXCEL or SAVEMEXCEL=1 is specified SmartSpice generates separate measure files for each .ALTER block in the netlist. When SAVEMEXCEL=2 is specified data from each .ALTER segment will be written into a single measure file.
SAVEMFILES	Saves .MEASURE results in a file with the extension of .mt, .ma or .ms which is associated to Transient, AC and DC analysis, respectively (when -hspice command line switch is not used). If the option SAVEMFILES or SAVEMFILES=1 is specified, SmartSpice generates .meas files in an HSPICE format. These files have different extensions depending on the analysis used (.mt* for transient, .ms* for AC, etc.). When SAVEMFILES=2 is specified, the .meas files are generated in SmartSpice format with extension .measure.raw*. In either case, if this is combined with the option POST then measurement data is also written to the output file. Default is OFF.

SPLITMEASURE	Setting this option to 1 causes a splitting of the meas section into separate parts; like measTYPE, where TYPE is one of the following AC, DC, FFT, FOUR, NET, NOISE, PZ or TRAN. Default is 0 (OFF).
SUPPRESSOUTRAW	Suppresses output and raw files for the child processes for the command option -mp. Default is 0 (off).
UNPROBERATIO	Specifies an un-probed nodes ratio. Default is 100. It means that SmartSpice saves 1/100 data points for un-probed nodes.

Syntax

```
.OPTION UNPROBERATIO=ratio_value
```

ratio_value	un-probed nodes ratio.

SmartSpice generates two sections in raw file: in first section it saves all probed nodes; in the second, all un-probed nodes.

In -hspice mode, SmartSpice writes two .tr files: one with un-sampled data (for example .tr0); the other with sampled data (for example .tr0p).

Example

```
.OPTION UNPROBERATIO=50
.unprobe v(4) v(5)
.probe v(1) v(2) v(3)
```

SmartSpice saves 1/50 data points for v(4) and v(5) nodes, and all data points for v(1), v(2) and v(3).

UNWRAP	Displays phase results for AC analysis in higher value rather than wrapped to a base 360 degree interval.

Syntax

```
.OPTION UNWRAP=0|1
```

This option allows the displaying of the phase results for AC analysis in unwrapped form (continuous phase plot). The unwrapped phase results are used to compute group delay, even if you do not set UNWRAP. By default, the wrapped phase value in the base interval of -180° to +180° is used. The convention is to normalize the phase output from -180° to +180°. Therefore, a phase of -185° is the same as a phase of +175°. Default is UNWRAP=0 if the option is not specified in the netlist, and UNWRAP=1 if the option name is specified without a corresponding value in the netlist.

Below is an example to illustrate how the phase wraps to the normalized base interval.

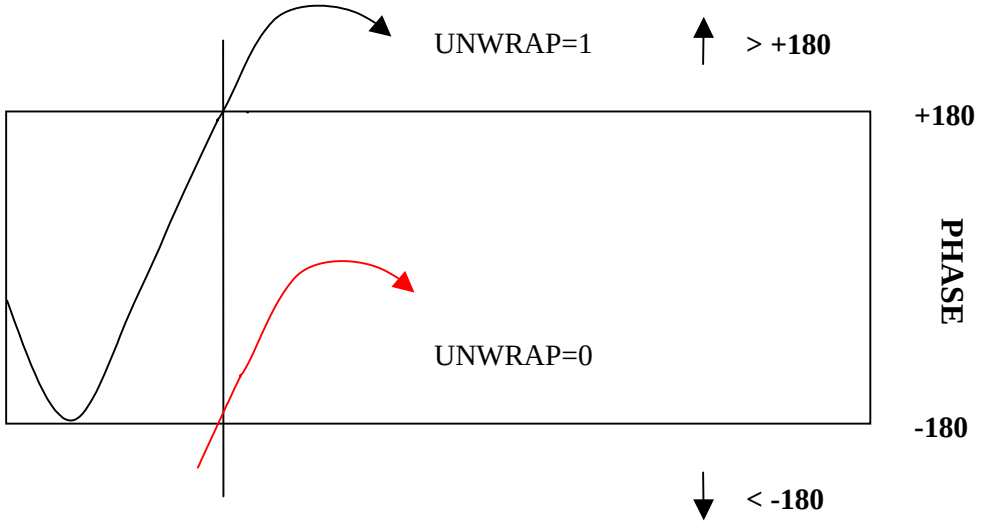


Figure 3-14 Default Method (without wrap)

Example

```
Phase=+174.8766 If UNWRAP=0
Phase=-185.1234 If UNWRAP=1
```

Internal Calculation

The phase value goes beyond -180, then it wraps to give a positive value.

If the phase value is -185.1234, then by default it is wrapped to +174.8766.

VIEWER=val	If in spectre compatibility mode and this option is set and equals "smartview", then generation of PSF data will be suppressed and regular binary raw file generation will be enforced.
WIDTH (CO)=val	Resets width (in columns) of the output from the .PRINT and .PLOT statements. The default width is 80, and the minimum width is 40.
WILDCARDSTAT	Prints wildcard statistic information.

Example

```
VIN   1 0   DC 0   SIN(0 0.1 5MEG)   AC 1
VCC   8 0   DC 10
VEE   9 0   DC -12
RS1   1 2   1K
RS2   5 0   1K
RC1   3 8   10K
RC2   4 8   10K
RBIAS   7 8   20K
CLOAD   3 4   5PF
Q1   3 2 6   QNL
Q2   4 5 6   QNL
Q3   6 7 9   QNL
Q4   7 7 9   QNL
```

```
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF CJC=2PF
VA=50)
.TRAN 5NS 500NS
.PRINT V(*)
.PROBE I(Q?) I(R*)
.OPTIONS NOMOD NODECK WILDCARDSTAT
.END
```

This example will produce next statistic information:

```
Wildcard statistics:

[.print]
  .print v(*)
    wildcard '*' matches ... 10 times

[.probe]
  .probe i(q?) i(r*)
    wildcard 'q?' matches ... 4 times
    wildcard 'r*' matches ... 5 times
```

Third Party Interface (Batch Mode Operation)

CDS (SDA)	This flag is used to support the Cadence Design Framework (WSF ASCII format). Default is 2.
CSDF	Generates a graphic file in ViewLogic format (CSDF format). Default is OFF.

The header line length is limited to 72 characters. The number of digits saved for output variable values into rawfile is 16.

Example

```
'i(1:mm1i1)' 'i(2:mm1i1)' 'i(3:mm1i1)'
'i(x__sub1.x__sub2.x__sub3.x__sub4.x__sub5.x__sub6.x__sub7.xsub1.m
m1i1)'
```

Long vectors names are truncated to be less than 72 characters. SmartSpice tries to express vector names such in the form as (index:vectorname). In the previous example, the indices are 0, 1, 2, 3, and the vector name is mm1i1. The indices are consistent with .pa file, like in -hspice mode:

```
1
x__sub1.x__sub2.x__sub3.x__sub4.x__sub5.x__sub6.x__sub7.x__sub8.xs
ub1
2
x__sub1.x__sub2.x__sub3.x__sub4.x__sub5.x__sub6.x__sub7.x__sub8.xs
ub2
3
x__sub1.x__sub2.x__sub3.x__sub4.x__sub5.x__sub6.x__sub7.x__sub8.xs
ub3
```

The problem has been fixed when the option CSDF is defined:

- SmartSpice generates multiple result files instead of a single file in case a deck has an analysis statement (DC/ TRAN/ AC) with second sweep specification.

- In case a deck has DC analysis statement(s), and the `start` value is greater than the `stop` value, such as `.DC vin 10 0 -1`. SmartSpice writes every analysis step into an individual result file.

EPIC	Prints `.op`/`.print` information in `.epc` file. The operating point information can be saved in HSPICE (`epic|epic=1`) or Berkeley (`epic=2`) format. Default is 0 - (no information and no file saved).

Syntax

```
.OPTION EPIC < = 1 | 2 >
```

Examples

```
.OPTION EPIC
.OPTION EPIC=1
.OPTION EPIC=2
```

In the first two examples SmartSpice prints an operating point information in HSpice format.

```
******  operating point information         tnom= 27 temp= 27
******
***** operating point status is all       simulation time is     0.
     node =voltage         node =voltage          node =voltage

+ 0:0        =   0.0000e+00 0:net1     = -9.2263e+00 0:net2     = -
7.6293e-01
+ 0:net3     =   0.0000e+00 0:net4     = -6.2070e-03 0:net5     =
4.5790e+00
+ 0:net6     =   4.5790e+00 0:net7     = -6.2070e-03 0:vcc      =
1.0000e+01
+ 0:vss     = -1.0000e+01
```

In the third example SmartSpice prints an operating point information in Berkeley format.

```
                          * D:\ex1.in
                    DC Operating Point Analysis, 27 deg C
                          Wed Sep 12 11:50:55 2007
---------------------------------------------------------------
                           i(c1)    =  0.0000e+00
                           i(r1)    =  5.4210e-04
                           i(r2)    =  5.4210e-04
                           i(r3)    = -6.2070e-06
                           i(r4)    =  6.2070e-06
                           i(rbias) = -9.6131e-04
                           v(0)     =  0.0000e+00
                           v(net1)  = -9.2263e+00
                           v(net2)  = -7.6293e-01
                           v(net3)  =  0.0000e+00
                           v(net4)  = -6.2070e-03
                           v(net5)  =  4.5790e+00
```

PSF	This flag is used to support the Cadence Design Framework (`PSF=1` for ASCII format and `PSF=2` for binary). Default is `1(ASCII)`.

WSF	Generates a graphic file in ViewLogic format (WSF format). Default is OFF.

Specific Cases

FLATTENED_DCMONTE	If set, SmartSpice will generate a flattened plot for DC Monte Carlo analysis. Also, the monte_carlo parameter vector will be added into the plot. The FLATTENED_DCMONTE mode can be set by either the .OPTIONS statement in the input deck, or by the variable in the control block. If set in both places, .OPTIONS will override the variable setting.
GELEMENTLAPLACEARG=1|0	Controls the algorithm modification to calculate the convolution in G-element when transfer function is defined by Laplace transform. G-element algorithms: • enhanced convolution algorithm with corrected DC component • convolution algorithm with IFFT Default is 0 for transient analysis.
IGNOREMCSTEPERROR	If set, then errors during Monte Carlo step simulation are ignored and Monte Carlo continues to the next step. Previously, SmartSpice aborted simulations in such cases. By default, this functionality is OFF, and may be turned ON with either an option or the variable with the same name.
LIS_NEW	In batch mode -b and -sb when .OPTION LIS_NEW=1 is specified, SmartSpice moves .PRINT output data to separate files with the extension .PRINT <Analysis Type> <number starting from zero>.
MODMONTE	Used to distinguish between global and local model variations. When .OPTION MODMONTE is set to 0 all devices of a specific model have the same model parameter values. When the .OPTION MODMONTE is set to 1 each device has unique model parameter values.
MONTECON=1|0	Forces Monte Carlo analysis to continue in case of non-convergence, device and temperature setup errors, and parametric reload issues. Default is 1.

Example

```
xmdut0   d0   g   s   b   nch_mac   w=01.40u   l=01.000u   globalflag=1
mismatchflag=1
xmdut1   d1   g   s   b   nch_mac   w=01.40u   l=01.000u   globalflag=1
mismatchflag=1
xmdut2   d2   g   s   b   nch_mac   w=01.40u   l=01.000u   globalflag=1
mismatchflag=1
.dc vgs 0 1.100000 1.100000 sweep monte=5 prmc
```

```
.OPTION MODMONTE=1
```

Output

```
MONTE CARLO STEP #1
      MODEL PARAMETER MONTE CARLO DEFINITIONS
Instance Name: xmdut1
@nch_mac.1[plo_tox] = 1.07944
@nch_mac.1[plo_dxl] = 0.169053
......
Instance Name: xmdut0
@nch_mac.1[plo_tox] = -0.219049
@nch_mac.1[plo_dxl] = 0.155796
......
Instance Name: xmdut2
@nch_mac.2[plo_tox] = -0.092009
@nch_mac.2[plo_dxl] = -1.13146
Parameter modification:
plo_dxw = -0.457528 [agauss(0, 1, 1, 1)]
plo_dxl = -1.13146 [agauss(0, 1, 1, 1)]
```

In this example all instances of the model nch_mac have unique local model parameters (plo_tox, plo_dxl).

SWEEPMONTE	allows you to run Monte Carlo statistical analysis as SWEEP (nested parametric analysis) for AC, DC, and TRAN analyses. By default this option is OFF.

Syntax

```
.option sweepmonte
```

Example

```
.option post
.save all
.TRAN 5NS 500NS
.DC VIN -0.25 0.25 0.025 sweep monte=5
.AC DEC 10 10KHZ 10GHZ
```

In this example SmartSpice performs Monte Carlo analysis for all analyses specified in netlist. Transient, DC, and AC analyses will be performed five times each. In -hspice mode SmartSpice generates fifteen raw files.

Example

```
.option post sweepmonte
.save all
.TRAN 5NS 500NS
.DC VIN -0.25 0.25 0.025 sweep monte=5
.AC DEC 10 10KHZ 10GHZ
```

In above example SmartSpice performs Monte Carlo analysis for .DC statement only. SmartSpice saves Monte Carlo iterations as correspondent sweep sections. A swept vector monte_carlo will be added into DC raw file. In -hspice mode SmartSpice generates three raw files.

RMNODELESS2	If the option RMNODELESS2 is set SmartSpice removes diodes, capacitors, resistors and inductors which have one connection.

Example

```
.OPTION RMNODELESS2
```

NPORTALG	S-element algorithms: • 0 - inverse FFT algorithm uses Y parameters for impulse functions generation • 1 - Foster network equivalent scheme generation algorithm • 2 - inverse FFT algorithm uses S parameters for impulse functions generation Default is 1 for transient analysis. For AC and not-transient analysis default is 0. Under not-transient analysis a Foster equivalent network is not applied.
SR_ABSTOL, SR_RELTOL, SR_VNTOL, SR_LTERATIO, SR_LVLTIM	Overrides corresponding option values for ABSTOL, RELTOL, VNTOL, LTERATIO or LVLTIM initially set after RESTORE when the SAVE/RESTORE feature is enabled.

Syntax

```
.option optionname = val
```

where:

```
optionname = { SR_ABSTOL | SR_RELTOL | SR_VNTOL | SR_LTERATIO |
SR_LVLTIM }
```

If SR_RELTOL=1e-4 is specified in the netlist and the feature SAVE/RESTORE is used, SmartSpice will override current RELTOL value with the value specified in SR_RELTOL (1e-4) initially set after RESTORE and will continue simulation with this new tolerance.

Example

```
.OPTION SR_RELTOL = 1e-4
```

SWEEPALG	Controls matrix processing during parametric analysis: • 0 - default behavior • 1 - model setup is called on every sweep step • 2 - block matrix reorder for sweep steps, which are without model setup. Default is 0.

SOA - Safe Operating Area

These options allow SmartSpice to issue warnings when the terminal voltages of a device exceed the SOA (Safe Operating Areas).

MAXWARNS	Controls the maximum number of messages to be printed. Default is 5.
WARN	Turns on (WARN=1) and off (WARN=0) the SOA check. Default is 1.

Currently supported devices are: MOSFET (BSIM3v3, BSIM4, PSP, HiSIMHV), Bipolar (SGP), Diode, Capacitor and Resistor.

Model parameters have been introduced that define SOA check maximums. If the absolute value of the voltage difference between device terminals exceeds the specified maximum, a warning is issued. Default value of these parameters is 0, which stands for no maximum and disables the check.

MOSFET

Model parameters: Vgs_max, Vgs_max, Vgd_max, Vds_max, Vbd_max, Vbd_max

Terminal voltages checked: Vgs, Vgb, Vgd, Vds, Vbd, Vbs

Checked conditions:

```
|Vgs| > Vgs_max
|Vgb| > Vgs_max
|Vgd| > Vgd_max
|Vds| > Vds_max
|Vbd| > Vbd_max
|Vbs| > Vbd_max
```

Bipolar

Model parameters: Vbe_max, Vbc_max, Vce_max

Terminal voltages checked: Vbe, Vbc, Vce

Checked conditions:

```
|Vbe| > Vbe_max
|Vbc| > Vbc_max
|Vce| > Vce_max
```

Diode

Model parameter: Bv_max

Terminal voltage checked: Vj (from P to N)

Checked condition:

```
|Vj| > Bv_max
```

Capacitor

Model and instance parameter: Bv_max (if both are given, instance parameter overrides model parameter)

Terminal voltage checked: Vc

Checked condition:

```
|Vc| > Bv_max
```

Resistor

Model and instance parameter: `Bv_max` (if both are given, instance parameter overrides model parameter)

Terminal voltage checked: `Vr`

Checked condition:

```
|Vr| > Bv_max
```

Table 3-7 Model and Pole/Zero Analysis Options

Model and Pole/Zero Analysis Options			
MOSFET	**Pole/Zero Analysis**	**General**	**Warnings**
ACM	ITLPZ	ASPEC (IPLDEL)	EXPERT
DEFAD	PZABSTOL (PZABS)	GEOSHRINK	NODIODEWARN
DEFAS	PZABS	MACMOD	NOINTERNALWARN
DEFL	PZACCEL	SCALE	NONANWARN
DEFNRD	PZITLIM (ITLPZ)	SCALM	NONEGWARN
DEFNRS	PZRATTOL (RITOL)	SHMOD (SELFT)	NOPARAMCHKWARN
DEFPD	PZRELTOL (PZTOL)	THMOD	NOSOAWARN
DEFPS	PZTOL		NOWARN
DEFW	RITOL		NOWARN_LESSTHAN TWOCONNECTIONS
HDIF	X0IM, X1IM, X2IM		NOWARN_LWRANGE
LD	X0I, X1I, X2I		NOWARN_SHORTDEV
MODELFAST	X0RE, X1RE, X2RE		SAVEMODELSLOG (SAVELODELLOG)
MSEARCH	X0R, X1R, X2R		WARNLIMIT
MSEARCHAUTO			
NOADJUST			
WL			
WNFLAG			
Bipolar	**Inductor**	**Transmission Lines**	**Topology Checks**
DCAP (JCAP)	ADDK (GENK)	RISETIME	CHECK_TERMINAL_ CONNECT
	KMIN (KLIM)	RLGCNOCHECK	
		TRYTOCOMPACT	

Model and Pole/Zero Analysis Options

MOSFET

ACM=val	Default value of parameter ACM for MOSFET models. Default is 0.
DEFAD=val	Resets the value for the default MOS drain diffusion area. Default is 0.0.
DEFAS=val	Resets the value for the default MOS source diffusion area. Default is 0.0.
DEFL=val	Resets the value for the default MOS channel length. Default is *1.0E-4*.
DEFNRD=val	Default number of drain squares for MOSFET model. Default is 0.
DEFNRS=val	Default number of source squares for MOSFET model. Default is 0.
DEFPD=val	Default drain periphery for MOSFET model. Default is 0.
DEFPS=val	Default source periphery for MOSFET model. Default is 0.
DEFW=val	Resets the value for the default MOS channel width. Default is 1.0E-4.
HDIF=val	Default value of the parameter HDIF for MOSFET models. Default is 0.
LD=val	Default value of the parameter LD for MOSFET models. Default is 0.
MODELFAST=1	Eliminates the calculations of the source/drain diode currents (BSIM3v3 and BSIM4 models) and the gate tunneling current (BSIM4 model only). Default is 0.
MSEARCH	0 – use “less and equal” criteria for the upper boundary for the geometry binning. For example: LMIN <= L <= LMAX. 1 (default) – use “less” criteria for the upper boundary for the geometry binning. For example: LMIN <= L < LMAX.
MSEARCHAUTO	1 - automatically choose the criteria for the upper boundary for the geometry binning. When the option MSEARCH=1 and no model has been found for the specific instance, SmartSpice will try to use “less and equal” criteria (option MSEARCH=0) for the upper boundary for geometry binning.

NOADJUST	Flag which turns off the automatic unit adjustment algorithm for oxide thickness. Default is OFF. The default units for oxide thickness (TOX) in all models are meters. In most cases, an automatic unit adjustment algorithm will recognize units properly, and it is not necessary to edit existing input decks. If the value of TOX is greater than 1.0, SmartSpice assumes that units are in Angstroms; if TOX is greater than 0.001 and less than 1.0, SmartSpice assumes that units are in microns, otherwise meters. However, if .OPTIONS NOADJUST is specified, SmartSpice assumes that unit for TOX is in meters.
WL	Reverses the order that the width and length of the MOSFET device is specified. If set to 1, width is the first parameter. Default is OFF.
WNFLAG	Controls MOSFET models binning equation selection. If the parameter WNFLAG is not specified in instance statement, then SmartSpice uses the .OPTIONS WNFLAG. Default is 1.

Pole/Zero Analysis

PZABSTOL (PZABS)=val	Absolute tolerances for poles and zeros. The .option PZABS default value is 1e-02.
PZACCEL	Flag that turns on Aitken's acceleration. Default is OFF.
PZITLIM (ITLPZ)=val	Pole/zero analysis iteration limit. Default is 100.
PZRATTOL (RITOL)=val	Minimum ratio value for (real/imaginary) or (imaginary/real) parts of the poles or zeros. Default is *1.e-8*.
PZRELTOL (PZTOL)=val	Relative tolerances for poles and zeros. Default is *1.0E-9*.
X0IM (X0I)=val	Imaginary part of the first starting point in the Muller pole/zero analysis algorithm. Default is 0.0.
X0RE (X0R)=val	Real part of the first starting point in the Muller pole/zero analysis algorithm. Default is 1.0.
X1IM (X1I)=val	Imaginary part of the second starting point in the Muller pole/zero analysis algorithm. Default is 0.0.
X1RE (X1R)=val	Real part of the second starting point in the Muller pole/zero analysis algorithm. Default is 0.0.
X2IM (X2I)=val	Imaginary part of the third starting point in the Muller pole/zero analysis algorithm. Default is 0.0.
X2RE (X2R)=val	Real part of the third starting point in the Muller pole/zero analysis algorithm. Default is -1.5.

General

ASPEC (IPLDEL)	If specified, default value of SCALE and SCALM is 1.e-6. If SCALE or SCALM is specified, then the specified value will be used, and this option will have no effect on that value. Default is OFF.
GEOSHRINK	This option describes device scaling factor. Default is 1.

Syntax

```
.OPTION GEOSHRINK=val
```

Besides the option SCALE, GEOSHRINK is used for scaling geometric parameters of instance. GEOSHRINK factor is applied to all elements in the netlist.

The values of geometric parameters are calculated as listed:

```
ELEMENT_DIMENSION = (ORIGINAL_DIMENSION · SCALE) · GEOSHRINK
```

Example

```
.OPTION GEOSHRINK = 0.95
.OPTION GEOSHRINK = 0.85
.OPTION SCALE = 1E-5
.OPTION SCALE = 1.5E-5
```

If a netlist contains more than a single scale/geoshrink definition, the last scale/geoshrink value is used.

The following values are applied to elements:

```
ELEMENT_DIMENSION = (ORIGINAL_DIMENSION · 1.5e-5) · 0.85
```

MACMOD	The option ‘macmod’ allows SmartSpice to have access to the subcircuit definition if there is no matching model for M-instance (MOSFET) or enable SmartSpice to have access to the MOSFET model definition if there is no matching subcircuit definition or Verilog-A module for X-instance.

Syntax

```
.OPTION MACMOD= [1|2|3|0]
```

If MACMOD=1, SmartSpice seeks a subcircuit definition for the M-instance if there is no matching model exists. The desired subcircuit name must match (case insensitive) the mname field in the M-instance statement.

If MACMOD=2, SmartSpice seeks a MOSFET model definition for the X-instance if there is no matching subcircuit or Verilog-A module definition. SmartSpice will output an error if the found model card that matched X-instance name is not a type of MOSFET models.

If MACMOD=3, SmartSpice enables both features: it seeks a subcircuit definition for the M-instance if there is no matching model exists and it seeks a MOSFET model definition for the X-instance if there is no matching subcircuit or Verilog-A module definition

If MACMOD=0 or there is no .option MACMOD specified in the netlist then both modes are disabled.

If there are multiple MACMOD options specified in the netlist, SmartSpice will use the last one.

Note: If option is specified but the value is not set, SmartSpice assumes that MACMOD=1.

SCALE=val	Scale factor for device parameters. Default is 1.
SCALM=val	Scales factors for model parameters. Default is 1.
SHMOD (SELFT)=val	Self-heating global selector used in UCSD-HBT/VBIC bipolar models and in BSIM3SOI MOS models. Default is 0.
THMOD=val	This option is used by the bipolar models that include self-heating capability: Mextram, HBT, HICUM and VBIC. An optional built-in thermal circuit can be created to perform temperature calculations, or to create external thermal networks. Different models are available and can be selected using the .OPTIONS THMOD option. Default is 0. In SmartSpice, two different models are available to build the internal electrothermal circuit, as shown in Figure 3-15:

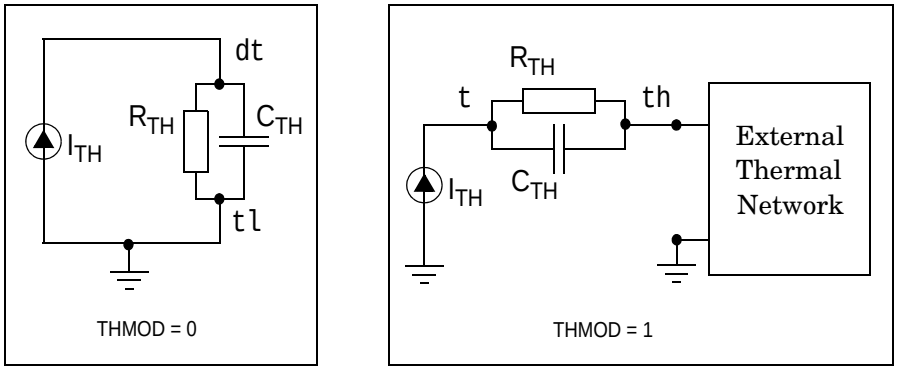


Figure 3-15 Thermal Subcircuit Topologies

Due to the incompatibility of these topologies, it is not possible to use them simultaneously in devices connected to external thermal networks. To address this, the option THMOD can be used to select a model for the overall circuit. The model used for a given device depends on the value of this option parameter, and whether that thermal nodes are set as terminals or not. If no thermal terminals are specified, the equivalent circuit corresponding to THMOD=0 is automatically selected, even if THMOD is set to a different value.

If THMOD=0 (default), the current I_{th} flows from the ground to a node ‘dt’ which is created internally if not set as a terminal. R_{th} and C_{th} are connected in parallel between ‘dt’ and the ground. In this case, the overall chip is assumed to be the heatsink, and the voltage of the electrical ground in SmartSpice (0V) corresponds to the operating temperature of the circuit (set by a .TEMP statement for example). So the voltage of the node ‘dt’ is the temperature rise of the instance above the circuit temperature. The transistor internal temperature is equal to *V(dt)+TEMP(circuit)* and is available as device output variable to be printed and plotted using the syntax @name[t].

If THMOD=1, the current I_{th} flows from the ground to the node ‘t’, which is created internally if not set as a terminal. R_{th} and C_{th} are connected in parallel between nodes ‘t’ and ‘th’. It is important to notice that at least the ‘th’ node must be set as terminal. Otherwise, the model corresponding to THMOD=0 is used for this device. Moreover, the terminal ‘th’ has to be connected to an external network. If not, non-convergence always occurs. The external thermal network can be a single voltage source, corresponding physically to a heatsink. For this case, the voltage of the electrical ground in SmartSpice (0V) corresponds to zero temperature in degrees Celsius. So the voltages of ‘t’ and ‘th’ nodes (and nodes of the thermal network) are temperature values expressed in degrees Celsius. The transistor internal temperature is equal to V(t) and is available as device output variable to be printed and plotted.

For both cases, thermal terminals can be used to construct thermal circuits where heating of one transistor by another takes place, or to construct elaborate self-heating models (where, for example, a better approximation than a single pole can be used to describe the time dependence of device temperature).

Warnings

Warnings can be enabled or disabled using the following options:

EXPERT	If non-zero, enables **Not a Number**, **Negative conductances or capacitances**, **Forward biased diodes** and **Internal model** warnings. Default is 0.
NODIODEWARN	Disables **Forward biased diode** warnings.
NOINTERNALWARN	Disables **Model internal** warnings.
NONANWARN	Disables **Not a Number** warnings.
NONEGWARN	Disables **Negative conductances or capacitances** warnings.
NOPARAMCHKWARN	Disables **Parameter Check** messages.
NOSOAWARN	Disables **Safe Operating Area** warnings.
NOWARN	Disables all warnings but fatal ones.
NOWARN_LESSTHANTWOCONNECTIONS	Suppresses output warning message when a node has less than 2 connections.
NOWARN_LWRANGE	Suppresses output warning message when an instance’s length or width are out of model's range.
NOWARN_SHORTDEV=val	Control warnings issued for devices with terminals connected together. If this option is set and equals 1, warnings on such devices will be suppressed. If set to 0, warnings will be printed. This option will override ‘suppressshortdevicewarnings’ if set. Default is unset.

SAVEMODELSLOG (SAVEMODELLOG)	Saves warnings to a logfile. Named after the netlist's filename. For example, running `MOS9test.in` would create a logfile named `MOS9test.log`. Fatal errors are always sent to screen; if this option is set they will also be saved to the logfile.
WARNLIMIT=val	Limits the number of warnings displayed for the following categories: **Negative conductances or capacitances**, **Forward biased diodes** and **Internal model warnings**. The value `0` is interpreted as 'no limiting'. See below for a list of warnings categories. Default is 5.

Table 3-8 Summary of Warnings Categories

Category	Default	Enabled by	Disabled by	Limited by
Parameter Check	Enabled		NOPARAMCHKWARN	
Not a Number	Disabled	EXPERT	NONANWARN	
Safe Operating Area	Enabled		NOSOAWARN	
Negative cond. or cap.	Disabled	EXPERT	NONEGWARN	WARNLIMIT
Forward biased diode	Disabled	EXPERT	NODIODEWARN	WARNLIMIT
Internal model warning	Disabled	EXPERT	NOINTERNALWARN	WARNLIMIT

Bipolar

DCAP (JCAP)=val	Capacitance model selector. Default is `0`.

Inductor

ADDK (GENK)=val	If the input deck contains inductance packages with coupling elements K, then SmartSpice generates secondary (hidden) coupling elements, which improve the stability of the system of equations describing the circuit behavior. Default is `1` (`ON`).
KMIN (KLIM)=val	Magnitude of hidden secondary coupling elements. Default is `0.01`.

Transmission Lines

RISETIME=val	Minimum source risetime in the circuit. This setting enables an estimation of the maximum frequency of the circuit which will be used as a default value. In some cases this value is used to calculate with time frequency constants needed for inverse FFt and convolution. If value is not given, default value is calculated automaticaly based on the following algorithm: • If PWL or PULSE sources are present in the circuit, risetime is equal to the smallest TR/TF or increment in time. • If there are no PWL, PULSE sources in the circuit, risetime is equal to '0.25* TSTEP' specified in the .TRAN statement. If this algorithm cannot find the default value, then risetime=1e-9.
RLGCNOCHECK	Flag to turn off all checking of RLGC matrices for non-physical negative values, and diagonal domination in all W element transmission line devices which are contained in the circuit. Default is OFF (checking is performed). If .OPTIONS RLGCNOCHECK is specified, the specification of the W element device parameter RLGCCHECK=1 allows the turning on of the RLGC matrix, checking for individual W element devices.
TRYTOCOMPACT	Tries to condense LTRA transmission lines past history of input voltages and currents. Default is OFF.

Topology Checks

CHECK_TERMINAL_CONNECT	Forces SmartSpice to check the topology of the subcircuits and determine subcircuit terminals with no connection to upper levels of subcircuit hierarchy. If such terminals are determined, an error message and table with these terminals appear. This is a fatal error so a simulation will not be performed.

Table 3-9 DC and Operating Point Analysis Options

DC and Operating Point Analysis Options		
Matrix/Solver	**Accuracy**	**Convergency**
ITERTOL	ABSH	ACCEPT
PIVALG	ABSI	AUTOTRANOP
PIVOT	ABSMOS	CONV
PIVREL	ABSTOL	DCGMIN (GMINDC)
PIVTOL	ABSVDC (DCVNTOL)	DCGMCHK
SOLVER	RELH	DCGMSTEPS
(see Section 21.1 Matrix/ Solver Options for full descriptions of these options)	RELI	DCGNODE (DCGSHUNT)
	RELMOS	DCPATH
	RELTOL (RELV)	DCSTEP
	RELVDC	EXPERT
	VNTOL (ABSV)	EXPLI
		GMIN
		GMINSTEPS
		ITL1
		ITL2
		NEWTOL
		SRCSTEPS (ITL6)
Output	**Initial Conditions**	**General**
OPSAVEALL	AUTO_CALLVSAVEV	ITL5
DC Analysis	GMAX	
CAPDC (DCCAP)	ICG	
DCPATHNODE		
Transient Analysis		
BIASCHKMODE		
BIASFILE		
BIASWARN		
BIASSTOP		

DC and Operating Point Analysis Options		
BIASSTOPTIME		
CAPTAB (CAPDC)		
CAPTABTIME		
CAPTABTIMEFROM		
CAPTABTIMETo		
CUTOFFTAB		
EXBYPASS		
KEEPOPINFO		
VFLOOR		

DC and Operating Point Analysis Options

Matrix/Solver (see Chapter 21 Using Solvers in SmartSpice)

Accuracy

ABSH=val	Sets the absolute current change through voltage-defined branches (voltage sources and inductors). As a default, SmartSpice uses ABSTOL value (ABSTOL default is *1.e-9*). To suppress checking for voltage-defined branch current convergency, specify ABSH=0.
ABSI=val	Sets the absolute current error tolerance used for Bipolars and JFETs. SmartSpice uses the ABSI value to determine if the internal currents of the device reached convergence on Newton iterations. Default is *1.0e-9* in -hspice mode otherwise is ignored if the value is not set.
ABSMOS=val	Resets the absolute current error tolerance used for MOSFET's. SmartSpice uses the ABSMOS setting to determine if the **drain-to-source current** solution has convergence on Newton iterations. Default is *1.0E-6*.
ABSTOL=val	Resets the absolute current error tolerance of the simulation. SmartSpice uses the ABSTOL setting for DC analysis and DC operating point calculation to determine if the current solution has convergence on Newton iterations. Default is *1.0E-9*.
ABSVDC (DCVNTOL)=val	Resets the absolute voltage error tolerance of the simulation. SmartSpice uses the ABSVDC setting for DC analysis and DC operating point calculation to determine if the voltage solution has convergence on Newton iterations. Default is VNTOL.
RELH=val	Sets the relative current tolerance through voltage-defined branches (voltage sources and inductors). As a default, SmartSpice uses the RELTOL value. RELH is effective to check convergency, but only if the value of the ABSH control option is greater than zero.
RELI=val	Sets the relative current error tolerance used for Bipolars and JFETs. SmartSpice uses the RELI value to determine if the internal currents of the device reached convergence on Newton iterations. Default is 0.01 (1%) in -hspice mode otherwise is ignored if the value is not set.
RELMOS=val	Resets the relative current error tolerance used for MOSFET's. SmartSpice uses the RELMOS setting to determine if the drain-to-source current solution has convergence on Newton iterations. Default is 0.05 (5%).

RELTOL (RELV)=val	Resets the relative error tolerance of the simulation (for voltages and currents). SmartSpice uses the RELTOL setting for DC analysis and DC operating point calculation to determine if the solution has convergence on Newton iterations. Default is *1.0E-3*.
RELVDC=val	Sets the relative node voltage error tolerance for DC analysis and DC operating point calculation. As a default, SmartSpice uses RELTOL value (RELTOL default is 0.001 for DC/OP analysis).
VNTOL (ABSV)=val	Resets the absolute voltage error tolerance of the simulation. Default is *5.0E-5*.

Table 3-10 Relative and Absolute Accuracy Tolerances

Type	Option (Aliases)	Default
Nodal Voltage Tolerances	RELTOL (RELV)	0.001
	VNTOL (ABSV)	50.e-6
Nodal Voltage Tolerances for DC/OP	RELVDC	RELTOL
	ABSVDC (DCVNTOL)	VNTOL
Brance Current Tolerances (voltage sources and inductors branch currents)	RELH	RELTOL
	ABSH	ABSTOL
	ABSTOL	1.e-9
Active device Current Tolerances		
Bipolar and JFETs	RELI	Default is 0.01 (1%) in -hspice mode otherwise is ignored if the value is not set.
	ABSI	Default is 1.0e-9 in -hspice mode otherwise is ignored if the value is not set.
MOSFETs	RELMOS	0.05
	ABSMOS	1.0e-6

The default convergence algorithm checks node voltages and branch currents only. Setting RELMOS, ABSMOS, RELI or ABSI options allows you to include the intrinsic currents of the active devices into the convergence criterion.

Hierarchical Options

A support of hierarchical options has been implemented. It allows a netlist to use different option settings on different circuit hierarchy levels. The following options can be specified as hierarchical:

- abstol
- reltol
- vntol
- bypass
- bytol
- hsimspeed

Syntax

```
.subckt <...> options: hsimspeed=0|3|5
Xcall <....> options: hsimspeed=0|3|5
```

Note: Hierarchical option scoping rule is local with the exception of the option hsimspeed. The .option hsimspeed and .param hsimspeed have a priority under corresponding hierarchical option.

Convergency

SmartSpice calculates the DC operating point if the input deck contains the OP statement, and if the UIC option is not specified prior to the Transient and small-signal analysis. The system of DC equations used corresponds to the circuit with the capacitors opened, and the inductors shorted. SmartSpice solves the system using the Newton-Raphson method. To improve Newton-Raphson method convergency, the DCGMIN conductance (default is *1.e-12*) is placed in parallel with all pn junctions of all active devices and all drain-to-source nodes of the MOSFET/SOI devices, as shown in Table 3-11.

If the standard Newton-Raphson method fails to converge, or if convergence requires a large number iterations, the additional advanced algorithms are used. Figure 3-16 illustrates the flow of auto convergence algorithm.

SmartSpice performs the auto convergence algorithm in four steps. Steps 2, 3 and 4 are advanced stepping algorithms, based on the continuation method, when the solution of each step provides a good initial guess for the next. Step 3 performs stepping algorithm, changing the diagonal elements of the Jacobi Matrix by the value of the option GMIN. The limit of iterations at each step of advanced stepping algorithms is specified by using ITL2 (50, by default) but SmartSpice will decide how many iterations are needed using ITL2 as a starting point.

CONV=5 **Standard Newton-Raphson iterations:** SmartSpice simulates circuit “as is” without any changes in the convergency control parameters. The limit of iterations is equal to ITL1 (100, by default). If SmartSpice does not converge, it goes to step 2.

CONV=1 **DCGMIN Stepping:** SmartSpice multiplies the DCGMIN (*1.0e-12*, by default) value by $10^{DCGMSTEPS}$, where DCGMSTEPS is the number of continuation method steps (10, by default), and simulates the circuit. If it converges, then DCGMIN is divided by 10 and the circuit is simulated again. This iteration process is continued until DCGMIN reaches the original DCGMIN value. If convergence is not achieved, SmartSpice goes to the step 3.

CONV=2 **diagGMIN Stepping:** SmartSpice multiplies the GMIN (*1.0e-12*, by default) value by $10^{GMINSTEPS}$, where GMINSTEPS is the number of continuation method steps (10, by default), changes the diagonal elements of the Jacobi Matrix by the value of parameter GMIN, and simulates the circuit. If it converges, then GMIN is divided by 10 and the circuit is simulated again. This iteration process is continued until GMIN reaches the original GMIN value. If convergence is not achieved, SmartSpice goes to the step 4.

CONV=4 **Source Stepping:** SmartSpice sets all voltage/current sources to 0 and simulates the circuit. If it converges, then the value of each voltage/current source is increased by an *SRCvalue/SRCSTEPS* increment, where *SRCvalue* is the original voltage/current source value, and *SRCSTEPS* is the number of continuation method steps (10, by default), and the circuit is simulated again. This iterative process is continued until the voltage/current source values reach the original level.

Pseudo-Transient Analysis (PTA)

Pseudo-Transient Analysis is invoked by the advanced stepping OP algorithm and cascaded with algorithms CONV=5 (default), CONV=1, 2 ,4. PTA is designed for OP calculation. The PTA functionality can be called directly, bypassing the stepping algorithms by specifying .option CONV=-1. The PTA function employs a modified regular transient analysis and uses controlling options described below. The OP calculated by PTA is a valid operating point which is used as a starting point for a regular transient analysis. PTA is recommended for circuits with floating nodes, varactors and slow convergence (when conv=1, 2 are too slow). The PTA function automatically generates a .tran statement based on the user specified .tran statement in the input deck. If there is no user defined .tran statement the PTA function will use internal defaults for it's own .tran statement. The PTA defaults can be changed using the following options.

AUTOTRANOP	This option controls the cascaded advanced convergence stepping algorithms. PTA is not invoked if set to 0. PTA is invoked if set to 1. If set to 2, PTA will be used in cascade to find the initial guess for the advanced convergence stepping algorithm. Default is 2.

DCTRANOPARG	Approximate timepoint at which solution is accepted and is the duration of PTA. Default is 'TSTOP* dctranopdampfactor' for operating point calculation under transient analysis Where TSTOP is the simulation end time parameter taken from .tran statement. If only .OP is specified the default for dctranoparg is 1e-9. Under .DC analysis, dctranoparg default value is 1e-6 for each dc sweep point. If .tran is combined with another analysis dctranoparg default value is 'TSTOP* dctranopdampfactor'. If non-transient and non-dc analysis are not combined with .tran the default for dctranoparg is 1e-9. PTA analysis may alter duration based on convergence/error control and several timepoints beyond of dctranoparg value may be calculated. PTA also may stop at several timepoints earlier based on the same error/convergence control. The last calculated timepoint is accepted as the solution for steady state. The .option ptasavewaveform preserves calculated vectors in the standard plot structure and can be used for tuning PTA duration.
DCTRANOPSTEP	Timestep for PTA. Default is equal to dctranoparg.
DCTRANOPDAMPFACTOR	Multiplier for dctranoparg. Default is 1.
PTACNODE	Connects a grounded capacitor to each circuit node node during PTA stage only. Shunt is applied by default. Default value is 1e-18. .option cnode value overrides .option ptacnode. Recommended range (1e-12;1e-18).
PTAGNODE	Connects a grounded conductance to each circuit node during PTA stage only. Shunt is applied by default. Default value is 1e-18. .option gnode value overrides .option ptagnode. Recommended range is (1e-12; 1e-18).
PTASAVEWAVEFORM	Saves waveforms calculated during PTA. Useful for circuit steady state detection. Default is 0 (do not save).

PTA is very similar to the regular transient analysis and uses timestep control algorithm. You should select reasonable duration of PTA. If the default value of dctranoparg is too large use the following algorithm

- Set .option dctranopdampfactor=1
- Set .option dctranoparg="value", where "value" is the duration of PTA. Using .option ptasavewaveform=1 you can estimate the "value" of dctranoparg. Usually a waveform has a ramped shape which become flat at a certain timepoint or will sligthly oscillate. Step away 20 -30 timepoints to the flat side and use a selected timepoint as a value for dctranoparg.
- Slow convergence might be fixed by adjusting .option ptacnode.
- Singularity during PTA might be fixed by .option ptagnode.

If the option dctranoparg is set to value greater than 8.64e4, it will be reset to TSTOP * dctranopdampfactor and the following warning will be printed out:

```
Warning: Option DCTRANOPARG = XXX is too large, new value = YYY is
being used.
```

If the option dctranopstep is set to value greater than 8.64e3, it will be reset to dctranoparg/10 and the following warning will be printed out:

```
Warning: Option DCTRANOPSTEP = WWW is too large, new value = ZZZ is
being used.
```

Example

```
.option conv=-1 dctranopdampfactor=1 dctranoparg=5e-07 ptacnode=1e-
12
```

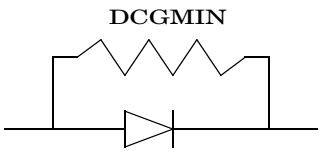
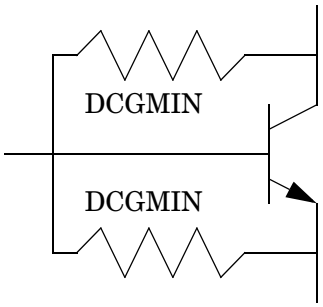
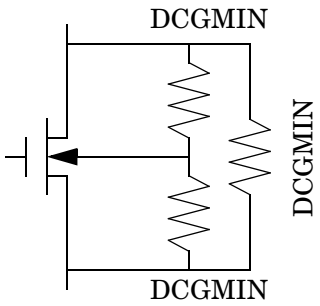
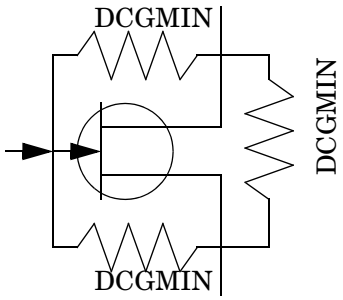
PTA is invoked directly bypassing cascade of CONV stepping algorithms

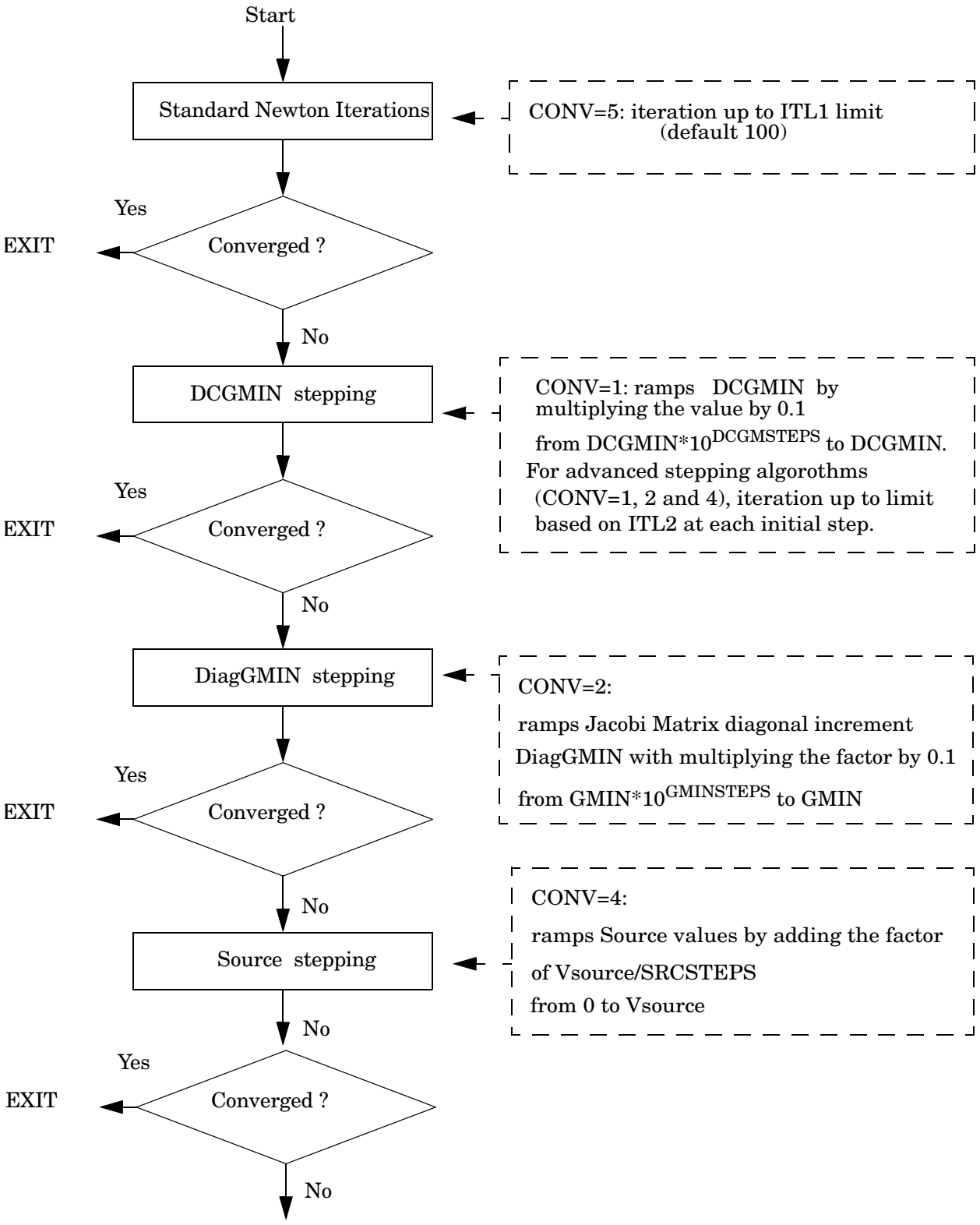
Duration of transient is dctranoparg=5e-07, during PTA only grounded capacitor 1p is connected and then disconnected for a regular transient.

The operating point calculated by PTA might be different to what the real OP analysis finds. The following recommendations can be applied for OP refinement, and can be applied if the simulator went through CONV=5,1,2,4.

- Use .savebias time.bias tran time=4e-09 all. 4e-09 is just an example; any time poin can be selectedt.
- Run PTA with options such as .option conv=-1 dctranopdampfactor=1 dctranoparg=5e-07 ptacnode=1e-12.
- When file "time.bias" is created stop regular transient.
- Remove all PTA settings - ".option conv=-1 dctranopdampfactor=1 dctranoparg=5e-07 ptacnode=1e-12 ".
- Remove .savebias card.
- Open time.bias file and replace .NODESET by .IC.
- Place .include time.bias in your netlist.
- Run simulation. See if the simulator finds OP without stepping CONV=1,2,4.

Table 3-11 DCGMIN implementation

Device	DC/OP Analysis
	DCGMIN=1.e-12 (default)
Diode	
BJT	
MOSFET/SOI	
MESFET/JFET/TFT	


Start
Standard Newton Iterations
CONV=5: iteration up to ITL1 limit (default 100)
Yes
EXIT
Converged ?
No
DCGMIN stepping
CONV=1: ramps DCGMIN by multiplying the value by 0.1
from DCGMIN*10^DCGMSTEPS to DCGMIN.
For advanced stepping algorothms (CONV=1, 2 and 4), iteration up to limit based on ITL2 at each initial step.
Yes
EXIT
Converged ?
No
DiagGMIN stepping
CONV=2:
ramps Jacobi Matrix diagonal increment DiagGMIN with multiplying the factor by 0.1
from GMIN*10^GMINSTEPS to GMIN
Yes
EXIT
Converged ?
No
Source stepping
CONV=4:
ramps Source values by adding the factor of Vsource/SRCSTEPS
from 0 to Vsource
No
Yes
EXIT
Converged ?
No

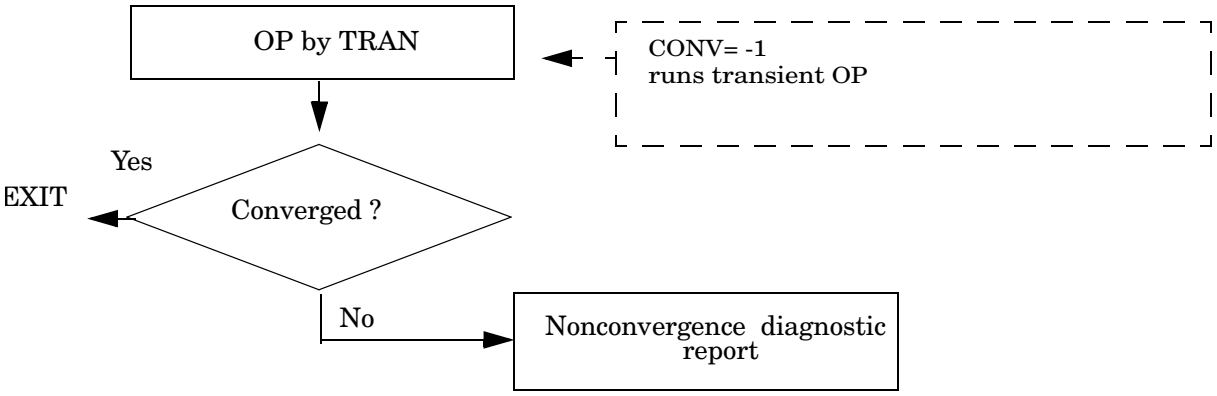


Figure 3-16 Auto Convergence Algorithm Flow Diagram

AutoConvergence Algorithm

When CONV=1,2,4 are failed SmartSpice automatically runs OP by transient(CONV=-1). The options dctranopstoptime, dctranopstep and dctranoparg control settings of ramping transient analysis.

The default settings are:

```
dctranopstoptime = 5e-09;
dctranopstep = 1e-09;
dctranoparg = 1e-09;
```

Transient analysis specification is not needed directly in the netlist.

SmartSpice calls transient analysis using dctranopstoptime, dctranopstep and dctranoparg values. The value of dctranoparg corresponds to the value of the TRANOP parameter in the .tran statement.

If convergence is not achieved, SmartSpice prints a non-convergence summary containing information regarding nonconvergent nodes and devices, as well as further information describing the algorithm used. Recommendations on DCGMIN and GMIN values, which could produced convergence, are also included.

Example

SmartSpice Computational Error Diagnostics:

```
Error: DCOP: no convergence for CONV=1

Stepping Algorithm CONV=1 DIAGNOSTICS for .TRAN at  0.000000e+00 :
DC_Gmin stepping SUCCEEDED on step 4 with DCGMIN=  2.867676e-06
DC_Gmin stepping failed on step 5
SUGGESTION : Convergence for CONV=1,3 may be achieved
             by increasing iteration limit ITL2 > 50
          or by increasing parameter DCGMIN >  2.867676e-06
             by statement .OPTION DCGMIN = 3.00e-6 DCGMSTEPS = 4
          or by .OPTION CONV=1 ACCEPT
REASON     :     Nonconvergence     of     node     voltage     V(
xda.xctl.x1.x41_banksel_c )
Final value :          6.818156e-01
Previous value :       6.865769e-01
Difference :          -4.761369e-03  Diff/Val :  6.835381e+01
User-defined VNTOL  :  1.000000e-06  RELTOL   :  1.000000e-04
```

```
Value prior to previous value :  6.860849e-01
Previous difference :  4.920447e-04
The number of iterations : 50

Nonconvergent Nodes
Node                          Voltage      Error        Tolerance
xda.xctl.x1.x41.ddh_bs_l    4.2241e-01   9.5259e-04   4.3241e-05
xda.xctl.x1.x41.m2_d        2.6384e-01   1.9000e-04   2.7384e-05
xda.xctl.x1.x41_banksel_c   6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x9.m12_d        4.8998e-04   3.6558e-06   1.0490e-06
xda.xctl.x1.x9.m14_s        5.0102e-01   9.7311e-04   5.1199e-05
xda.xctl.x1.x9.ddh_ld0bs_   2.3082e-03   2.3034e-05   1.2308e-06
xda.xctl.x1.x9.m17_d        1.5926e-03   2.3034e-05   1.1593e-06
xda.xctl.x1.x9.m15_s        8.3898e-01   1.2291e-04   8.4898e-05
xda.xctl.x1.su_bs0_c_me1b   6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc1         6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc2         6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc3         6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc4         6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc5         6.8182e-01   4.7614e-03   6.9658e-05
The greatest node voltage : V( vddh_lib ) =  1.300000e+00
The greatest current      : i(vdd) = -1.710159e+00
```

In this example SmartSpice performed DCGMIN stepping. By default, *DCGMIN=1.e-12*. For this reason, an initial value *DCGMIN=DCGMIN * 1.0e10=1.0e-2* was used. SmartSpice converged for *DCGMIN=1.0e-2, 1.0e-3, 1.0e-4, 1.0e-5*, and failed for *DCGMIN=1.0e-6*. After that, SmartSpice used a special procedure with automatic variation DCGMIN step size, reached the value *DCGMIN=2.867676e-06*, failed again, and generated these diagnostic messages:

```
DC_Gmin stepping SUCCEEDED on step 4 with DCGMIN=  2.867676e-06
DC_Gmin stepping failed on step 5
```

From this message, we know that SmartSpice will converge for *DCGMIN value 1.e-2, ..., 1.e-5, 2.867676e-06*, and will not for *1.e-6*. SmartSpice therefore generated the messages:

```
SUGGESTION : Convergence for CONV=1,3 may be achieved
               by increasing iteration limit ITL2 > 50
          or by increasing parameter DCGMIN >  2.867676e-06
               by statement .OPTION DCGMIN = 3.00e-6 DCGMSTEPS = 4
          or by .OPTION CONV=1 ACCEPT
```

SmartSpice checks to confirm that convergence criteria is satisfied for node voltages and devices separately on each Newton-Raphson iteration. When checking node voltages on the iteration '*m*', it does not accept convergence if, for at least one node voltage, the following condition is not satisfied:

$$\left|V^m - V^{m-1}\right| \le max(\left|V^m\right|, \left|V^{m-1}\right|) \times RELTOL + VNTOL$$

```
REASON     :     Nonconvergence     of     node     voltage     V(
xda.xctl.x1.x41_banksel_c )
Final value :          6.818156e-01
Previous value :       6.865769e-01
Difference :          -4.761369e-03  Diff/Val :  6.835381e+01
User-defined VNTOL  :  1.000000e-06  RELTOL   :  1.000000e-04
Value prior to previous value :  6.860849e-01
Previous difference :  4.920447e-04
The number of iterations : 50
```

The previous voltage criterion is checked only if the device current criterion that can be written in the form is satisfied:

$$\left|I(V^m) - I^{m-1}\right| \le max(\left|I(V^m)\right|, \left|I^m\right|) \times RELTOL + ABSTOL$$

where $I(V^m)$ and I^m are the actual and linearized branch currents respectively on the iteration '*m*', and *ABSTOL* is the absolute error tolerance for currents.

While printing the brief nonconvergency summary (when the `.OPTIONS EXPERT` option is not specified), SmartSpice outputs either the number of nonconvergent devices, or the first nonconvergent node voltage. The keyword `EXPERT` in the `.OPTIONS` statement causes more detailed nonconvergence diagnostics to be printed. SmartSpice prints the list of all nonconvergent nodes or devices:

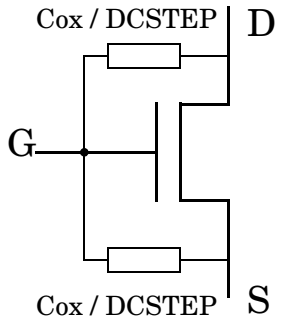
```
Nonconvergent Nodes
Node                          Voltage      Error        Tolerance
xda.xctl.x1.x41.ddh_bs_l    4.2241e-01   9.5259e-04   4.3241e-05
xda.xctl.x1.x41.m2_d        2.6384e-01   1.9000e-04   2.7384e-05
xda.xctl.x1.x41_banksel_c   6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x9.m12_d        4.8998e-04   3.6558e-06   1.0490e-06
xda.xctl.x1.x9.m14_s        5.0102e-01   9.7311e-04   5.1199e-05
xda.xctl.x1.x9.ddh_ld0bs_   2.3082e-03   2.3034e-05   1.2308e-06
xda.xctl.x1.x9.m17_d        1.5926e-03   2.3034e-05   1.1593e-06
xda.xctl.x1.x9.m15_s        8.3898e-01   1.2291e-04   8.4898e-05
xda.xctl.x1.su_bs0_c_me1b   6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc1         6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc2         6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc3         6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc4         6.8182e-01   4.7614e-03   6.9658e-05
xda.xctl.x1.x56.nc5         6.8182e-01   4.7614e-03   6.9658e-05
```

ACCEPT	Causes SmartSpice, when DC/OP nonconvergency takes place, to accept DC/OP a solution with the least (and < `1.e-4`) `DCGMIN` stepping algorithm (`CONV=1`) factor value, at which convergency has been achieved. Default is `OFF`.
AUTOTRANOP=val	This option controls the cascaded advanced convergence stepping algorithms. PTA is not invoked if set to 0. PTA is invoked if set to 1. If set to 2, PTA will be used in cascade to find the initial guess for the advanced convergence stepping algorithm. Default is 2.

CONV=val	Resets the algorithm used for the DC operating point calculation in all analysis. • CONV=1 performs advanced DCGMIN stepping; • CONV=2 performs advanced DiagGMIN stepping; • CONV=4 performs advanced source stepping; • CONV=-1 • CONV=5 automatically cycles through the above algorithms (CONV=1,2,4,-1). If the first convergence method fails, it will proceed to next. Default is 5.
DCGMIN (GMINDC)=val	Resets a conductance placed in parallel with all pn junctions of all models, and all drain-to-source nodes of the MOSFET/SOI models during the DC/OP analyses only. It improves convergence properties. If the DCGMIN option is equal to zero, the parallel conductances are not connected during the DC/OP calculations, except for advanced DCGMIN (CONV=1,3) stepping algorithm. Default is *1.0E-12*.
DCGMCHK	Checks the influence of DCGMIN on the accuracy of the output variables, if *DCGMIN > 1.e-15*. SmartSpice analyzes the influence of DCGMIN on the accuracy of node voltages and branch currents *calculated at the first DC operating point*. It checks whether the DCGMIN value is compatible with accuracy parameter RELTOL, ABSTOL and VNTOL values. SmartSpice may decrease the DCGMIN value in order to satisfy accuracy tolerances. Default is OFF.
DCGMSTEPS=val	Sets the number of steps for advanced DCGMIN stepping algorithm, in order to improve DC convergence properties. In the case convergence problems, SmartSpice sequentially multiplies DCGMIN by the coefficients $10^{DCGMSTEPS+1}$, $10^{DCGMSTEPS}$, ..., 1 and recalculates the DC operating point. Default is 10.
DCGNODE (DCGSHUNT)=val	Connects a grounded conductance of specified value to every circuit node to improve DC convergence properties, and help to solve the “Matrix is singular” problems caused by a specific circuit topology. The conductances are connected during the DC operating point computation phase only. Default is 0 (even if GNODE used).

DCPATH=val	Sets the conductance value that SmartSpice will use to connect “floating” nodes to ground. A floating node is a node from which there is no “DC path” to ground regardless of a specified DCSTEP option. To suppress DCPATH, specify DCPATH=0.0. Default is *1.0E-12*. If .ic condition is specified for floating node, the .ic functionality (shunting) will override dcpath/dcpathnode shunting and dcpath/dcpathnode shunting will not be applied.
DCSTEP=val	If the specified value *dcstepval* is greater than zero, then SmartSpice replaces DC model and element capacitors with conductances of magnitude *condval*, as follows: $condval = \frac{capval}{dcstepval}$ where *capval* is the actual device capacitance value.

Concerning model capacitors in MOSFETs, TFT and SOIs, the conductance *condval* is placed as follows:



The capacitances are replaced with conductances **during the DC computation phase only**. They are then released when the circuit is simulated in the time domain. It helps to solve “Matrix is singular” problems caused by capacitors opening during the DC (OP) analysis. Default is 0.0.

EXPERT<=val>	Provides information about nonconvergent nodes and devices; activates warning messages about NaN, and nonphysical negative values in active device models; and creates convergency plots. EXPERT=777 makes this information more detailed. EXPERT=779 activates warning messages about BSIM3v3 and BSIM3H model parameters, which are out of range. Default is OFF.

Note: Use the option EXPERT to give more details when debugging a problem circuit.

EXPLI=val	Sets the explosion current value. When a diode current exceeds EXPLI, current is computed using linear extrapolation.When *EXPLI > 0*, junctions characteristics above EXPLI are linear, using the slope at the explosion point. This helps to obtain convergence when currents reach unphysical values. Explosion current limitation applies to all models that include diodes (BJTs, Diodes level 1 and 500, and MOSFET with extrinsic diodes enabled with DIOLEV=1, 2, 3, 4 or 5). Default is *exp(45)=3.5E19 A* for BJT (levels 1 and 2) and Diodes (levels 1 and 500) to assure backward compatibility. For other models, default is 0 (limiting is disabled). EXPLI is also a model parameter. If specified within a model card, it overrides the global value set by the .OPTIONS statement.
GMIN=val	Resets the minimum conductance allowed by the program. Resets Jacobi Matrix diagonal increment for DC/OP convergence advanced DiagGMIN (CONV=2) stepping algorithm. Default is *1.0E-12*.
GMINSTEPS=val	Sets the number of steps for advanced DiagGMIN stepping algorithms, in order to improve DC convergence properties. In the case convergence problems, SmartSpice sequentially multiplies GMIN by the coefficients $10^{GMINSTEPD+1}$, $10^{GMINSTEPS}$, ..., 1 and recalculates the DC operating point. Default is 10.
ITL1=val	Resets the DC operating point iteration limit when SmartSpice performs standard Newton iterations (CONV=0). Default is 100.
ITL2=val	Resets the iteration limit at any point of the DC transfer curve, and the iteration limit which is used for the advanced stepping algorithms. Default is 50.
NEWTOL	Calculates one more convergent iteration past convergence for every DC/OP solution calculated. When NEWTOL is not set, once convergence is determined, the iteration process is terminated. Default is OFF.
SRCSTEPS (ITL6)=val	Sets the number of steps for advanced source stepping algorithm, in order to improve DC convergence properties. Default is 10.

Output

OPSAVEALL	OPSAVEALL = 1: SmartSpice saves all .OP analysis vectors. OPSAVEALL = 0: SmartSpice saves only vectors which are specified in the output dot statements. Default is ON.

Example

```
.OPTION OPSAVEALL=0
.OP
.PROBE OP V(1) v(2)
```

In this example SmartSpice saves only V(1) and V(2) vectors.

DC Analysis

CAPDC (DCCAP)=val	If CAPDC=1, then in the DC analysis SmartSpice calculates the capacitance values of a circuit (both model and element). Default is 0. This option will not automatically print the capacitance values.
DCPATHNODE	Defines the node that “floating” nodes will be connected to using the DCPATH conductance value.

Syntax

```
.option DCPATHNODE=”name”
```

Example

```
.option DCPATH = 1e-12
.option DCPATHNODE = “n27”
```

In this example SmartSpice uses conductance value 1e-12 to connect “floating” nodes to the node “n27”.

Transient Analysis

BIASCHKMODE	This option can be set to 0 (default), 1 or 2 to invoke different algorithms for voltage monitoring. Please refer to the description of the .BIASCHK statement for further details.
BIASFILE	Applying this option causes the output of all .BIASCHK statements to be sent to the specified file.
BIASWARN	If this option is set to 1, a warning message is sent out instantly if any local bias voltage exceeds the limit during Transient analysis. The results summary is sent out after Transient analysis completion. If this option is set to 0 (the default value), no messages are sent out during Transient analysis. The summary is sent out after Transient analysis completion.
BIASSTOP	The controlling option BIASSTOP affects the simulation flow in transient analysis if the internal condition for .BIASCHK is satisfied. Simulation flow will be stopped, resulting in the message “Error: .biaschk: biaschk aborted because .option biasstop is specified”. The resolution of stopping the simulation is the nearest time point. That means when all circuit instances (devices or circuit nodes) were checked for the current time point, the analysis will be stopped.

Example

```
.OPTION BIASSTOP
```

BIASSTOPTIME	The controlling option BIASSTOPTIME affects the simulation flow in transient analysis if the internal condition for .biaschk is satisfied. Simulation flow will be stopped indicating a message: `Error: .biaschk: biaschk aborted because .option biasstoptime is specified.` Also, final transient analysis time and the value of BIASSTOPTIME will be printed. The granularity of stopping is a time point. That means when all circuit instances (devices or circuit nodes) were checked for current time point analysis will be stopped. Analysis will be stopped if BIASSTOPTIME is reached and a breakdown condition is detected before BIASSTOPTIME. If BIASSTOPTIME is reached, and no breakdown condition has occurred, then transient analysis will overcome the BIASSTOPTIME point and will stop after first breakdown condition is detected.

Example

```
.OPTION BIASSTOPTIME=60ns
```

CAPTAB (or **CAPDC**)	Calculates the contributions on nodal capacitances of MOSFET transistors and passive capacitors. The table of capacitances associated with circuit nodes is printed as a result of the operating point calculation at *t=0* for Transient analysis. Default is OFF. CAPTAB=2 prints a nodal capacitance table in a single column format:
	`node = capacitance`
	`<Node1> = <Value1>`
	`...`
	`<NodeN> = <ValueN>`
	Nodal capacitance table (NCT) is printed for every operating point analysis under .OPTION CAPDC and .OPTION CAPTAB. In DC sweep analysis, NCT is printed under .OPTION CAPDC=2 or .OPTION CAPTAB=2. NCT is HSPICE compatible. Printing under DCsweep is an enhancement over HSPICE.

CAPTABTIME	Permits printing of the nodal capacitance table in a specified timepoint for transient analysis.

Example

```
.OPTION CAPTABTIME=1n
```

CAPTABTIMEFROM	Specifies the left boundary of the time window beginning from which the nodal capacitance table will be printed for every time-point.

Example

```
.OPTION CAPTABTIMEFROM=1e-06
```

CAPTABTIMETO	Specifies the right boundary of the time window beginning from which the printing of the nodal capacitance table will be stopped.

Example

```
.OPTION CAPTABTIMETO=2e-06
```

CUTOFFTAB	Performs a search for the cut-off nodes at the end of the operating point calculation. A node is considered cut-off if it is shared (via source or drain connections) between two cut-off MOSFET devices. Found nodes are reported in table. Default is OFF.
EXPBYPASS	The following algorithm applies to expressions in RES, IND elements if they don't contain integral or derivative: 1. Expression evaluation is skipped if expression arguments remain the same. Expression and derivative values are taken from prior computation. Otherwise: 2. Derivative calculation is skipped if old and new expression values are within specified tolerance. If specific condition is met, then derivative values corresponding to the old expression value are reused. Tolerance is controlled by EXPBYPASS option. Increasing this value will cause more derivative calculations to be skipped but may also cause nonconvergence or timestep reduction. 0 value turns off derivative calculation bypass. Default is 0 (off).
KEEPOPINFO	Causes SmartSpice to save the operating point information for each small-signal (.AC, .DISTO, .PZ) analysis into a rawfile and appear as a plot in the **Vectors** window. Default is OFF.
VFLOOR	Lower voltages that are printed in the OP output listing. All voltages lower than VFLOOR are printed as 0. Default is 0.

Note: Expression and derivative calculation bypass is not effective for expressions containing integral, partial integral, derivative, partial derivative.

Initial Conditions

AUTO_CALLVSAVEV	Equivalent to the parameters SAVEV and CALLV in TRAN, AC, DC analysis. This option may take two values: 0 and 1. Default is 0 (turned off).

Example

```
.OPTION AUTO_CALLVSAVEV (means 1)
.OPTION AUTO_CALLVSAVEV=1
```

ICG (GMAX)=val	This option is used to calculate the value of the parallel conductance *GIC*, which defines the strength of the current source attached to each node specified in .IC and .NODESET statements, by the formula: $GIC = max(ICG, 10000 \times GCIR)$ *GCIR* is the equivalent conductance of the circuit connected to the initialization node. The combination of independent current source and parallel conductance is active only during the DC operating point calculation, and only at first iteration for the .NODESET statement case. Default is ICG=1.
NODESET=1|0	Triggers a savebias/loadbias mechanism for parametric transient analysis. On the first parametric step, bias will be saved and then reused as a starting point for the next step (e.g. .DATA). Default is 0 (off).

General

ITL5=val	Resets the DC transfer curve total iteration limit. The option ITL5=0 allows an infinite number of iterations. Default is 0.

Table 3-12 Transient Analysis Options

Transient Analysis Options		
Speed	**Accuracy**	**Convergency**
AUTOSTOP	ABSH	CNODE (CSHUNT)
BYPASS	ABSI	DCPATH
BYTOL	ABSMOS	EQNTHRESHOLD
CAP_MNA_FORMULA	ABSTOL	EXPLI
FAST	ACCURATE	GMIN
HSIMSPEED	CHGTOL	GNODE (GSHUNT)
MBYPASS	FFT_ACCURATE	NEWTOL
VZERO	RELH	
	RELI	
	RELMOS	
	RELTOL (RELV)	
	RUNLVL	
	TRTOL	
	VNTOL (ABSV)	
TimeStep	**Algorithm**	**General**
COEF1 (FT)	COEF1 (FT)	BIASCHKMODE
HMIN	HMIN	BIASFILE
HMINREJ	HMINREJ	BIASWARN
IMAX	INTEGR	BREAKSLOPE (SLOPETOL)
ITL4	ITL4 (IMAX)	KEEPFLAT HIERSUBCKT
ITL7	ITL41	ITL5
ITL41	MAXORD	LIMPTS
LTERATIO	METHOD	LOGIC
LVLTIM	RMAX (RFLAG)	MINBREAK
RMAX (RFLAG)	RMIN	TTICK
RELREF	TMAX (DELMAX)	REBINNING_VERBOSE
RELVAR	VSTA	SEED
RMIN		TTICK

Transient Analysis Options		
TMAX (DELMAX)		NOSINSOURCEBREAK
VSTA		
VSTALIM		

Transient Analysis Options

Speed

AUTOSTOP<=1,2, 'expression'>	If AUTOSTOP=1 the Transient analysis will be stopped when all the independent measurements have been completed. If AUTOSTOP=2 the Transient analysis will be stoped when at least one of the independent measurements has been completed. If AUTOSTOP='expression' the Transient analysis will be stoped when the 'expression' takes the value of logic '1'. The 'expression' can involve results of measurements , variables and logical operators with them. Default is 0 (OFF). This option will not be effective for DERIV or WAVE measurements.

Example

```
.meas tran m1 trig v(1) val=2.5 rise=1 targ v(3) val=3.5 rise=1
.meas tran m2 find v(3) when v(2)=4 fall=1
.meas tran m3 WHEN V(3)=3.0 RISE=ALLEVENT

.OPTION AUTOSTOP='m1&&(m2||m3)||v(3)>4'
```

The Transient analysis stops when m1 measure is obtained and m2 or m3 measure is obtained or just value of variable v(3) will be more than 4.

BYPASS<=1, 2>	SmartSpice bypasses the calculation of unchanging elements. Default is 0 (OFF). BYPASS=2 is supported in the following models: DIO, BJT, BSIM3v3, BSIM4, RPI a-Si TFT (RPIaTFT) and RPI p-Si TFT (RPIpTFT).
BYTOL=val	Specifies the tolerance for voltage at which a device is considered latent. If the bias voltages of a given device change less than the value of BYTOL between two consecutive iterations, then the device is considered latent. Default is *MBYPASS x VNTOL*.
CAP_MNA_FORMULA	Changes MNA (Modified Nodal Analysis) formulation of the nonlinear capacitor. If specified, the value of the nonlinear capacitor is considered constant on the two continuous timepoints. This option makes the capacitor model compatible with Eldo, Spectre and HSPICE, and significantly increases the performance (capacitor derivative calculations are not necessary). Under DC analysis, the option is reset in 0 (disabled) under any mode. If PTA is working under DC analysis option is set to the value specified by the user or to the default value.
FAST	Speeds up simulation by bypassing the calculation of unchanging elements (sets BYPASS=1) and sets default value for $TMAX = (tstop - tstart)/100$. Default is OFF.
HSIMSPEED=3	For vzero=0 only. Supported in the BSIM3v3, BSIM4, RPI a- and poly-Si TFT, and diode models.

MBYPASS=val	Multiplication factor used to compute the default value of BYTOL. Default is 1.0.
VZERO=val	This option changes the MNA (Modified Nodal Analysis) formulation. Default is 0. When VZERO=0, then the MNA formulation used is identical to the approach used by UC Berkeley SPICE3, and generates the set of Newton-Raphson method equations for nodal voltages and branch currents. • When VZERO=1: MNA formulation generates the set of equations with respect to updating the nodal voltages and branch currents. • When VZERO=2: Incorporates the functionality of vzero=1 (solve equations with respect to update (delta)) and adds processing of voltage sources so that branches are analysed first and together with VSRC nodes removed from the equation set. Update (delta) for VSRC nodes is always zero; VSRC currents are calculated by the postprocessor and not via the standard simulation evaluation (unlike VZERO=0 and 1). You may see (in GUI print nodeconn command) equations and their order, all VSRC nodes and branches have negative equation numbers and do not participate in the simulation analysis.

Example

vdd, vdd1, vdd2, 11,12,13 are VSRC nodes equations with vzero=2.

Location of VSRC branches and nodes; they have a negative equation number.

node name	equation number	connection number	node type
i(vdd)	eq.num=-12	N=0	type=NODE_CURRENT
i(vdd1)	eq.num=-11	N=0	type=NODE_CURRENT
i(vdd2)	eq.num=-10	N=0	type=NODE_CURRENT
i(v1)	eq.num=-9	N=0	type=NODE_CURRENT
i(v2)	eq.num=-8	N=0	type=NODE_CURRENT
i(v3)	eq.num=-7	N=0	type=NODE_CURRENT
13	eq.num=-6	N=7	type=NODE_VOLTAGE
12	eq.num=-5	N=57	type=NODE_VOLTAGE
11	eq.num=-4	N=3	type=NODE_VOLTAGE
vdd2	eq.num=-3	N=9	type=NODE_VOLTAGE
vdd1	eq.num=-2	N=28	type=NODE_VOLTAGE
vdd	eq.num=-1	N=89	type=NODE_VOLTAGE
0	eq.num=0	N=117	type=NODE_VOLTAGE

Accuracy

ABSH=val	Sets the absolute current change through voltage-defined branches (voltage sources and inductors). As a default, SmartSpice uses the ABSTOL value (ABSTOL default is 1.0E-9). To suppress checking for voltage-defined branch current convergency, specify ABSH=0.
ABSMOS=val	Resets the absolute current error tolerance used for MOSFETs. SmartSpice uses the ABMOS setting to determine if the drain-to-source current solution has converged on Newton iterations. Default is 1.0E-6.
ABSTOL=val	Resets the absolute current error tolerance of the simulation. Default is 1.0E-9.
ACCURATE=val	This option is used during the DC operating point calculations and Transient analysis. Default is 1. It sets the following accuracy control options:

For ACCURATE=1:

```
RELTOL=1.0E-3
VNTOL=1.0E-6
ABSTOL=1.0E-12
TRTOL=3.5
METHOD=GEAR
VSTALIM=0.1
FFT_ACCURATE
```

For ACCURATE=2:

```
RELTOL=1.0E-3
VNTOL=1.0E-6
ABSTOL=1.0E-12
TRTOL=3.5
ITL41=ITL4 =10 (default)
METHOD=GEAR
FFT_ACCURATE
```

If ACCURATE is used in conjunction with any of the following accuracy control options:

- RELTOL
- VNTOL
- ABSTOL
- TRTOL
- METHOD
- VSTALIM
- ITL41
- ITL4

SmartSpice will use the value from the netlist for these options.

CHGTOL=val	Resets the absolute charge tolerance of the simulation. Default is 1.0E-14.
FFT_ACCURATE	Causes SmartSpice to dynamically adjust the time step so that each FFT point will be a real simulation point. This eliminates the interpolation error, and provides the highest FFT accuracy with minimal overhead in simulation time. Default is OFF. .OPTIONS ACCURATE internally turns on the FFT_ACCURATE option.
RELH=val	Sets the relative current tolerance, through voltage-defined branches (voltage sources and inductors). As a default, SmartSpice uses RELTOL value. RELH is effective to check convergency, but only if the value of the ABSH control option is greater than zero.
RELTOL (RELV)=val	Resets the relative (for voltages, currents, charges and etc.) error tolerance of the simulation. Default is 1.0E-2.
RUNLVL=1	Effective under default timestep algorithm lvltim=4. Considered as an extension of runlvl=2. Uses relaxation algorithm for waveform slope control. Recommended for any type of circuit. Accuracy is in the range 5%-15%. Might significantly (2-4x) reduce number of timepoints.
RUNLVL=2	Effective under default timestep algorithm lvltim=4. Improves control of abrupt changes in the waveform. Recommended for analog and digital circuits. Provides more efficient control for circuits with PULSE, PWL voltage sources. Accuracy is not affected; purpose is to decrease the number of timepoints
RUNLVL=3	Effective under default timestep algorithm lvltim=4. Improves relaxation of timestep control for digital type circuits. Keeps accuracy at 1%-2%.
RUNLVL=5	Effective under default timestep algorithm lvltim=4. Allows time steps to be bigger then maximum time step estimated by transient analysis. Keeps accuracy at 1%-2%
TRTOL=val	Resets the internal timestep multiplier, used by local truncation error timestep algorithm. Default is 7.0.
VNTOL (ABSV)=val	Resets the absolute voltage error tolerance of the simulation. Default is 5.0E-5.

Table 3-13 Relative and Absolute Accuracy Tolerances

Type	Option (Aliases)	Default
Nodal Voltage Tolerances	RELTOL (RELV)	0.01
	VNTOL (ABSV)	50.e-6
Branch Current Tolerances (voltage sources and inductors branch currents)	RELH	RELTOL
	ABSH	ABSTOL
	ABSTOL	1.e-9
Active device Current Tolerances		
Diodes, bipolar and JFETs	RELI	Default is 0.01 (1%) in -hspice mode otherwise is ignored if the value is not set.
	ABSI	Default is 1.0e-9 in -hspice mode otherwise is ignored if the value is not set.
MOSFETs	RELMOS	ineffective
	ABSMOS	ineffective

Convergency

CNODE (CSHUNT)=val	Connects a grounded capacitance of specified value to every circuit node. It also helps to solve some “Timestep too small” problems caused by high-frequency oscillations or numerical noise. Default is 0.
DCPATH=val	Sets the conductance value that SmartSpice will use to connect “floating” nodes to ground. A floating node is a node from which there is no “DC path” to ground. To suppress DCPATH, specify DCPATH=0.0. Default is 1.0E-11.
EQNTHRESHOLD	Supports the solver switch from default to SPEEDS. SmartSpice will use solver SPEEDS if the option solver is not given, the option eqnthreshold is specified and circuit equation size is greater than eqnthresholdvalue. Default eqnthreshold value is -1. The option eqnthreshold can be set using the variable eqnthreshold in the INI-file.

EXPLI=val	Sets the explosion current value. When a diode current exceeds EXPLI, current is computed using linear extrapolation. When *EXPLI > 0*, junction characteristics above EXPLI are linear, using the slope at the explosion point. This helps to obtain convergence when currents reach unphysical values. Explosion current limitation applies to all models that include diodes (BJTs, Diodes level 1 and 500, and MOSFET with extrinsic diodes enabled with DIOLEV=1, 2, 3, 4 or 5). Default is *exp(45)=3.5E19 A* for BJT (levels 1 and 2) and Diodes (levels 1 and 500), to assure backward compatibility. For other models, default is 0 (limiting is disabled). EXPLI is also a model parameter. If specified within a model card, it overrides the global value set by the .OPTIONS statement.
GMIN=val	Resets the minimum conductance allowed by the program. Resets the conductance which is placed in parallel with all pn junctions of all models and all drain-to-source nodes of the MOSFET/SOI models to improve Newton iterations convergence properties at each time point during Transient analysis. If the option GMIN is set to zero, the parallel conductances are not connected. Default is 1.0E-12.
GNODE (GSHUNT)=val	Connects a grounded conductance of specified value to every circuit node to improve Newton iterations convergence properties. It also helps to solve some “Timestep too small” problems caused by high-frequency oscillations or numerical noise, and solve the “Matrix is singular” problems caused by specific circuit topology. Default is 0.
NEWTOL	Calculates one more convergent iteration past convergence for every timepoint circuit solution calculated. When NEWTOL is not set, once convergence is determined, the iteration process is terminated. Default is OFF. When NEWTOL is set, the value of the parameter ITL41 is increased by 2.

TimeStep

LTERATIO=val	Scaling factor for relative tolerance in LTE algorithm for LVLTIM=2, and 4. The local truncation error tolerance is calculated by SmartSpice internally InternalToleranceLTE = RELTOL * LTERATIO (RELTOL - .OPTION RELTOL, LTERATIO - .OPTION LTERATIO) and is used only for truncation error estimation. The default value depends on the user input. If LTERATIO is directly specified in the netlist, then InternalToleranceLTE=RELTOL*LTERATIO (from the user .OPTION). If LTERATIO is not specified in the netlist, then it is set to 1 if RELTOL < 0.0015, otherwise LTERATIO=1./SQRT(RELTOL*1000). A smaller value of LTERATIO will produce more accurate result. IF LTErej in the statistics table (.OPTION ACCT) has many LTE device rejections, then LTERATIO must be decreased. Another regulation factor for removing LTE device rejections – decrease .OPTION RELTOL(InternalToleranceLTE = RELTOL * LTERATIO).
LVLTIM=val	Selects the timestep algorithm used for Transient analysis: • LVLTIM=0 selects the iteration count algorithm (ITC). • LVLTIM=1 selects the voltage slope changes algorithm (without timestep reversal), along with the iteration count algorithm. • LVLTIM=2 selects the local truncation error (LTE) algorithm, along with the iteration count algorithm. • LVLTIM=3 selects the voltage slope changes algorithm (with timestep reversal), along with the iteration count algorithm. • LVLTIM=4 selects generic SmartSpice advanced algorithm along with ITC, LTE, and voltage slope changes (with timestep reversal). • LVLTIM=5 selects modified generic algorithm. Default is 4.
RELREF=LOCAL|GLOBAL	Determines which values are used to compute relative tolerance in voltage slope changes algorithm. You can set RELREF to the following options: • RELREF=LOCAL compares the changes in quantities at each node relative to the current value of that node. • RELREF=GLOBAL compares the changes in quantities at each node relative to the maximum of all voltages in the circuit. Default is LOCAL for LVLTIM=4, and GLOBAL for LVLTIM=1, 3 and 5.
RELVAR=val	Sets the relative voltage changes for LVLTIM=1, and 3 if the .OPTIONS parameter VSTA is not specified. Default is 0.3.

Time-integration Algorithm

INTEGR	Automatically selects a proper integration method (either a GEAR or TRAPEZOIDAL). Default is OFF.

MAXORD=val	Maximum order of integration. MAXORD can be either 1 or 2. If MAXORD=1, both TRAPEZOIDAL and GEAR methods are the backward Euler method. Default is 2.
METHOD=name	Resets the integration method for Transient analysis (either TRAPEZOIDAL or GEAR). The TRAPEZOIDAL integration method generally results in reduce simulation time, with more accurate results. The GEAR method smooths out the oscillations found in the TRAPEZOIDAL method. Default is TRAP.

Timestep Algorithm

COEF1 (FT)=val	Resets a decreasing *timestep* coefficient for the iteration set that does not converge. New *timestep* is calculated as the non-convergency *timestep* is multiplied by COEF1. Default is 0.25.
ITL4 (IMAX)=val	Resets the Newton method iteration limit on any integration step in the Transient analysis. If convergence criteria is not satisfied by integration *step*, and the number of iterations is greater than ITL4, the *timestep* is multiplied by the coefficient COEF1. Default is 10.
ITL7	Specifies the number of TSTS recovery tries per timepoint in transient and PTA analysis. The .option expert might be used to track the recovery algorithm. Default is 3.
ITL41=val	If the number of iterations on an integration *step* is greater than ITL41, the time step is not increased. Default is 5.
HMIN=val	Set minimum value for integration *timestep*, when the solution is accepted, regardless of iteration convergency. If HMIN is specified, and the convergence criteria is not satisfied on integration step equals to HMIN, the *timestep* is not decreased and the calculated timepoint is accepted. Default is 0 (OFF).
HMINREJ=val	Option (flag) to count rejections based on the HMIN value. If set in the netlist, SmartSpice will create the vector v(HMINrej) and print the value of HMIN and the rejection count HMINrej at the end of the transient simulation.
RMAX (RFLAG)=val	Sets multiplier for the default maximum internal timestep when neither the TMAX option nor the *tmax* parameter in the .TRAN statement is used. In this case only, the maximum internal timestep is the smaller of $RMAX \times tstep$, or $(tstop - tstart)/50$. Default is 10.

RMIN=val	Sets the multiplier, when the low limit for integration timestep *delmin* is calculated, as: $$delmin = rmin \times tstep,$$ where *tstep* is the first parameter from the .TRAN statement *tstep tstop tstart tmax*. Default is *RMIN=1.e-9*. If the .OPTIONS parameter *rmin* is not specified and .TRAN statement parameter $tstep \geq 0.1$, then $delmin = 1.e-15$.
TMAX (DELMAX)=val	Sets the maximum internal *timestep* used during Transient analysis, and has the same effect as the *tmax* parameter in the .TRAN statement. The value used in the .TRAN statement will override this value.
VSTA=val	Resets the limit on the maximum voltage change from one timepoint to the next. Default is 1000. If the circuit contains MOS devices default VSTA value is recalculated at each time step as: $$VSTA = max(VSTALIM, |V|_{max} \cdot 0.2).$$ If the circuit contains controlled sources default VSTA value is: $$VSTA = max(VSTALIM, |V|_{max} \cdot 0.1)$$ where *VSTALIM* is the .OPTIONS parameter, and $|V|_{max}$ is the maximum voltage in the circuit at the current time point.
VSTALIM=val	Resets the low limit for calculated default value: $$VSTA = max(VSTALIM, |V|_{max}/K)$$ where *Vmax* is the maximum voltage in the circuit at current time point, *K*=5 if the circuit contains MOS devices, and *K*=10 if the circuit contains voltage/current controlled sources. Default value of VSTALIM is 0.8 Volts. If .OPTIONS ACCURATE<=1> is specified, the default value is 0.1 Volt.

General

BREAKSLOPE (SLOPETOL)	Sets a lower limit for the relative difference in slopes between two consecutive PWL segments. If the difference is less than the BREAKSLOPE value, the intermediate point between the two segments is ignored. Default is 0 (OFF).
ITL5=val	Resets the Transient analysis total iteration limit. SmartSpice terminates the Transient analysis when the total number of iterations equals *ITL5*. The option ITL5=0 allows an infinite number of iteration. Default is 0.
KEEPFLAT HIERSUBCKT	This options is used to control subcircuits creating in -fast mode. .OPTION KEEPFLAT="sub1 sub3" This syntax means that subcircuits sub1 and sub3 will be parsered like in regular SmartSpice mode. .OPTION HIERSUBCKT="sub2" This syntax means that subcircuit sub2 will be parsered in -fast mode. All other subcircuits will be parsered in regular mode.
LIMPTS=val	Maximum number of points in the Transient analysis. SmartSpice terminates the Transient analysis when the number of time points equals the LIMPTS value. Default is infinite.
LOGIC=val	If LOGIC=1 or 2, then event-driven transient simulation of the circuit is performed. This option can be used when the circuit is described by an analog behavioral device level that is purely digital. Setting LOGIC=2 allows the waveforms to be correctly displayed as digital signals. Default is 0.
MINBREAK	Effects minimum separation time between breakpoint values for the breakpoint table. If the new insertion in the breakpoint table is closer (in time) than the MINBREAK value to the existing breakpoint, then SmartSpice ignores the new breakpoint. The option MINBREAK has an alias of TIMERES. Default is *TMAX/2.0E4*.
REBINNING_VERBOSE	By default, printing a table of rebinned instances is disabled. The option REBINNING_VERBOSE has been introduced to enable printing of the rebinning table. You must specify this option in netlist to print out the rebinning information during source and/or simulation stage.
SEED	Specifies initial seed value for random number generators used in HSPICE compatible Monte-Carlo analyses. Default is 1.
TTICK	The smallest time resolution in event-driven transient simulation. If a delay type analog behavioral device has the parameter *DELAY* < *TTICK*, then this parameter is set to zero (shorted nodes). Default is *1.0E-12*.

NOSINSOURCEBREAK	Controls breakpoints for the sinusoidal sources. • NOSINSOURCEBREAK=0 - insert breakpoints for the sinusoidal sources • NOSINSOURCEBREAK=1 - do not insert breakpoints for the sinusoidal sources • NOSINSOURCEBREAK=2 - do not insert breakpoints for the sinusoidal sources only when option TMAX (DELMAX) is specified Default is 0 (OFF). In HSPICE compatibility mode the default is 2.

IBIS

D_IBIS	Specifies the location of IBIS files. Several paths can be specified. An IBIS file will be searched in all specified paths if the filename given on a B (buffer) statement is not an absolute path, and is not found in the directory from which the simulation runs or in the directory of the netlist containing the calling B statement.

Example

```
.OPTION D_IBIS='/home/mylogin/myIbisModels'
```

d_bis is also available as a variable, and can be set in SmartSpice .ini files. For example:

```
set d_ibis = ( . /home/mylogin/myibismodels )
```

Note: Even though the word d_ibis is not case-sensitive (like other option keywords), the arguments of this option are case-sensitive.

Verilog-A

VERILOGA-ARGS	Specifies the options for the Verilog-A compiler. Details of acceptable options and other methods of settings options for Verilog-A, can be found in Section 8.7 Options for Verilog-A.

Example

```
.OPTION  VERILOGA-ARGS = "-debug"
```

VLG_TO_VSRC	This option can be used for debugging purposes. It creates a file VLG_TO_VSRC.lib, where all VLG device terminals are connected to corresponding VSRC devices with voltage level equals value.

Syntax

```
.option vlg_to_vsrc=value
```

Examples

```
.option vlg_to_vsrc=5.0

// VA file for module stim
‘include "constants.vams"
‘include "disciplines.vams"
module stim(dpa_out_a, dpa_out_b, en_out);
output [25:0] dpa_out_a;
output [25:0] dpa_out_b;
output en_out;
...
// SPICE NETLIST
* Stage 1 - Generate a file "VLG_TO_VSRC.lib"
...
YVLG1 net1<25> net1<24> net1<23> net1<22> net1<21> net1<20>
+ net1<19> net1<18> net1<17> net1<16> net1<15> net1<14> net1<13>
+ net1<12> net1<11> net1<10>
+ net1<9> net1<8> net1<7> net1<6> net1<5> net1<4> net1<3> net1<2>
+ net1<1> net1<0> net0<25> net0<24> net0<23> net0<22> net0<21>
+ net0<20> net0<19> net0<18> net0<17> net0<16> net0<15> net0<14>
+ net0<13> net0<12> net0<11> net0<10> net0<9> net0<8> net0<7>
+ net0<6> net0<5> net0<4> net0<3> net0<2> net0<1> net0<0> net2 stim
.option vlg_to_vsrc=5.0
.END
* Stage 2 - Usage of a file "VLG_TO_VSRC.lib"
...
.include "VLG_TO_VSRC.lib"
.END
```

SmartSpice generates a file `VLG_TO_VSRC.lib`. On second stage a VLG instance is replaced by including the file `VLG_TO_VSRC.lib` into the netlist. A file `VLG_TO_VSRC.lib` contains VSRC statements.

Example

```
v_net1<25> net1<25> 0 5.0000
v_net1<24> net1<24> 0 5.0000
v_net1<23> net1<23> 0 5.0000
```

RubberBand

RBDISPLAYPARASITIC	Accepts the following values: • 0 - the rubberband tab “Devices” contains all elements • 1 - the rubberband tab “Devices” contains parasitic R and C elements only • 2 - the rubberband tab “Devices” contains parasitic R elements only • 3 - the rubberband tab “Devices” contains parasitic C elements only • 4 - the rubberband tab “Devices” contains parasitic L elements only • 5 - the rubberband tab “Devices” contains parasitic K elements only

RLC-Reduction

RCLEVEL	• RCLEVEL=-1 (default) - disable RLC-reduction; • RCLEVEL=0 - perform RLC-reduction on subcircuits; • RCLEVEL=1 - perform RLC-reduction on subcircuits with post-filtering; • RCLEVEL=2 - perform RLC-reduction on subcircuits and top-level netlist; • RCLEVEL=3 - perform RLC-reduction on subcircuits and top-level netlist with post-filtering; • RCLEVEL=4 - perform RLC-reduction on expanded netlist; • RCLEVEL=5 - perform RLC-reduction on expanded netlist with post-filtering; • RCLEVEL=6 - perform RLC-reduction for DSPF annotated netlist; • RCLEVEL=7 - perform RLC-reduction for DSPF annotated netlist with post-filtering;
RMIN_RC	R values threshold. Default is 1e-3
CMIN_RC	C values threshold. Default is 1e-22
LMIN_RC	L values threshold. Default is 1e-9
RMAX_RC	Maximum resistance for the optimized RLC-reduction
TAU_MIN_RC	Nodal time constant for the time domain method. Default is 1e-12.
METHOD_RC	• METHOD_RC=0 - Scattering Parameter based Macromodeling method; • METHOD_RC=1 (default) - Time domain method • METHOD_RC=2 - Omit main reduction
REDUCE_ALL_RLC	Controls which R, L, and C elements in the circuit can be selected for reduction. If the option is set to 1, the R, L, and C elements with any set of parameters are selected for reduction, except the elements that have models. If the resistance, inductance, or capacitance of a parasitic element is specified by a parameter or expression, such elements cannot be reduced. If the option is set to 0, only the R, L, and C elements with single resistance, inductance, and capacitance parameter can be selected for reduction. Default is 1.
GROUND_NODE_RC	Ground node name for RLC-Reduction.
REDUCE_PARALLEL_MOS	Perform parallel reduction of MOSFETs. Default is 0 (OFF).
SUBNODE_DELIMITER	Subnode delimiter for RLC-Reduction. Default is ‘:’.
REDUCE_PARALLEL_RLC	Perform parallel reduction of RLCs. Default is 1 (ON).

REDUCE_SERIES_RLC	Perform series reduction of RLCs. Default is 1 (ON).

Expression Accelerator (EAC) Library

EACFLAG	Connect EAC library to SmartSpice
EACPATH	Path to EAC library directory

Syntax

```
.OPTION EACFLAG=1 EACPATH='path_to_eac_library_directory'
```

User Defined Hierarchy Partitioning Options

Here is the list of options for user defined hierarchy partitioning algorithm:

.option separator_level=number	All cells starting from this level of hierarchy will form blocks for independent calculation. All elements starting from top level to separator level exclusively will form interconnect network. If not defined auto detection algorithm will calculate separator level to minimize interconnect network and maximize number of separate blocks..
.option hpp_block_size=number	Size of block sets number of internal nodes for each block. After defining separator level all cells there will be reordered and merged into composite blocks to reduce number of small blocks on matrix diagonal. This option specifies desired size of block. Default value = 100.

Graph Partitioning Options

```
-partition startup key
```

Initializes graph partitioning for HPP mode. In this mode circuit graph partitioned to form blocks. This key should be used with `-hpp` startup key. By default number of `partition=4`, but it can be redefined with `.option hpp_partition_count`. If some partitions too small they will be merged into one. Related options:

.option hpp_partition_count=number	Set desired partitions count for graph partitioning. Default value = 4.

Example

```
smartspice -hpp -partition
smartspice -hppmultirate -partition
```

Common Options

.option hpp_nestdiss_intercon=number	Specifies number of levels of nested dissection which will be used to decompose interconnection matrix to provide more parallelism in calculation. Can speedup calculation on multiple CPUs. Default value = 0.

Domain Decomposition Solver

As mentioned above DDS solver finds solution for set of small matrices instead of one global matrix. This gives higher parallelism and less factorization time. DDS solver can be engaged by:

```
.option solver=dds
startup key -forcesolver dds
```

Isomorphism

Under DDS solver and user defined hierarchy, the partitioning block isomorphism feature available. During simulation topologically, isomorphic blocks share their results. So instead of simulating all blocks only one of them simulated and others use calculated results. Always check simulation results for accuracy.

.option hpp_block_isomorphism	Turns on isomorphic blocks detection.
.option hpp_block_isomorphism_reltol	Isomorphism relative voltage error tolerance for blocks. Default value 1e-3.
.option hpp_block_isomorphism_abstol	Isomorphism absolute current error tolerance for blocks. Default value 1e-9.
.option hpp_block_isomorphism_vntol	Isomorphism absolute voltage error tolerance for blocks. Default value 50.e-6.

Chapter 4
Expressions

An expression is an algebraic combination of:

- simulation unknown variables (node voltages and branch currents)
- device/model output parameters
- numbers
- constants
- parameter labels
- global parameters (temperature (`TEMP`))
- global variables (time (`TIME`), time step (`TSTEP`) for transient analysis, and frequency (`HERTZ`) for AC analysis)
- operations
- functions
- predefined macros
- user-predefined function (by using `.DEFINE`, `.FUNC` and `.PARAM` statements)

Expressions can be used at input deck statements to:

1. Specify (by means of calculations) the value of a parameter, instead of the number or parameter label specifications;
2. Calculation (measurement) of output variable (vectors) parameters (possible as vectors too).

In the first case, the function and macros arguments can only be scalar values (vector component is the scalar). The result of function performing will be scalar as well. In the second case, arguments can be both the vector and the scalar values, and cannot be global variables (`TIME`, `TSTEP`, `HERTZ`).

Expressions must be enclosed in single quotes (’) or curly brackets ({}).

4.1 Functions

SmartSpice supports the functions listed in Table 4-1. The table gives the definition for each function, and shows the types of function arguments.

Table 4-1 Functions

Function	Scalar	Vector	Description	Notes
`abs(x)`	+	+	Absolute value x	For complex X: *abs(X) = magnitude(X)*
`acos(x)`	+	+	Arc cosine x	
`acosh(x)`	+	+	Hyperbolic arc cosine x	For complex X: *acosh(X) = acosh(reX)*
`asin(x)`	+	+	Arc sine x	
`asinh(x)`	+	+	Hyperbolic arc sine x	For complex X: *asinh(X) = asinh(reX)*

Function	Scalar	Vector	Description	Notes
`arctan(x)`	+	-	Arc tangent x	For vector use macros *arctan(x)*
`atan(x)`	+	+	Arc tangent x	
`atanh(x)`	+	+	Hyperbolic arc tangent x	For complex X: *atanh(X) = atanh(reX)*
`avg(x)` or `mean(x)`	-	+	Average value of the vector x: $avg(x) = \frac{\frac{1}{N} \times \sum_{i=1}^{N} h_i \times x_i}{\sum_{i=1}^{N} h_i}$	Result is scalar. For complex vector X: *avg(X)=avg(reX)+j* x *avg(imX)*.
`ceil(arg1)`			Calculates the ceiling of a value `arg1`	
`convol(x)`	-	+	Convolution of the complex vector $X(f_k)$, f is vector of frequencies, $k=1,..,N$: $conv(X(f_k)) = \sum_{i=1}^{k} \Delta_i \times reX_i \times imX_{k-i}$, $\Delta_i = f_i - f_{i-1}$	
`cos(x)`	+	+	Cosine of x	
`cosh(x)`	+	+	Hyperbolic cosine of x	
`d(x)` or `deriv(x)`	-	+	Derivative of the vector x with respect to its scale variable	A SmartSpice variable `dpolydegree` specifying the degree of the polynomial used for derivative calculation.
`db(x)`	-	+	Magnitude of the complex vector in decibels: $db(x) = 20 \times \log(mag(x))$	

Function	Scalar	Vector	Description	Notes
`erf(x)`	+	+	Error function $erf(x) = \frac{2}{\sqrt{\pi}} \times \int_0^x e^{-t^2} dt$	
`erfc(x)`	+	+	Error function complement *erfc(x) = 1 - erf(x)*	
`exp(x)`	+	+	Exponential: *e* raised to power *x*	
`floor(arg1)`	+	+	Calculates the floor of a value `arg1`	
`fmod(arg1, arg2)`	+	+	Calculates the float-point remainder of `arg1`/`arg2`	
`gd(x)` or `gdelay(x)`	-	+	Group delay of the complex vector x: $-\frac{\partial}{\partial f} phase$	Only complex vector
`im(x)` or `imag(x)`	-	+	Imaginary part of the complex vector x	Only complex vector
`int(x)` or `trunc(x)`	+	+	Returns the integer part of x	For complex X: *int(x) = int(reX)*
`integral(x)` or `s(x)`	-	+	Integral of a vector x over its scale variable	Result is scalar. For complex X: *integral(x) = integral(mag(X))*
`interpolate(x)`	-	+	Result of interpolating the named vector onto the scale of the named plot. This function uses the variable *polydegree* to determine the degree of interpolation. This is useful if the argument belongs to a plot that is not the current plot.	Restrictions are that the current scale, the old scale, and the argument must be real, and both scales must be either strictly increasing or strictly decreasing.
`j(x)`	-	+	*i* (square root of -1) times vector x	
`length(x)`	-	+	Length of the vector x	Result is scalar.

Function	Scalar	Vector	Description	Notes
`limit(x,min,max)`	+	-	Result is `min` if `x < min`, `max` if `x > max`, and `x` otherwise	pspice compatibility function. Example: `R1 x xx r='limit (-0.7, 0.1, 0.6) + 8'`
`log10(x)`	+	+	Logarithm (base 10) of x	
`log(x)` or `ln(x)`	+	+	Natural logarithm (base *e*) of x	
`mag(x)` or `magnitude(x)`	-	+	Magnitude of the complex vector x	
`max(x,y)`	+	+	Maximum of two values, if *x>y* then *max(x,y)=x*; otherwise *=y*	For complex values function uses its *magnitude* values. The arguments can have different type (one - scalar, another - vector). The result always is scalar.
`max(x1, x2,...)`	+	-	Maximum of list values	For complex values function uses its *magnitude* values. Number of values < 20.
`max(x)`	-	+	Maximum of the vector x	For complex x: *max(x) = max(mag(x))*
`min(x,y)`	+	+	Minimum of two values, if *x<y* then *min(x,y)=x*; otherwise *=y*	For complex values function uses its *magnitude* values. The arguments can have different type (one - scalar, another - vector). The result always is scalar.
`min(x1, x2,...)`	+	-	Minimum of list values	For complex values function uses its *magnitude* values. Number of values < 20.
`min(x)`	-	+	Minimum of the vector x	For complex x: *min(x) = min(mag(x))*
`norm(x)`	-	+	Vector x is normalized to 1 (the magnitude of the largest component will be 1)	
`par(x)`	-	+	Returns vector x: *par(x)=x*	
`ph(x)` or `phase(x)`	-	+	Phase of vector (in radians)	Only complex vector

Function	Scalar	Vector	Description	Notes
`pos(x)`	-	+	result is a vector containing the values 1, if the corresponding element of x is positive, or 0 if the corresponding element of x is negative.	Only real vector
`pow(x,y)`	+	-	Absolute power: absolute value of x raised to integer part of y: *pow(x,y) = abs(x)^int(y)*	For vectors use macros pow(x,y)
`pwr(x,y)`	+	-	Signed power: absolute value of x (with sign of x) raised to y power: *pwr(x,y) = sgn(x)*abs(x)^y*	For vectors use macros pwr(x,y)
`re(x)` or `real(x)`	-	+	Real part of vector x	
`rms(x)`	-	+	Root Mean Square: $rms(x) = \sqrt{\frac{\sum_{i=1}^{N} x^2{}_i}{N}}$	Result is scalar. For complex vector X: *avg(X)=rms(reX)+j* x *rms(imX).*
`rnd(x)`	+	+	Returns a random integer number (or the vector with each component a random integer number) between 0 and absolute value x (absolute value of the vector x components)	
`round(arg1)`	+	+	Returns an integer closest in value to the argument `arg1`	
`sgn(x)`	+	+	Sign of value, if *x<0* then *sgn(x) = -1*; if *x=0* then *sgn(x) = 0*; if *x>0* then *sgn(x) = 1*	For complex vector X: *sgn(X) = sgn(reX).* Function sign(x,y) is macros, used *sgn(x)*
`sin(x)`	+	+	Sine of x	
`sinh(x)`	+	+	Hyperbolic sine of x	

Function	Scalar	Vector	Description	Notes
`smabs (x,eps)`	+	+	Returns absolute value x. $smabs(x, eps) = \sqrt{x^2 + eps}$	For complex x: *smabs*(*x*,*eps*)=*abs*(*x*)
`smmax(x,y,eps)`	+	+	Returns maximum of two values `x,y`. $smmax(x, y, eps) = y + 0.5 \cdot ((x-y) + \sqrt{(x-y)^2 + eps})$	For complex values x,y: *smmax*(*x*,*y*,*eps*) = *max*(*x*,*y*)
`smmin(x,y,eps)`	+	+	Returns minimum of two values `x,y`. $smmin(x, y, eps) = x - 0.5 \cdot ((x-y) + \sqrt{(x-y)^2 + eps})$	For complex values x,y: *smmin*(*x*,*y*,*eps*) = *min*(*x*,*y*)
`smsgn(x,eps)`	+	+	Returns sign of value x. If *x* < - *eps* then *smsgn*(*x*,*eps*) = *-1*, If *x* > *eps* then *smsgn*(*x*,*eps*) = *1*, else *smsgn*(*x*,*eps*) = *sin*((*pI*/*2*)*(*x*/*eps*))	For complex vector X: *smsgn*(*x*,*eps*) = *sgn*(*x*)
`sqrt(x)`	+	+	Square root of x	For real value x: *sqrt(x) = sqrt(abs(x))*
`tan(x)`	+	+	Tangent of x	
`tanh(x)`	+	+	Hyperbolic tangent of x	

Function	Scalar	Vector	Description	Notes
`valif(log_cond,x,y)` or if `(log_cond, x,y)`- in `'-pspice'` mode	+	-	Logical expression: *if(log_cond) then x else y*	*log_cond* is the a logical condition, which consist of two expressions compared using operators: ==; !=; <; >; <=; >= or their alphabetical equivalents: eq, ne, lt, gt, le,ge.
`unitvec` (number)	+	+	Result is a vector of length *number* composed of ones. If *number* is a vector, then *number = int(number[0]),* i.e. the first value of the vector truncated to an integer.	Result always is a vector.
`vector` (number)	+	+	Result is a vector of length *number*, with elements *0, 1, ..., number-1*. If *number* is a vector, then *number = int(number[0]),* i.e. the first value of the vector truncated to an integer.	Result always is a vector.

4.2 Predefined Macros

Table 4-2 lists the predefined macros supported by SmartSpice. The predefined macro arguments are vectors.

Table 4-2 Predefined Macros

Macro	Definition	Definition with -hspice flag	Notes
`arctan(x)`	*atan(x)*		
`capp(x,w)`	*-im(x)/(w*mag(x)*mag(x))*		Only for complex vector `x`
`caps(x,w)`	*-1/(w*im(x))*		Only for complex vector `x`
`dv(x)`	*deriv(v(x))*		
`dv(x,y)`	*deriv(v(x)-v(y))*		
`db10(x)`	*db(x)/2*		
`db20(x)`	*db(x)*		
`dbm(x)`	*db(x/2)/2+30*		
`g(x)`	*gdelay(x)*		Only for complex vector `x`
`idb(x)`	*db(i(x))*		
`ii(x)`	*im(i(x))*		Only for complex vector `i(x)`
`img(x)`	*im(x)*		Only for complex vector `x`
`indp(x,w)`	*mag(x)*mag(x)/(w*im(x))*		Only for complex vector `x`
`inds(x,w)`	*im(x)/w*		Only for complex vector `x`
`ip(x)`	*ph(i(x))*		Only for complex vector `i(x)`
`ir(x)`	*re(i(x))*		Only for complex vector `x`
`it(x)`	*gdelay(i(x))*		Only for complex vector `i(x)`
`m(x)`	*mag(x)*		Only for complex vector `x`
`p(x)`	*v(node) * i(device)*		Computes power dissipation
`pow(x,y)`	*abs(x)^int(y)*		
`pwr(x,y)`	*sgn(x)*(abs(x)^y)*		
`r(x)`	*re(x)*		Only for complex vector `x`
`resp(x)`	*mag(x)*mag(x)/re(x)*		Only for complex vector `x`
`ress(x)`	*re(x)*		Only for complex vector `x`
`sign(x,y)`	*sgn(y)*abs(x)*		
`sv(x)`	*integral(v(x))*		

Macro	Definition	Definition with -hspice flag	Notes
`sv(x,y)`	*integral(v(x)-v(y))*		
`stp(x)`	*0.5 + 0.5*sgn(x)*		
`v(x,y)`	*vdiff(x,y)*		
`vdb(x)`	*db(v(x))*		Only for complex vector `v(x)`
`vdb(x,y)`	*db(v(x)-v(y))*	*vdb(x)-vdb(y)*	Only for complex vector `v(x), v(y)`
`vdiff(x,y)`	*v(x)-v(y)*		Only for complex vector `v(x), v(y)`
`vgd(x)`	*gdelay(v(x))*		Only for complex vector `v(x)`
`vgd(x,y)`	*gdelay(v(x)-v(y))*		Only for complex vector `v(x), v(y)`
`vi(x)`	*im(v(x))*		Only for complex vector `v(x)`
`vi(x,y)`	*im(v(x)-v(y))*	*vi(x)-vi(y)*	Only for complex vector `v(x), v(y)`
`vm(x)`	*mag(v(x))*		Only for complex vector `v(x)`
`vm(x,y)`	*mag(v(x)-v(y))*	*vm(x)-vm(y)*	Only for complex vector `v(x), v(y)`
`vp(x)`	*ph(v(x))*		Only for complex vector `v(x)`
`vp(x,y)`	*ph(v(x)-v(y))*	*vp(x)-vp(y)*	Only for complex vector `v(x), v(y)`
`vr(x)`	*re(v(x))*		Only for complex vector `v(x)`
`vr(x,y)`	*re(v(x)-v(y))*	*vr(x)-vr(y)*	Only for complex vector `v(x), v(y)`
`vt(x)`	*gdelay(v(x))*		Only for complex vector `v(x)`
`vt(x,y)`	*gdelay (v(x))-gdelay (v(y))*		Only for complex vectors `v(x)` and `v(y)`
`w(x)`			Alias to `p(x)`

Examples

```
vdb(6)      Magnitude of the voltage at node 6 in decibels
vi(5)       Imaginary part of the voltage at node 5
vm(3,4)     Magnitude of the voltage across nodes 3 and 4
vp(8,9)     Phase of the voltage across nodes 8 and 9
vgd(11)     Group delay of the voltage at node 11
dv(12,13)   Derivative of the voltage across nodes 12 and 13
sv(14)      Integral of the voltage at node 14
vr(15,16)   Real part of the voltage across nodes 15 and 16
```

4.3 Constants

Table 4-3 lists the constants supported by SmartSpice. The values are in MKS unit.

Table 4-3 Constants

Constant	Description
boltz	Boltzmann constant *k*=1.38062e-23 joules/K
c	Speed of light *c*=299792500 m/sec
e	Base of natural logarithms e=2.7182818284590452354
echarge	Charge of an electron *q*=1.60219e-19 C
kelvin	Temperature of absolute zero $T_0 = -273.15°C$
pi	$\pi = 3.1415926535897932384$
planck	Planck's constant *h*=6.62620e-34 joules/sec
true	Defined as 1
false	Defined as 0
yes	Defined as 1
no	Defined as 0

4.4 Global Parameters

Table 4-4 lists the global parameters supported by SmartSpice.

Table 4-4 Global Parameters

Parameter name	Description
`temp`	The circuit temperature in °C
`temper`	Always has the same value as the global parameter `temp`. Can be used in any parameter and vector expressions in the input deck, but cannot be directly modified by parametric analysis commands. Can be changed by varying the circuit temperature in the parametric analysis commands `.TEMP`, `.MODIF`, `.ST` and `.DC`.

4.5 Operators

Table 4-5 lists the operations supported by SmartSpice.

The results of the modulo operation is the remainder when the first number is divided by the second number. Both modulo operation arguments are rounded down to the nearest integer before the operation is performed.

The comma operation result is the notation *a,b* which refers to the complex number with real part *a* and imaginary part *b*. It is equivalent to the expression *a + b*. The notation *a,b* cannot be used in the argument to a macro and *min*, *max* functions since commas are used to separate the arguments and parentheses can be ignored.

Table 4-5 Operations

Operation Type	Operator	Meaning	Example	Synonym	Notes
Algebraic	+	Plus	*a + b*		
	+	Unary Plus	*+a*		
	-	Minus	*a - b*		
	-	Unary Minus	*-a*		
	*	Multiply	*a*b*		
	/	Divide	*a/b*		
	^	Power	*a^b*		
	**	Power	*a**b*		
	%	Modulo	*a%b*		
	,	Comma	*a,b*		
Logical	&	And	*a&b*	and	for vector *a,b*
	|	Or	*a|b*	or	for vector *a,b*
	&&	And	*a&&b*	and	for scalar *a,b*
	||	Or	*a||b*	or	for scalar *a,b*
	!	Unary negation	*!a*	not	
	[cond] ? x:y	Ternary conditional expression			See Example below
Relational	<	Less than	*a<b*	lt	
	>	Greater than	*a>b*	gt	
	<=	Less than or equal	*a<=b*	le	
	>=	Greater than or equal	*a>=b*	ge	
	==	Equal	*a==b*	eq	
	!=	Not equal	*a!=b*	ne	

Example

```
.param Param1=5 Param2=10
+ Test='(Param>Param2) ? 1 : 2'
+ Test2='5*((Param1<Param2) ? (Param+1) : (Param2+2))'


RS1    1  2    'distr(5,5,6, 0.75)'
.param distr(b,n,w,skew1bm1w) ='n +  2. *shift* (w - n) * (b - n) /
(w == n
+ || n == b || w == b ? 1. : w - b + 0.75*shift * (w + b - 2. * n))'
```

Chapter 5
Commands

5.1 Introduction

This chapter describes SmartSpice commands. SmartSpice commands are issued from the command line of the SmartSpice Main window, or directly from a terminal prompt when SmartSpice is run in command mode. SmartSpice in command mode has a command-shell interface with many C-shell like features. To run SmartSpice in command mode, use the `-c` command line option. For example:

```
SmartSpice -c
```

starts SmartSpice in command mode, creates a shell, and (after initialization) prompts you to enter a command.

5.2 Commands

When a command is entered, it is interpreted in one of the following ways:

1. It can be a predefined command, in which case it is executed.
2. It can be an alias, in which case the line is replaced with the result of an alias substitution. The substituted line is reparsed.
3. It can be the name of a circuit file, in which case it is loaded as if it were a source command.
4. It can be the name of a file containing a command script, in which case SmartSpice searches directories in the current search path for the file, and executes the script in the file. Command scripts are discussed later in this chapter.
5. It can be an assignment statement, which consists of a vector name, an “=” (equal) symbol, and an expression (see the syntax for the `let` command), in which case it is executed as if it were preceded by the word `let`.
6. It can be a UNIX command, in which case (if the variable `unixcom` is set) it is executed as though it were typed to the shell.

5.2.1 Command Descriptions

This section describes all the commands available in the SmartSpice command input mode. For commands having counterparts with identical syntax to analysis statements (AC, DC, TRAN, NET, NOISE, DISTO, etc.), only a brief description is given. More information about the statements that correspond to these commands can be found in Chapter 3 Statements.

ac

Syntax

```
ac <.AC statement arguments >
```

Performs an AC analysis. SmartSpice uses the option values specified in the current input deck.

alias

Syntax

```
alias <word> <text>
```

Causes `word` to be aliased to `text`. Whenever a command line beginning with `word` is typed, `text` is substituted. Arguments are either appended to the end, or substituted in if history characters are present in the text. See Section 5.3“C-Shell Like Features”.

Examples

```
alias pwd shell pwd
alias a alias
alias u unalias
alias h history
alias man help
alias pcomb2 plot ‘!:1’ ‘!:2’ combplot
```

asciiplot

Syntax

```
asciiplot plot_arguments
```

This command produces an ASCII plot of the vectors. See the plot command description for the syntax of `plot_arguments`. The variables `width`, `height` and `nobreak` determine the plot width and height; if there are page breaks, respectively. `asciiplot` uses a simple sort of linear interpolation, therefore it is not possible to use `asciiplot` with a non-monotonic x-scale. For example: `sin(time)`.

Examples

```
asciiplot v(1) v(2) log(v(3))
asciiplot all > file1
```

The first example produces an `asciiplot` of vectors `v(1)`, `v(2)`, and `log(v(3))`. The second example stores `asciiplots` of all vectors from the current plot to the file `file1`.

calibratedevices

Syntax

```
calibratedevices device_type <checkcrossiv>
```

`device_type` must be one of the following types:

- `d` - diodes
- `m` - MOSFET
- `checkcrossiv` - runs simulation for generated decks and finds crossings in I-V characteristics.

The command `calibratedevices` allows you to take I-V characteristics for all circuit devices, and will generate the model cards, the input decks, the options and script file to take I-V characteristics for the specified device type. The files will be stored in the `Calibrate/device_type` folder next to the input deck file.

The command `calibratedevices` can generate decks using Spectre syntax for MOSFET devices in Spectre-compatibility mode.

cancel

Syntax

```
cancel
```

Aborts and quits from the simulation loop.

cd

Syntax

```
cd <directory>
```

Changes the current working directory to `directory` or the user home directory if no `directory` is specified.

Examples

```
cd
cd ../SmartSpiceDir
```

The first example changes the working directory to the user home directory. The second example changes the working directory to `../SmartSpiceDir`.

checkcrossiv

Syntax

```
checkcrossiv <rawfile>
```

The command `checkcrossiv` finds crossings between sweeps within a specified raw-file and saves results into a file `iv_cross.dat`. Parameter `rawfile` is optional and can be `.sw?`-file or raw-file.

Example 1

```
source m.x7.m2.sp
run batchprint
checkcrossiv
```

The `m.x7.m2.sp` must have `.DC` with sweep, for example:

```
.DC VDS -3.3 0.5 0.01 VGS -2 0 0.1
```

Example 2

```
checkcrossiv mos123.sw0
```

Example 3

```
checkcrossiv mos123.raw
```

The first example sources `m.x7.m2.sp`, simulates the current circuit and finds crossings between sweeps in the current plot.

The second example finds crossings between sweeps within a specified hspice compatible sw0-file.

The third example finds crossings between sweeps within a specified raw-file.

For examples for Figure 5-1 the command issues warnings:

```
Analysing plot 'dc1'...
Analysing vector 'i(vbs)':... start:
Warning: crossing at sweep 'vg' = 0 and 0.1, where 'vds' =
1.222222E-002 and 'i(vbs)' = 2.434691E-014
Warning: crossing at sweep 'vg' = 0.1 and 0.2, where 'vds' =
3.200000E-002 and 'i(vbs)' = 6.374465E-014
```

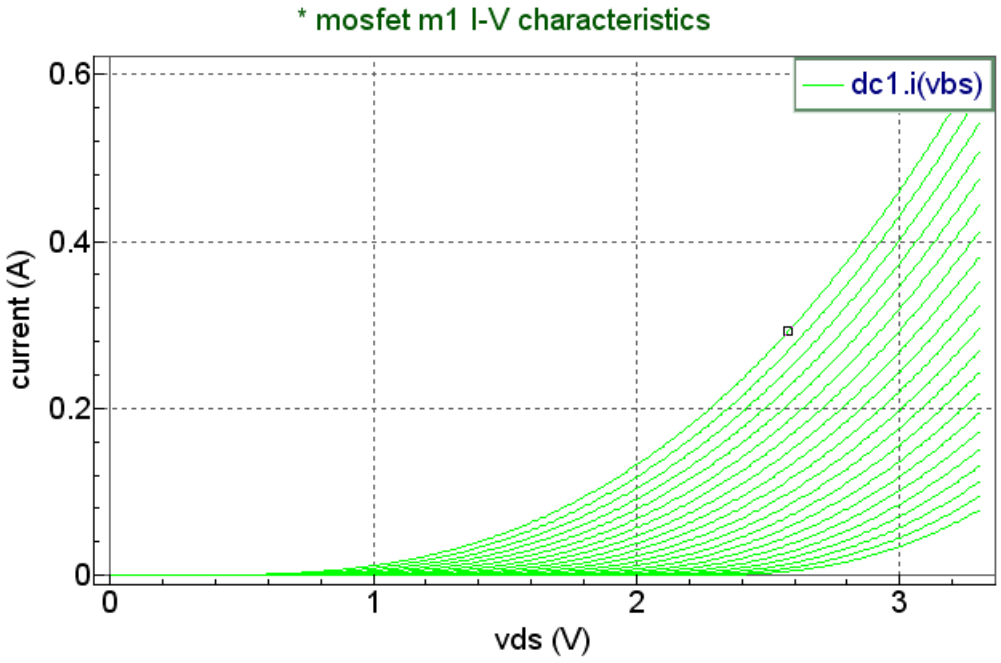


Figure 5-1 I(VBS) sweeps

checkfsdbresolution

Syntax

```
checkfsdbresolution fsdb_rawfile <-c>
```

The command checkfsdbresolution gets information about resolutions (tolerances) from a rawfile and prints FSDB_VPRBTOL and FSDB_IPRBTOL values. If '-c' flag is present it forces calculation of FSDB_VPRBTOL and FSDB_IPRBTOL resolutions.

Examples

```
checkfsdbresolution ex1.fsdb
```

The first example searches FSDB_VPRBTOL and FSDB_IPRBTOL vectors in ex1.fsdb file. If these resolution vectors are found the command prints the resolutions; otherwise, if the resolution vectors does not exist, the command scans all current and voltage signals and calculates the minimal FSDB_VPRBTOL and FSDB_IPRBTOL resolutions.

```
checkfsdbresolution ex1.fsdb -c
```

The second example shows how to force calculation minimal FSDB_VPRBTOL and FSDB_IPRBTOL resolutions for all current and voltage signals.

cmcstat

In the continuous model parameter algorithm all devices are grouped with respect to the arguments of the formula used for model parameter. This command is used to see how many different groups are created in the target model

compose

Syntax

```
compose vecname param1 = value1 <param2 = value2...>
compose vecname values value1 <value2 ...>
```

Both forms of this command create a new vector vecname. In the first form, the values in the vector are determined by the parameters given. In the second form, the given values are used to create the new vector.

The possible parameters are:

start	The value at which the vector should start.
stop	The value at which the vector should stop.
step	The difference between successive elements.
lin	The number of points, linearly spaced.
log	The number of points, logarithmically spaced.
dec	The number of points per decade, logarithmically spaced.
center	Where to center the range of points.
span	The range of points.

gauss	The number of points in the gaussian distribution.
mean	The mean value for the gaussian distribution.
sd	The standard deviation for the gaussian distribution.
random	The number of randomly selected points.
pool	The name of a vector (already defined) from which to get random values. Default is unitvec(npoints).

Compared to the `let` command, the `compose` command is a more powerful method of creating new vectors.

Examples

```
compose new1 values 10 10.5 11 12 15
compose new2 start=100 stop=200 step=5
```

- The first example creates new vector `new1`, with dimension `5` and values `10`, `10.5`, `11`, `12` and `15`.
- The second example creates new vector `new2` with values `100`, `105`, `110`, ..., `195`, `200`.

cont

Syntax

```
cont
```

Continues analysis from the point where it was paused.

cutplot

Syntax

```
cutplot  from=start  to=stop
```

from, to	The mandatory keywords.
start, stop	The starting and ending scale vector values define the interval where current plot vector values will be duplicated in a new plot.

Creates a new plot, and copies vector data on a specified interval from the current plot.

Example

```
cutplot    from=10ns to=300ns
```

The `start` and `stop` values have to be both specified.

dc

Syntax

```
dc <.DC statement arguments>
```

Performs a DC sweep analysis. SmartSpice uses the option values specified in the current input deck.

define

Syntax

```
define <function (arg1,...) <expression > >
```

Defines a macro with the name `function` and arguments `arg1,...` to be `expression`. When the function is used, the actual arguments are substituted for the formal arguments. If `expression` is not present, the definition for `function` is printed. If `define` is entered on its own without any function name, then all currently active definitions are printed.

Examples

```
define
define max
define capp(z,w) (-in(z))/(w*mag(z)*mag(z))
define sgn(x) (x)/(abs(x)+1)
define resp(z) mag(z)*mag(z)/re(z)
```

- In the first example, the `define` command without arguments prints all currently active macros.
- The second example prints the definition for the macro `max`.
- The last three examples define new macros `capp`, `sgn` and `resp`, respectively.

deftype

Syntax

```
deftype v <vectype <abbrev> >
deftype p <plottype word1 <word2... >>
```

The first form of this command defines a new vector type `vectype`. The vector type `vectype` can then be used as a vector type specification in a rawfile, and will be used by the `display` command to indicate the vector type. It can also be used as an argument of the `settype` command. If `abbrev` is specified, then the vector `vecname` of type `vectype` can be named `abbrev (vecname)` if `vecname` does not contain parentheses. Also, `abbrev` will be used as a unit name of axes labels for on-screen plotting.

The second form of this command defines a new plot type `plottype`. A plot will be of the type `plottype` if its name contains any of the words: `word1, word2, ...`.

If used without arguments, the first and second forms list all currently defined types for vectors and plots, respectively.

Examples

```
deftype v
deftype v resistance ohm
deftype v inductance H
deftype v mut_inductance
deftype p
deftype p newac AC_anal ac_anal
```

- The first example prints all currently defined vector types.
- The second example defines the new vector type `resistance` with the abbreviation `ohm`.
- The third example defines the new vector type `inductance`, with the abbreviation `H`.

- The fourth example defines the new vector type `mut_inductance`, without an abbreviation.
- The fifth example prints all currently active plot types.
- The sixth example defines the new plottype `newac`. This type will be assigned to any plot whose name contains one of the words: `AC_anal` and `ac_anal`.

destroycirc

Syntax

```
destroycirc  <all | circuitname1 ... | circuitnumber1 ... >
```

It destroys the data assigned to circuits `circuitname1,...`, or circuits with numbers `circuitnumber1,...`. This is needed when many large circuits are loaded and simulated. If the argument is `all`, all circuits will be destroyed. If no argument is given, the current circuit is destroyed.

Examples

```
destroycirc
destroycirc all
destroycirc 14 16 17
destroycirc diffpair rtlinv
```

- In the first example, the current circuit is destroyed.
- In the second example, all circuits are destroyed.
- In the third example, circuit numbers `14`, `16` and `17` are destroyed.
- In the fourth example, circuits with names that start with `diffpair` and `rtlinv` are destroyed.

destroyplot

Syntax

```
destroyplot <all | plotname1...>
```

It destroys the data in the named plot. This is needed when many large simulations are being done. It is advisable to run the `rusage` command occasionally when there is a possibility of running out of space. When the argument `all` is used, all plots except the constants plot will be destroyed. It is impossible to destroy the constants plot. If no argument is given, the current plot is destroyed.

Examples

```
destroyplot
destroyplot all
destroyplot dc1 tran2
```

- In the first example, the current plot is destroyed.
- In the second example, all plots (except the constants plot) are destroyed.
- In the third example, plots `dc1` and `tran2` are destroyed.

diff

Syntax

```
diff plotname1 plotname2 <vecname1 ... >
```

This command compares the vectors `vecname1...` in the plots `plotname1` and `plotname2`, and prints any values which differ significantly in corresponding vectors of the two plots. If no vector name is given, all vectors are compared. The variables `diff_abstol`, `diff_reltol` and `diff_vntol` are used to determine if two values are significantly different.

Examples

```
diff tran2 tran1
diff dc3 dc1 v(4) ic(ql)
```

- In the first example, all vectors from plots `tran2` and `tran1` are compared.
- In the second example, only vectors `v(4)` and `ic(ql)` from plots `dc3` and `dc1` are compared.

display

Syntax

```
display <all | vecname1... >
```

Lists the names, types and lengths of the vectors `vecname1,...`, and whether the vectors are real or complex. If the argument is `all`, or if no argument is given, all vectors from the current plot will be listed. Additional information is also given if there is a minimum or maximum value for the vector defined, this value is listed; if there is a default grid type or a default plot type, values are set; and if there is a default color or a default scale for the vector, this is noted. Additionally, one vector is labeled `[scale]`. When a command (such as `plot`) is given without an argument, this scale is used for the x-axis. It is always the first vector in a rawfile, or the first vector defined in a new plot.

Examples

```
display
display all
display v(1) i(R2) ig(m22)
display tran2.all tran3.v(4)
```

- The first and second examples both display the names, types and lengths of all vectors from the current plot.
- The third example displays the names, types and lengths of only three vectors: `v(1)`, `i(R2)` and `ig(m22)`.
- The fourth example displays the name, types and lengths of all vectors from plot `tran2`, and vector `v(4)` from plot `tran3`.

disto

Syntax

```
disto <.DISTO statement arguments >
```

Performs a distortion analysis. SmartSpice uses the option values specified in the current input deck.

echo

Syntax

```
echo <arg1 arg2 ...>
```

The echo commands writes its arguments to the standard output. echo is useful for producing diagnostics in SmartSpice scripts, and for displaying the contents of variables.

edit

Syntax

```
edit <filename>
```

If no filename is specified, this command edits the current input deck file, allowing you to modify it and load it back in. If a filename is specified, this command will edit that file and load it, making that circuit the current one.

Examples

```
edit
edit schmitt.in
```

The first example edits the current input deck. The second example edits the file schmitt.in.

email

Syntax

```
email <file_name> <email_address> <subject >
```

This command emails the file file_name to the email address email_address with a subject subject. If subject is not specified, the SmartSpice version information is used, (for example, SmartSpice 4.6.5). If email_address is not specified, the file will be sent to the address stored in the variable email_address.

This command is used to report bugs or suggestions.

Examples

```
email ex03.in
email bug1.in john “New Suggestions”
```

The first example emails ex03.in to the email address stored in the variable email_address. The second example emails the file bug1.in to user john within the same company, with the subject: New Suggestions.

fft

Syntax

```
fft <all | ov1 ...>
```

This command performs a “Fast Fourier Transform” (FFT) on specified output variables: ov1, ... from the current plot. For a complete description of output variables and expressions, see Chapter 6 Outputs.

If the argument is all, or if it is omitted, all vectors from the current plot will be transformed.

The results of the FFT are stored as vectors with the same names as the original ones, but created in a new plot.

If the scale of the current plot is not frequency, a Forward Fourier Transform is performed, and the results are stored in a new ac type plot, with the scale frequency. The title of the new plot is created by appending the string “(forward fft)” to the title of the original one.

If the scale of the current plot is `frequency`, a reverse Fourier Transform is performed, and the results are stored in a new `tran` type plot, with the scale `time`. The title of the new plot is created by appending the string "`(reverse fft)`" to the title of the original one.

Examples

```
fft
fft all
fft v(1) v(2) i(vin)
fft mag(v(1)*v(2))
```

- The first and second examples get the FFT of all vectors in the current plot.
- The third example will transform only the specified vectors `v(1)`, `v(2)` and `i(vin)`.
- The fourth example transforms the magnitude of the product of the vectors `v(1)` and `v(2)`. The FFT result will be a vector named `mag(v(1)*v(2))`. To print or plot this vector, the vector name must be enclosed in double quotes as follows:

```
plot "mag(v(1)*v(2))"
```

fourier

Syntax

```
fourier <.FOUR statement arguments>
```

Specifies Fourier analysis. SmartSpice will use the option values specified in the current input deck.

hardcopy

Syntax

```
hardcopy <filename <plot arguments>>
```

This command writes one or more vectors or expressions to the hardcopy file; a file that can be printed on one of the following hardcopy devices: postscript printer, laserjet or pen-plotter. The type of hardcopy file can be selected using the variable `hcopydevtype` (the default is postscript).

The `hardcopy` command has a syntax identical to the `plot` command, except for a file name in which to put the `plot` image. If no filename is given, a temporary hardcopy file will be created. If no `plot_arguments` are given, all vectors from the current plot will be written.

Examples

```
hardcopy
hardcopy tranplot.ps
hardcopy /tmp/file1.ps v(1) v(2)*ic(q3)
hardcopy frame1 v(4) v(5) xlimit 1 2.5 ylimit 0 2
```

- In the first example, hardcopy writes all vectors from the current plot to the temporary hardcopy file (with a unique name generated by the program).
- The second example writes all vectors from the current plot to the hardcopy file `tranplot.ps`.
- The third example writes the vector `v(1)` and the expression *v(2)*ic(q3)* to the hardcopy file `/tmp/file1.ps`.
- The fourth example writes vector `v(5)` to the hardcopy file `frame1`, applying the specified `xlimit` and `ylimit` arguments.

help

Syntax

```
help <topic>
```

This command invokes the help system. To ensure that the help system is always up to date, and that it is coherent on all platforms, SmartSpice uses Acrobat Reader and the Portable Document Format (PDF) to provide online help.

history

Syntax

```
history <-r> <number>
```

This command displays the last `number` commands typed in SmartSpice, or all the commands saved if there are no arguments. The number of commands saved is determined by the value of the `history` variable. If the `-r` flag is given, the list is printed in reverse order.

Examples

```
history
history -r 20
```

The first example displays all saved commands. The second example prints, in reverse order, the 20 most recent commands.

iget

Syntax

```
iget vecname [vecname .....]
```

The `iget` command has been added so that after using `iload` to load the information about the vectors in a large plot (without actually loading the vectors themselves), you can select the vectors to be loaded and load all of the required vectors in one pass.

Examples

```
iget v1 v3
```

The specified vectors must be in the currently selected plot. The order of the vectors is not important.

iload

Syntax

```
iload
```

For rawfiles larger than 100 Mbytes, loading all the vectors generated in a simulation could take some time, and in the case of very large rawfiles, it could result in the available memory being exhausted. Normally, you do not need to view or use all of the vectors stored in the rawfile.

This command allows you to load only those vectors that are required. It can only be applied to SmartSpice or Hspice binary format rawfiles (`binary` and `binary_float`). The `iload` command is analogous to the original load command, except it loads the specified rawfile incrementally. In this case, only the header information describing the type of plot, number of

vectors, names of vectors, etc. is extracted. The actual vectors are not read, and memory is not allocated for them.

Vector data will only be read from the rawfile when a vector is used. This will occur when a vector is referenced by a command, as an argument to a plot command, or in an expression. SmartSpice will automatically detect that the vector has not been loaded from the rawfile yet, and load the data, before proceeding with the required operations.

Alternatively, after previewing the names and lengths of the vectors and the type of plot, you may use the `iget` command to load all desired vectors in one pass.

It is possible to load a rawfile incrementally from the **Load Rawfile** dialog. By default, a full load will be performed. However, the **Incremental Load** toggle can be selected before the **Load** button is pressed.

Examples

```
iload
```

iplot

Syntax

```
iplot < all | vecname1 ...>
```

This command identifies vectors from the current circuit for incremental plotting; their values will be plotted as the simulation runs. If the `all` argument is given, all the vectors from the current circuit will be incrementally plotted. If used without arguments, this command lists vectors that will be incrementally plotted.

Examples

```
iplot
iplot all
iplot v(1) v(2) i(load)
```

- In the first example, the `iplot` command, used without arguments, lists all vectors from the current circuit that will be incrementally plotted.
- In the second example, all the vectors from the current circuit are specified for incremental plotting.
- In the third example, only vectors `v(1)`, `v(2)` and `i(load)` are specified for incremental plotting.

iprint

Syntax

```
iprint <all | vecname1 ...>
```

This command specifies vectors from the current circuit for incremental printing; their values will be printed as the simulation runs. If the `all` argument is given, all the vectors from the current circuit will be incrementally printed. If used without arguments, this command lists vectors that will be incrementally printed.

Examples

```
iprint
iprint all
iprint v(1) i(c2) i(vin)
```

- In the first example, the `iprint` command, used without arguments, lists all vectors from the current circuit that will be incrementally printed.
- In the second example, all the vectors from the current circuit are specified for incremental printing.
- In the third example, only vectors `v(1)`, `i(c2)` and `i(vin)` from the current circuit are printed during simulation.

let

Syntax

```
let <vecname=expr >
```

This command creates a vector with the name `vecname`, and the value given by the expression `expr`. None of the vector options (such as default, scale, color, etc.) that are read from the rawfile are preserved when a vector is created using this command. If there are no arguments, the `let` command performs the same function as the display command without arguments and will list all vectors from the current plot.

Examples

```
let
let new1=v(2)+2*v(4)
let new2=i(R2)*v(6,7)/2
```

In the first example, the `let` command, used without arguments, displays the names, types and lengths of all vectors from the current plot. The second and third examples create new vectors: `new1` and `new2`, respectively.

linearize

Syntax

```
linearize <vecname1 ...>
```

The purpose of this command is to force the results of a Transient analysis to conform to a linear scale. Due to the algorithm that determines the time steps used, the time scale can not be linear. If no argument is given, all the vectors in the current plot are copied to a new plot, which becomes the current plot, and their data is interpolated onto a linear time scale. If `vecname1,...` are given, then only those vectors are copied. The variable *polydegree* determines how the interpolation is performed. The current plot must correspond to the current circuit (otherwise the wrong values for the start, step, and stop times can be used).

The variables `linearize_tstart` and `linearize_tstep` may be used to control the linearization, when it is performed as a post-processing step after simulation.

Examples

```
linearize
linearize v(1) v(2) ic(ql)
set linearize_tstart=10e-9
set linearize_tstep=1e-9
linearize v(out) v(in)
```

- In the first example, the `linearize` command, used without arguments, causes all vectors from the current plot to be linearized.
- In the second example, only the vectors `v(1)`, `v(2)` and `ic(ql)` from the current plot will be linearized.

- In the third example, the two vectors `v(out)` and `v(in)` will be linearized starting from `10ns` to the endtime of the simulation, with a step size of `1ns`. A new plot is created to hold the new linearized vectors and the corresponding linearized time scale. In all three examples, the current plot must be transient.

listing

Syntax

```
listing <listing_type > device_type
```

where `device_type` is `r,c,m,...`

Shows a listing of the current circuit. The argument `listing_type` controls the format of the listing. The possible values for `listing_type` are:

`logical`	Shows the listing with the comments removed and continuation lines appended to the end of the previous line.
`physical`	Shows the listing with the comments and continuation lines preserved.
`deck`	Similar to the physical listing, except that the line numbers are omitted. This listing type shows the input file verbatim.
`expanded`	This listing type shows the circuit listing after subcircuit expansion. It also shows error messages associated with particular lines.
`param`	Similar to the expanded listing type, with the addition of parameter names and their corresponding values shown after the lines in which these parameters are used.

If `listing_type` is omitted, the listing command shows a logical listing.

Examples

```
listing
listing physical
listing expanded
listing param
listing deck
listing expanded  r  (print out only resistors)
```

In the first example, the listing command without arguments shows the logical listing of the current circuit. The second, third, fourth and fifth examples show the `physical`, `expanded`, `param` and `deck` listings of the current circuit.

listingtofile

Syntax

```
listingtofile filename
```

Saves whole listing of the current circuit into one file.

Example

```
Listingtofile file.dat
```

load

Syntax

```
load <rawfilename1 ... >   < from=start  to=stop>
```

from, to	The mandatory keywords.
start, stop	The starting and ending scale vector values define the interval where plot vector values will be loaded.

Loads the data from the rawfiles `rawfilename1,...`. The default rawfile name is `SmartSpice.raw`, or the argument to the `-r` startup option, if one was given. The last plot in the rawfile becomes the current plot.

Examples

```
load
load diffpair.raw
load rtlinv.raw schmitt.raw rc.raw
load file.raw from=10ns to=300ns
```

In the first example, the `load` command used without arguments, causes the rawfile `SmartSpice.raw`, or the argument to the `-r` startup option, to be loaded.

In the second example, the rawfile `diffpair.raw` will be loaded.

In the third example, three rawfiles `rtlinv.raw`, `schmitt.raw` and `rc.raw` will be loaded.

In the fourth example, SmartSpice loads the plot from the rawfile `file.raw` on time interval from `10ns` to `300ns`. In case when the raw file consists more than one plot the specified interval will have effect for all plots. The `start` and `stop` values have to be specified both.

makepwl

Syntax

```
makepwl <filename> <y vector> <x vector>
```

This command opens `filename` (overwriting it if it exists already) and creates piecewise linear data pairs using the `y vector` and `x vector` data. Both vectors must be available within the SmartSpice environment, but need not be the same length. The piecewise linear data is created by pairing up the `y` data with the corresponding `x` data. Normally, the `x` data vector will be the scale to which the `y` data vector is plotted (i.e., its values will increase monotonically). The resulting file can then be utilized by the `pwlfile` option of the dependent source components of a circuit.

Example

```
makepwl ../pwldata/v1pwldata v(1) time
```

This example creates or overwrites the file `v1pwldata` in the subdirectory `pwldata`, and pairs up the values of the vector `v(1)` and the vector `time`. The command will terminate when the end of either vector is reached.

measure

Syntax

```
measure <.MEASURE statement arguments>
```

Performs measurement of circuit characteristics. Use this to reduce output and CPU time.

mt_notsafe_com

Syntax

```
mt_notsafe_com
```

Displays a hint of which commands are multi-thread-safe.

net

Syntax

```
net <.NET statement arguments>
```

Performs small-signal network analysis. SmartSpice will use the option values specified in the current input deck.

nodeconn

Syntax

```
nodeconn
```

This command prints the list of nodes, number of connections to the node, and the type of node.

```
number of equations : 8
NODE NUMBER CONNECTION TABLE

node name   |equation number    |connection number    |node type
0            eq.num=0            N=117                type=NODE_VOLTAGE
vdd          eq.num=1            N=89                 type=NODE_VOLTAGE
vdd1         eq.num=2            N=28                 type=NODE_VOLTAGE
vdd2         eq.num=3            N=9                  type=NODE_VOLTAGE
11           eq.num=4            N=3                  type=NODE_VOLTAGE
12           eq.num=5            N=57                 type=NODE_VOLTAGE
13           eq.num=6            N=7                  type=NODE_VOLTAGE
7            eq.num=7            N=4                  type=NODE_VOLTAGE
```

nodelist

Syntax

```
nodelist
```

This command lists a summary of the connections (node table) of the current circuit. It is an interactive equivalent of the `NODE` option (see Section 3.14“.OPTIONS (Option Specification)”).

noise

Syntax

```
noise <.NOISE statement arguments>
```

Performs a noise analysis. SmartSpice will use the option values specified in the current input deck.

op

Syntax

```
op <savev> <callv>
```

Performs an operating point analysis. SmartSpice will use the option values specified in the current input deck.

pause

Syntax

```
pause
```

Provides a pause in the analysis.

plot

Syntax

```
plot ov1 <vs xov1> <ov2 <vs xov2>...> <plot_options>
```

This command plots `ov1, ov2...` on the screen (on a graphics terminal).

ov1, ov2, ... xov1, xov2...	Output variables or expressions. For a description of output variables and expressions, see Chapter 6 Outputs and Chapter 4 Expressions.

For each set of output variables or expressions that is followed by a `vs xov` clause, all of those vectors are plotted with the `xov` for a scale. Multiple sets of output variables or expressions with different x-scales can be given in this manner.

To reference vectors in a plot that is not the current plot, use the notation `plotname.vecname`. The `setplot` command can be used to change the current plot. Wildcards can be used in place of both `plotname` and `vecname`. Possible `plotname` wildcards are:

```
all, alltran, alldc, allac, allop, allmeas
```

`all` is the only acceptable `vecname` wildcard.

Examples

```
plot all.(vcc)
plot alldc.v(out)
plot allmeas.trise
plot dcl.all
```

The first command plots vcc from all available plots. The second command plots v(out) from all DC analysis plots. The next command plots trise from all measurement plots. The last command plots all vectors from the dcl plot.

A vector name beginning with the @ symbol is considered a reference to an internal device, to a model parameter, or to a circuit parameter. If the vector name is the form @name[param], it denotes the parameter param of the device, or model name for the current circuit.

A vector can be either the name of a vector already defined, a floating-point number (a scalar), or a list such as [elt1 elt2 ... eltn], which is a vector of length n.

The notation expr [lower upper] where lower and upper are numbers, denotes the range of elements from expr between lower and upper.

The notation expr [num] denotes the numth element of expr.

If upper is less than lower, the order of the elements in the vector is reversed. In all other cases, the brackets ([and]) serve to surround literal vectors as described above.

plot_options	One or more of the plot options in the following list. Note that options must appear after output variables and expressions

The plot options nogrid, noxgrid and noygrid define the grid type. Only one grid type can be used. If no grid type is specified, the program checks the value of the string variable gridtype. The default is to plot both x and y grids.

nogrid	Does not plot a grid.
noxgrid	Does not plot an x grid.
noygrid	Does not plot a y grid.

The plot options linplot, combplot and pointplot define the plot style. Only one plot style can be used. If no plot style is specified, the program checks the value of the string variable plotstyle

.

linplot	Creates a plot by connecting data points with a linear line. This is the default plot style.
combplot	Uses a comb plot instead of connected points. Each point is connected to the bottom of the screen by a line.
pointplot	Plots data points as unconnected points. Each successive output variable is plotted with a different character to mark the points. The characters used can be changed by setting the variable pointchars.

The plot options linlin, loglin, linlog, loglog, polar, and smith define the scale type. Only one scale type can be used. If no scale type is specified, the program checks the value of the string variable scaletype.

linlin	Uses a linear scale for both axes. This is the default scale type.
linlog	Uses a linear scale for the x axis, and a log scale for the y axis.
loglin	Uses a log scale for the x axis, and a linear scale for the y axis.
loglog	Uses a log scale for both axes.
polar	Uses a polar grid.
smith	Uses a smith grid.
nointerp	This option is used for the asciiplot option. The nointerp option suppresses the interpolation normally used to force the data onto a linear scale.
samep	If samep is given, all the options used in the previous plot, hardcopy or asciiplot commands are used for the current command, unless they are overridden on the current command.
title string	The string will be used as the title printed near the top of the graph.
xcompress value	Specifies that only one out of every value points should be plotted.
xdelta value	This option is used for asciiplot. Specifies value as the spacing between grid lines on the x-axis.
xindices lower upper	Only data points with indices between lower and upper are plotted. upper must be greater than or equal to lower.
xlabel string	Use string as the label for the x-axis, instead of the name of the scale.
xlimit lower upper	The plot area in the x-direction is restricted to positions between lower and upper.
ydelta value	Specifies value as the spacing between the grid lines on the y-axis.
ylabel string	Use string as the label for the y-axis. By default, none are printed.
ylimit lower upper	Limits the plot area in the y-direction to positions between lower and upper.
discontinuous	Used when the x-axis data is not monotonic. The discontinuous option will hide the flyback lines when the direction of the x-axis value changes.

Options xlimit, ylimit, xcompress, xdelta and ydelta can be abbreviated to xl, yl, xcomp, xdel and ydel, respectively.

Examples

```
plot v(2) ig(m3)
plot v(4) (v(6)+v(8)) vs time
plot v(3,4)*i(R34) loglin xlimit 0 5 ylimit 0 2
plot v(1) ic(ql) xcompress 5
plot v(1) i(vin) xindices 20 30
plot v(1) vs v(3)
plot v(1) v(2) yl 1 2 xl -5 12.2
```

- The first example plots `v(2)` and the gate current `ig` of the device `m3`.
- The next example plots `v(4)` and the expression *(v(6)+v(8))* with time on the `x`-axis.
- In the third example, the expression *v(3,4)*i(R34)* is plotted using a log `x` axis and a linear `y` axis. The `x` axis minimum and maximum are `0` and `5` respectively, while those of the `y` axis are `0` and `2` respectively.
- In the fourth example, every 5th point on each of the vectors `v(1)` and `ic(q1)` is plotted.
- In the fifth example, the values between the 20th and 30th data point on `v(1)` and `i(vin)` are plotted.
- The sixth example, plots vector `v(1)` versus vector `v(3)`. `v(3)` is used as the `x`-axis.
- In the final example, the vectors `v(1)` and `v(2)` are plotted with `y`-limits `1` and `2`, and `x`-limits `-5` and `12.2`.

print

Syntax

```
print <col|line> ov1 <ov2 ...>
```

This command prints values of the output variables or expressions `ov1, ov2...`. For a description of output variables and expressions, see Chapter 6 Outputs.

If the `col` argument is present, the expressions are printed side by side. If `line` is given, the expressions are printed horizontally. If all the expressions have a length of one, `line` is the default; otherwise, `col` is the default. The variables `width`, `height` and `nobreak` determine the width and height of the printout, and whether there are page breaks, respectively. If the expression is `all`, all vectors from the current plot are printed. The scale vector (time, frequency) will be in the first column, unless the boolean variable `noprintscale` is set.

Examples

```
print v(1) i(l2)
print col all
print line (v(4)+v(6))/2
```

- The first example prints vectors `v(1)` and `i(l2)`.
- The second example prints, in columns, all vectors from the current plot.
- The third example prints, horizontally, the expression *(v(4)+v(6))/2*.

printpar

Syntax

```
printpar
+ <rlclinear <resmin=value> <indmin=value> <capmin=value>
+ | rlcnonlinear> <device_name | model_mame>
```

It prints device and model parameters.

`rlclinear`	Print only linear resistors/inductors/capacitors
`resmin`	Print only linear resistors with resistance value greater or equal than resmin value
`indmin`	Print only linear inductors with inductance value greater or equal than indmin value
`capmin`	Print only linear capacitors with capacitance value greater or equal than capmin value
`rlcnonlinear`	Print only non-linear resistors/inductors/capacitors
`device_name`	Device name
`model_name`	Model name

Example

```
printpar
printpar m.x1.m1
printpar m*
printpar nch.14
printpar rlclinear res_model
printpar rlcnonlinear c*
```

The first example prints all device and model parameters of the circuit.

The second example prints model and instance parameters of `m.x1.m1`.

The third example prints all MOSFET device and model parameters.

The fourth example prints the model `nch.14` parameters. Also, SmartSpice outputs parameters of all devices which have this model.

The fifth example prints the model `res_model` paramters. Also, SmartSpice outputs parameters of all linear resistors, which have this model.

The sixth example prints model and instance parameters of all non-linear capacitors.

proc2mod

Syntax

```
proc2mod procfilename modfilename procname
```

Converts a process file `procfilename` into a set of BSIM `.MODEL` statements, and writes them in the file `modfilename`. `procname` is the process name, and is used as a prefix for BSIM model names.

Examples

```
proc2mod process.pro process.mod BSIM
```

This example converts the process file `process.pro` into the model file `process.mod`, with a BSIM prefix for all generated models.

pz

Syntax

```
pz <.PZ statement arguments>
```

Performs pole-zero analysis. SmartSpice will use the option values specified in the current input deck.

quit

Syntax

```
quit
```

Exits the program. If there are circuits that are in the middle of a simulation, or plots that have not been saved in a file, a warning asks you to confirm before quitting. The variable `noaskquit` disables this confirmation request.

Examples

```
quit
```

rawconvert

Syntax

```
rawconvert fromfile tofile newformat <fsdb_version>
<fsdb_vprbtol=value> <fsdb_iprbtol=value>
```

Reads existing rawfile `fromfile`, and writes the new rawfile `tofile` in the format `newformat`. The new format can be:

- `ascii`
- `binary`
- `binary_float` (`float` means single precision)
- `csdf`
- `excel`
- `fsdb` (SpringSoft FSDB binary format)
- `hspice_binary`
- `hspice_ascii`
- `ssf` (Silvaco Standard Structure File)
- `data` (Silvaco Data Format)
- `tabular`
- `isdb` (SimWave ISDB Format)
- `psf` (Cadence PSF `ascii` Format)
- `psf_binary` (Cadence PSF `binary` Format)
- `wsf` (Cadence WSF (`ascii`) Format)
- `wsf_binary` (Cadence WSF (`binary`)Format)

The `fsdb_version` is an optional parameter and is used only with `fsdb` format.

This parameter specifies version of SpringSoft FSDB library to be used for FSDB file generation.

Possible values are:

```
1 (FSDB lib v.4.1)
2 (FSDB lib v.4.2) (default)
3 (FSDB lib v.4.3)
5 (FSDB lib v.5.0)
```

The fsdb_vprbtol and fsdb_iprbtol are optional resolution parameters. These parameters allow saving a new FSDB file with specified voltage and current resolutions.

Default voltage resolution is fsdb_vprbtol=1E-9, and default current resolution is fsdb_iprbtol=1E-12.

Example

```
rawconvert diffp.raw diffp.raw1 excel
```

This example reads the rawfile diffp.raw, and writes it to rawfile diffp.raw1 in excel format.

```
rawconvert diffp.raw diffp.fsdb fsdb 3
```

This example reads the rawfile diffp.raw, and writes it to binary compressed rawfile diffp.fsdb in FSDB format ver.4.3.

```
rawconvert diffp.fsdb diffp.fsdb fsdb 3 fsdb_vprbtol=1.2e-6
fsdb_iprbtol=2.3e-9
```

This example reads fsdb-file diffp.fsdb and writes a new fsdb-file diffp.fsdb in FSDB format ver.4.3 with new voltage resolution fsdb_vprbtol=1.2e-6 and current resolution fsdb_iprbtol=2.3e-9.

rehash

Syntax

```
rehash
```

Recalculates the internal hash tables used when looking up UNIX commands, and makes all UNIX commands in the user path available for command completion. This command is effective only if the unixcom variable is set. This command is not available on Windows versions of SmartSpice.

Example

```
rehash
```

run

Syntax

```
run <rawfile | batchprint | alters>
```

Runs the current circuit simulation using the .AC, .OP, .TRAN or .DC statements in the input deck. If rawfile is specified, the output is saved in this file. Otherwise the output becomes the current plot. If the batchprint keyword is specified, the program will also execute all output control statements from the input deck (.PRINT, .PLOT, .MEASURE, .FOUR).

If the current input deck is part of a .ALTER sequence, then all decks that are part of this sequence will be run automatically. If the run option from the Analysis menu is used, and the selected deck is part of a .ALTER sequence, then a dialog will be displayed allowing you to run the selected deck, or all decks, in the sequence.

Examples

```
run
run diffpair.raw
```

The first example simulates the current circuit, executing simulation statements from its input deck. Simulation results are stored in memory and can be accessed immediately after the simulation.

The second example simulates the current circuit, and stores the simulation results in the rawfile `diffpair.raw` without keeping the results in memory. To access simulation results, they must be loaded from the file `diffpair.raw`.

rusage

Syntax

```
rusage <all | restype1...>
```

Prints various resource usage statistics. If `all` is specified as an argument, all the resources are printed. If no arguments are given, only total time and space usage are printed.

If `all` is not given as an argument, information for each specified resource type `restype1...` is printed. The recognized resource types are:

accept	The number of accepted time points.
elapsed	The amount of elapsed time since the last `rusage` elapsed command.
faults	The number of page faults caused by the program so far (BSD UNIX only).
loadtime	The amount of time spent loading the matrix.
lutime	The amount of time spent doing transient L-U decomposition.
rejected	The number of rejected time points.
solvetime	The amount of time spent solving the matrix.
space	The amount of data space currently used by the program.
time	The CPU time used so far.
totaltime	The total amount of time spent by the program since it was started.
totiter	The total number of iterations.
traniter	The number of transient iterations.
tranpoints	The number of transient time points.
transolvetime	The transient solve time.
trantime	The total transient time so far.

Examples

```
rusage
rusage all
rusage elapsed space time
```

- In the first example, the `rusage` command (without arguments) prints the total time and space used by the program.
- The second example prints all resources.
- The third example prints elapsed time, space and CPU time.

save

Syntax

```
save <all | vecname1 ...>
```

This command specifies the vectors `vecname1,...` from the current circuit that is to be saved during the simulation. Only these vectors will be available for post-processing. If the `all` argument is given, all vectors from the current circuit will be saved. If used without arguments, this command lists all vectors from the current circuit that will be saved during simulation.

Examples

```
save
save all
save i(R2) ig(m3) v(1) v(3)
```

- In the first example, the `save` command (without arguments) lists all vectors from the current circuit that will be saved during simulation.
- The second example causes all vectors of the current circuit to be saved.
- The third example identifies four vectors to be saved: `i(R2)`, `ig(m3)`, `v(1)` and `v(3)`.

set

Syntax

```
set <varname <=value> ...>
```

The `set` command allows you to examine and to set variables.

To examine variables, use the `set` command without arguments. The names and values of all the variables will be printed.

Variables prefixed with an “*” are associated with the current plot, and will change when the current plot changes.

Variables prefixed by a “+” are associated with the current circuit. If a variable is set that has the same name as the one associated with the current plot or circuit, it takes precedence for printing with the `set` command and $ expansion, but it will not change the value used by the circuit.

Some variables (e.g., `plots`) are read-only and can not be changed.

When used to set a variable, the `set` command requires a name `varname` and an optional `value`. A `value` can be a number, a string, or a list of values. If `value` is omitted, the variable is set to a boolean value `true`. The value of `varname` can be inserted into a command using the `$varname` notation. If a variable is set to a list of values that are enclosed in parentheses

(which must be separated from their values by blank spaces), the value of the variable is the list. The variables that have meaning to SmartSpice are listed in Section 5.5“Variables”.

Examples

```
set
set nobreak
set list_all_parameters
set polydegree=2
set gridstyle=loglog
set width=80 height=14
set myplotlist = (ac1 ac3 ac4)
```

- In the first example, the `set` command is used without arguments and causes the names and values of all variables to be printed.
- In the second and third examples, the variables `nobreak` and `list_all_parameters` are set to the boolean value `true`.
- The fourth example sets the variable `polydegree` to the numerical value `2`.
- The fifth example sets the string variable `gridstyle` to `loglog`.
- The sixth example sets the numerical variables `width` and height to `80` and `14`, respectively.

setcirc

Syntax

```
setcirc <circuitname | circuitnumber>
```

This command sets the current circuit to the circuit with the name `circuitname`, or to the number `circuitnumber`. If used without an argument, `setcirc` prints a menu of the currently loaded circuits, and prompts you to select a current circuit.

Examples

```
setcirc
setcirc 2
setcirc diffpair
```

- In the first example, the `setcirc` command used without arguments causes the menu of all loaded circuits to be printed. You are prompted to select a circuit to be the current circuit.
- The second example causes circuit number `2` to become the current circuit.
- The third example causes the circuit `diffpair` to become the current circuit.

setplot

Syntax

```
setplot <plotname>
```

This command sets the current plot to the plot `plotname`. If no `plotname` is given, this command prints a menu of the currently available plots and prompts you to select the current plot. Note that here the word “plot” refers to a group of vectors that are the result of one simulation run.

Examples

```
setplot
```

```
setplot dc1
setplot tran2
```

- The first example causes the menu of all plots to be printed; you are then prompted to select the current plot.
- The second example causes the plot `dc1` to become current.
- The third example causes the plot `tran2` to become current.

setscale

Syntax

```
setscale <vecname>
```

If used without an argument, this command prints the scale used for the current plot. If `vecname` is specified, and `vecname` is an existing `vector` from the current plot, the `setscale` command sets the scale of the current plot to the vector `vecname`.

Examples

```
setscale
setscale v(1)
```

The first example causes the scale of the current plot to be shown. The second example sets the scale of the current plot to `v(1)`.

settype

Syntax

```
settype vectype < all | vecname1 ...>
```

This command changes the type of vectors `vecname1,...` to `vectype`. If the argument `all` is given, all of the vectors from the current plot will be changed to the type `vectype`. The currently defined vector types can be listed using the `deftype v` command.

Examples

```
settype current all
settype voltage v12 tran2.v12
```

The first example sets all vectors from the current plot to the `current` type. The second example sets two vectors `v12` and `tran2.v12` to the `voltage` type.

shell

Syntax

```
shell <arguments>
```

This command executes the shell specified by the `shell` variable, if this variable is set. It executes the shell specified by the environment variable `SHELL`, if it is set; otherwise, it executes `csh`. If used with `arguments`, the `shell` command executes the `arguments` as a command to the shell.

Examples

```
shell
shell ls *.in
shell pwd
```

show

Syntax

```
show <-t | -l | -ll> <<devname1 | modelname1> ..: <paramname1 ...
>>
```

This command prints the named device or model parameters of the current circuit. Either the device/model name list or the parameter name list can be `all`, and the device/model names can contain the *,?, and [ ] wildcard characters.

The options `-t`, `-l` and `-ll` specify the format of the output:

- `-t` specifies tabular (condensed) format.
- `-l` specifies the list format (one parameter per line).
- `-ll` specifies the long list format (one parameter per line, including parameter descriptions).

The default format is list.

Examples

```
show -t all
show -ll
show ql
show qnl : bf
```

- The first and second examples print all parameters of all devices and models in the current circuit in tabular and long list format, respectively.
- The third example prints all parameters of the device `ql`.
- The fourth example prints only the parameter `bf` of the model `qnl`.

skip

Syntax

```
skip
```

Skips iteration in the simulation loop.

source

Syntax

```
source filename1 <filename2...>
```

This command reads input files `filename1, filename2,...`. The input file contains a circuit description (netlist, control statements, and models). It can also contain interactive SmartSpice commands, which must be surrounded by the `.CONTROL` and `.ENDC` statements, or prefixed by *# in order to be recognized as interactive commands. These commands are executed immediately after the input file is loaded. The input file, therefore, can have one of three forms:

- Ordinary input deck, containing only a circuit description, without interactive commands.
- Ordinary input deck, plus some interactive commands.
- SmartSpice command script that contains only interactive commands.

The first line of the input file is always considered a title line, and is used as a circuit name or command script name.

Examples

```
source diffpair.in
source deck1.in deck2.in deck3.in
source comfile1
```

- The first example loads one input file `diffpair.in`.
- The second example loads three input files: `deck1.in`, `deck2.in` and `deck3.in`.
- The third example loads and executes the command file `comfile1`.

state

Syntax

```
state
```

Prints the name of the current circuit, its status (simulation in progress or not), the type of simulation, and the number of simulation points.

Examples

```
state
```

step

Syntax

```
step <number>
```

Single-steps the simulation, or lets it run for number steps if `number` is given.

Examples

```
step
step 20
```

stop

Syntax

```
stop <after num> <when val1 cond val2> . . .
```

This command sets a simulation breakpoint. If an `after` clause is given, the simulation will stop after `num` points. If a `when` clause is given, at each point, the `val1 cond val2` condition will be checked, and if it is true, the simulation will stop. If more than one `when` or `after` clause is put on the line, the conjunction of the conditions is checked. The values `val1` and `val2` can be either constants or node names.

The condition `conds` is one of the following:

- `eq` or = equal to)
- `ne` or <> (not equal to)
- `gt` or > (greater than)
- `lt` or < (less than)
- `ge` or >= (greater than or equal to)
- `le` or <= (less than or equal to)

Note: This command, "<" and ">" does not denote an IO redirection. IO redirection is disabled for the `stop` command.

Examples

```
stop after 20
stop when i(Rs1)<=10e-3
stop after 10 when v(3)>4
```

- The first example sets the breakpoint after `20` simulations.
- The second example sets a breakpoint when `i(Rs1)` becomes less than or equal to `10u`.
- The third example causes the simulation to be interrupted when voltage `v(3)` becomes greater than `4` but not in the first `10` simulation points.

tf

Syntax

```
tf <.TF statement arguments>
```

Performs transfer function calculations. SmartSpice will use the option values specified in the current input deck.

tran

Syntax

```
tran <.TRAN statement args>
```

Performs a transient analysis. SmartSpice will use the option values specified in the current input deck.

unalias

Syntax

```
unalias alias1 <alias2...>
```

This command removes aliases `alias1, alias2, ...`. The argument can be "*", in which case all aliases are removed.

Examples

```
unalias *
unalias begin
unalias ll lpr ff
```

- The first example removes all active aliases.
- The second example removes the alias `begin`.
- The third example removes aliases `ll`, `lpr` and `ff`.

undefine

Syntax

```
undefine function1 <function2 ... >
```

Removes the definitions for the user-defined macros `function1, function2,...`. If the argument is “*”, then all macro definitions are removed.

Examples

```
undefine *
undefine min max
```

The first example removes all active macros. The second example removes two macros: `min` and `max`.

uniplot

Syntax

```
uniplot all | vecname1 ...
```

This command removes vectors `vecname1,...` of the current circuit from the list for incremental plotting. If the `all` argument is given, none of the current circuit vectors will be incrementally plotted.

Examples

```
uniplot all
uniplot v(1) v(2) i(cload)
```

The first example removes all vectors in the current circuit from the list for incremental plotting. The second example removes from this list only the vectors `v(1)`, `v(2)` and `i(cload)`.

uniprint

Syntax

```
uniprint all | vecname1 ...
```

This command removes vectors `vecname1,...` of the current circuit from the list for incremental printing. If the `all` argument is given, none of the current circuit vectors will be printed.

Examples

```
uniprint all
uniprint v(2) ig(m2) i(cload)
```

The first example removes all vectors in the current circuit from the list for incremental printing. The second example removes from this list only the vectors `v(2)`, `ig(m2)` and `i(cload)`.

unlet

Syntax

```
unlet vecname1 <vecname2 ...>
```

This command removes vectors `vecname1, vecname2,...` from the current plot. If `vecname` is the default scale (e.g., time), then one of the remaining vectors will become the default scale.

Examples

```
unlet v(1)
```

```
unlet i(R2) ic(q1) id(m22)
```

The first example removes the `vector v(1)` from the current plot. The second example removes three vectors: `i(R2)`, `ic(ql)` and `id(m22)`.

unsave

Syntax

```
unsave all | vecname1 ...
```

This command removes current circuit vectors `vecname1,...` from the list of vectors that will be saved during the simulation. If the `all` argument is given, none of the current circuit vectors will be saved during the simulation.

Examples

```
unsave all
unsave v(3) i(vin) is(m21)
```

The first example causes no vectors from the current circuit to be saved during simulation. The second example removes three vectors: `v(3)`, `i(vin)` and `is(m21)` from the list of vectors that will be saved during simulation.

unset

Syntax

```
unset varname1 <varname2...>
```

This command removes variables `varname1, varname2,...` from the list of active variables. If the argument is "*", then all variables are unset.

Examples

```
unset *
unset resp i newflag oldplot
```

The first example removes all active variables. The second example removes variables `resp`, `i`, `newflag` and `oldplot`.

unstop

Syntax

```
unstop breakpointnum1 <breakpointnum2 ... >
```

Removes breakpoints with numbers `breakpointnum1, breakpointnum2,...`, which are set using the `stop` command. Breakpoint numbers can be viewed by issuing the `stop` command without arguments.

Examples

```
unstop 2 5 18
```

This example removes breakpoints with numbers `2`, `5` and `18`.

version

Syntax

```
version
```

This command prints the current version of SmartSpice, and the host name of the computer that SmartSpice is running on.

where

Syntax

```
where
```

When performing a transient or operating point analysis, the name of the last node or device to cause non-convergence is saved. The `where` command prints out this information so that the circuit can be examined and the problem can be corrected. This can be done either in the middle of a run, or after the simulator terminates the analysis. When the analysis slows down severely or hangs, the simulator should be interrupted and the `where` command should be issued. Note that only one node or device is printed; there can be problems with more than one node.

write

Syntax

```
write <rawfile <all> <ov1 ov2 ....>>
```

This command writes the output variables or expressions `ov1, ov2` to the `rawfile`. For a description of output variables and expressions, see Chapter 6 Outputs.

Vectors are grouped together by plots and written out as such. For example, if the expression list contains three vectors from one plot and two from another, then two plots will be written, one with three vectors and one with two. The scale for a vector will be automatically included in the rawfile.

The default format for the rawfile is binary, but this can be changed by setting the `rawfiletype` variable. The default rawfile is `SmartSpice.raw`, or the argument to the `-r` flag on the command line. If no output variables or expressions are specified, all vectors from the current plot will be written.

Examples

```
write
write diffpair.raw
write rawfile1.raw v(1) v(3) i(R1)
write raw1 v(1)-tran2.v(1) i(cload) ib(q1)
write raw all
write rawfile2.raw v(*) i(r?)
```

- In the first example, the `write` command without arguments writes all vectors from the current plot in the rawfile `SmartSpice.raw`.
- The second example writes all vectors from the current plot in the rawfile `diffpair.raw`.
- The third example writes vectors `v(1)`, `v(3)` and `i(R1)` in the rawfile `rawfile1.raw`.
- The fourth example writes vectors `ib(ql)`, `i(cload)` and the expression *v(1)-tran2.v(1)* in the rawfile `raw1`.
- The fifth example writes all vectors from all plots in the rawfile `raw`.
- The sixth example writes all circuit node voltages, and all currents, whose names consist of the letter R, followed by any single character, in the rawfile `rawfile2.raw`.

Other syntax is supported by the command write.

Syntax

```
write <rawfile <from=var1 to=var2> <ov1 ov2 ...>>
```

Example

```
write user.raw from=5n to=50n tran1.v(1)
write user1.raw from=meas1.m1 to=meas1.m2 tran1.v(3)
```

The first example writes to the file user1.raw a part of the vector tran1.v(1) from 5n to 50n.

In the second example, the values of from and to are specified through measure results meas1.m1 and meas1.m2.

5.2.2 Common Functionality

An underlying STOPCONT mode is supported for DC, AC, OP, PZ, DISTO, NOISE, NET, RF (SmartSpice RF only), TRAN and TF analyses. This STOPCONT mode turns on the primitive multitasking in the postprocessor, and provides the runtime stop-continue control during an analysis. If the netlist contains other types of analyses (DC, OP, AC, etc.) the syntax order of the analyses will be rearranged so that the analysis is the last one in the list. The feature works for each analysis in the SmartSpice netlist.

The stop-continue feature contains 2 subsystems: the set of runtime control commands, and the multitasking postprocessor mechanism which allows the postprocessor functionality to be applied to the analysis “on the fly”. Runtime control commands are regular SmartSpice external commands: pause, cont, cancel, mt_notsafe_com:

- The pause command provides a pause for the analysis.
- The cont continues analysis from the point where it was paused.
- The cancel aborts and quits from analysis simulation loop.

These commands allow you to control the simulation flow in the analysis. The “on the fly” mechanism permits an execution of multi thread postprocessor commands. This means that during the progress of the analysis you can execute the commands: plot, print, and so forth.

Not all commands are permissible, for example, commands which can modify the output structure (removing the vector or the plot, and running the new analysis is not allowed and is not multi thread-safe). The command mt_notsafe_com displays a hint on which commands are multi thread-safe and are permitted in the mechanism “on the fly”. The stop-continue feature is supported in the parallel mode of SmartSpice.

Also, the feature is implemented in the command mode (smartspice -c) on Linux, GUI mode and supported on each platform. This feature is intended for R&D in SmartSpice. The main advantage of the using of the stop-continue feature is that you do not need to wait for the completion of the analysis. It is possible to use a full functional postprocessor during progression of the analysis. This feature slightly aggravates (5%-10%) the performance due to the threads synchronization, and the complexity of the communication of the analysis core and the postprocessor. It is recommended to allocate one extra CPU if you work with this feature. For example, if a computer equipped by 4 CPUs and SmartSpice will operate in the parallel mode, the command line is smartspice -P 2 or -P 3. The 4th CPU will be launched for stop-continue functionality. To get the stop-continue feature working, load the netlist in SmartSpice and run the simulation. After that, open the menu **Display→Vectors** in the GUI, or use the .PLOT command to inspect the results of the simulation. Also, you can

input commands in to the command window on the Main GUI window. Also, the command window accepts the stop-continue commands mentioned above: `pause`, `cont`, `cancel`, `mt_notsafe_com`. If operating on Linux, the same sequence of actions are required in the command mode (`smartspice -c`).

The commands `pause` and `stop` are not necessary after simulation without stopping the analysis. But they are recommended if you run a lot of commands during the progress of the analysis. If you run the `write`, `load` and `measure` commands, it is recommended that you use the command `pause` before and then `cont` after the execution of the postprocessor command. The stop-continue feature supports parametric analyses: `.ALTER`, `.TEMP`, `.MODIF` and `.ST`.

The **Runtime** dialog for the analysis is equipped with 3 buttons: **Pause**, **Skip** and **Stop**. The Pause button pauses the simulation. The **Skip** button will only be displayed if parametric analysis is used (`ALTER`, `MODIF`, `ST`, `TEMP`). The **Skip** button aborts analysis simulation on the current parametric step, and proceeds to the next value of the parametric loop. The Stop button aborts current analysis, and allows to simulator to start next analysis. All others types of analyses have a **Runtime** dialog with one **Stop** button. It is permitted to have in the netlist `TRAN`, `AC` and `DC`. The analysis follows stop-continue rules, and all other behavior as in the old “modal” fashion.

To resume a simulation menu **Analysis→Run** (**Ctrl+R**), or **Analysis→Continue** must be used. Another way to resume a simulation is to use command `cont` in the command GUI window.

If you abort AC, DC, etc. using the **Stop** button in the **Runtime** dialog, SmartSpice proceeds to the next analysis. If AC, DC, etc. are not converging, SmartSpice aborts all analyses.

5.3 C-Shell Like Features

There are several features available in the SmartSpice command mode that are derived from the user interface of the C-Shell. These features are:

- Aliases
- Command Completion
- Global Substitution
- History Substitution
- I/O Redirection
- Quoting
- Multiple Commands

5.3.1 Aliases

After history expansion, if the first word on the command line has been defined as an alias, the text for which it is an alias is substituted. The alias can contain references to the arguments provided on the command line, in which case the appropriate arguments are substituted. If there are no such references, any specified arguments are appended to the end of the alias text.

In the body of the alias text, any strings of the form `!:number` are replaced with the numbered argument of the actual command line. Note that when the alias is defined with the alias command, these strings must be quoted to prevent history substitution from replacing the "`!`" characters before the alias command can use them. For example, the command:

```
alias pcomb2 plot '!:1' '!:2' combplot
```

causes `pcomb2 v(1) v(2)` to be replaced with `plot v(1) v(2) combplot`. When defining these types of aliases, do not quote the argument list substitutions in this manner.

If a command line starts with a backslash ("\"), any alias substitution is inhibited.

5.3.2 Command Completion

With BSD UNIX, Tenex-style command completion is available in command mode. If EOF (**Ctrl-D**) is typed after the first character on the line, a list of the commands or possible arguments is printed. Entering a **Ctrl-D** as the first character on a line will terminate the program. If the **Esc** key is pressed, then the program attempts to complete the word being typed based on the choices available. If there is more than one possibility, it will complete as much of the command as possible. Command completion is used for commands, most keywords, variables, vector names, file names and several other types of arguments.

5.3.3 Global Substitution

The characters "~", "{", and "}" have the same effects as they do in the C-Shell (i.e., home directory and alternative expansion).

The string `~user` at the beginning of a word is replaced by the home directory, or if the first component of the pathname is simply ~, the directory of the current user is used.

The string:

```
StringA{String1,String2,... StringN}StringB
```

is replaced by the list of words:

```
StringAString1StringB, StringAString2StringB
+ ... StringAStringNStringB.
```

Braces can be nested.

It is possible to use the wildcard characters *, ?, [, and] to match filenames, where * denotes 0 or more characters, ? denotes one character, and [ ] denotes a range of characters, but the variable `noglob` must first be unset. When `noglob` is unset, the wildcard characters cannot be used for algebraic expressions. Set the variable `noglob` again after completing wildcard expansion. Note that the pattern `[^abc]` will match all characters except `a`, `b` and `c`.

5.3.4 History Substitution

History substitutions are prefixed by the character `!`, or if located at the beginning of a line, the character `^`. The string `!!` is replaced by the previous command, the string `!prefix` is replaced by the last command with that prefix. The string `!?pattern` is replaced by the last command containing that pattern. The string `!number` is replaced by the event with that number, and `^oldpattern^newpattern` is replaced by the previous command with `newpattern` substituted for `oldpattern`.

When substitutions are performed using the `^old^new` syntax, there can be no spaces in either `old` or `new`.

A `!string` sequence can be followed by a modifier prefixed with a `:`. This modifier can select one or more words from the event. The modifier `:1` selects the first word, `:2-5` selects the second through the fifth word, `:$` selects the last word, and `:$-0` selects all of the words but reverses their order. Two other (`:`) modifiers are supported: `:p` will cause the command to be printed but not executed, and `:s^old^new` will replace the pattern `old` with the pattern `new`. The sequence `^old^new` is synonymous with `!!:s^old^new`.

All the commands typed in are saved in the history list. This list can be examined with the `history` command, and its maximum length can be changed by changing the value of the `history` variable.

5.3.5 I/O Redirection

The input or output (I/O) commands can be changed from the terminal to a file by including an I/O redirection on the command line. The possible redirections are:

`> file`	Sends the output of the command to the file.
`>> file`	Appends output of the command to the file or creates the file if it does not exist.
`>& file`	Sends both the output and the error messages to the file.
`>>& file`	Appends both the output and the error messages to the file.
`< file`	Reads input from the file.

Both an input and an output redirection can be present on a command line. More than one of each can not be present, however. I/O redirections must be at the end of the command line.

5.3.6 Quoting

Words can be quoted with the characters:

- “ (double quote)
- ’ (single quote)
- ` (back quote)

A word enclosed by any of these quotes can contain blank spaces.

A string enclosed by double quotes can include further character substitution, but it will be considered one word by the program.

A string enclosed by single quotes also has all its special characters protected. No global expansion (*, ?, etc.), variable expansion ($), or history substitution (^, !) will be recognized. Numbers are still considered numbers.

A string enclosed by backquotes is considered a command to the shell. The command is executed, and the output of the command replaces the text.

5.3.7 Multiple Commands

More than one command can be put on one line and separated by semicolons (`;`). For example, to load the circuit file `rtlinv.in`, run the simulation, and print all vectors, issue the command:

```
source rtlinv.in; run; print all
```

5.3.8 UNIX Commands

If the variable `unixcom` is set and the operating system is UNIX, commands that are not built-in are considered UNIX commands and executed as if the program were a shell.

5.3.9 Variable Substitution

The values of variables can be used in commands with the notation `$var` where the value of the variable is to appear. The special variables `$$` and `$<` refer to the process ID of the program, and to a line of input, which is read from the terminal when the variable is evaluated, respectively. If a variable has a name of the form `$$var`, then `var` is considered to be a vector, and its value is taken to be the value of the variable. If `var` is a list variable, then the expression `$var[low-high]` represents a range of elements. Either the upper or lower index can be left out, and the reverse of a list can be obtained with `$var[len-0]`. Also, the notation `$?var` has a value of `1` if the variable `var` is defined, otherwise it has a value of 0. The `$#var` notation is equal to the number of elements in `var` if the variable is a list, `1` if the variable is a number or string, and `0` if the variable is a boolean.

5.3.10 Comments

Comments are indicated by the ‘#’ symbol. It can occur anywhere on a line between the `.CONTROL` and `.ENDC` statements. All text to the right of this symbol is considered a comment and is ignored.

5.4 Command Scripts

If a word is typed as a command, and there is no built-in command with that name, the word is assumed to be a file name containing a list or script of commands. The file can reside in any directory in the search path. When the file is found, the commands in the file are executed. The syntax for a command script is:

```
Explanatory title
.control
<SmartSpice commands>
.endc
```

The body of the script can contain any SmartSpice command, including calls to other script files. Local variables can be used, and arguments passed to the script file can be recovered using the `argv` and `argc` variables. Script files use the same syntax as C-shell scripts (i.e., they have the same conditional loop and assignment statements). Supported structures include: `foreach`, `while`, `repeat`, `dowhile`, `if-then-else`, `label`, `goto`, `continue` and `break`. These features allow you to generate complex scripts to automate the simulation and analysis process. SmartSpice also fully supports standard I/O redirection operators and backquote substitution, which allows shell commands to be evaluated and the results to be assigned to internal SmartSpice variables. In addition, SmartSpice scripts can call SmartSpice commands directly.

Examples

```
Script1 Example
.control
if $argc > 0
echo $argc
   repeat $argc -1
shift
source $argv [0]
run
end
.endc
```

This example script will accept any number of input decks supplied as arguments, and `source` and `run` each of them in turn. If the file containing the script is named `script1`, then the SmartSpice command to call it is:

```
script1 test1.in test2.in
```

where `test1.in` and `test2.in` are both input decks. Control returns to SmartSpice when the script terminates.

```
Script2 Example
.control
foreach pl $plots
strcmp i “constants” $pl
if $i eq 0
continue
end
setplot $pl
measure max_v2 max v(2)
echo “Maximum value of v(2) in” $pl “is” $max_v2
end
unset pl i max_v2
.endc
```

This script will measure the maximum value of the vector `v(2)` in each plot, with the exception of the `constants` plot, and print the value out.

```
Script3 Example
.control
set i = 0
set allow_index_op=true
repeat
if $i eq length(v(2))
break
end
print v(2)[{$i}]
set i = `expr $I + 1`
end
unset i
.endc
```

Note: The quotes around this Unix expression (`` `expr $I + 1` ``) need to be followed, otherwise the evaluation will be incorrect.

This script will print out each of the elements of the `v(2)` vector. The repeat command is used to iterate over the vector, while the shell command `expr` is used to increment the count variable `i` each time.

```
Script4 Example
.control
setplot tran3
measure max_v1 max v(1)
measure max_v2 max v(2)
set numdgt = 12
echo “The maximum value of v(1) is ” $max_v1 > out.$$
echo “The maximum value of v(2) is ” $max_v2 >> out.$$
shell cat out.$$
shell rm out.$$
.endc
```

This script measures the maximum values of `v(1)` and `v(2)` in the plot `tran3`. A file is then created using the PID of the current SmartSpice process in its name, and a report containing these measurements written to it. The last two commands list the contents of the file and then remove it.

5.4.1 Enhanced Control of Embedded Commands

SmartSpice allows embedded commands to be added to a deck by preceding them with the characters '*#'. To other SPICE simulators, these look like comments, but SmartSpice interprets them as if they were enclosed within a .CONTROL block. They are executed each time the deck is sourced. Thus, the line:

```
*#version
```

when contained within an input deck, it's interpreted as:

```
.CONTROL
version
.ENDC
```

It will print version information about SmartSpice every time the deck is sourced. This is not a new feature, though it was not documented in the past.

A problem arises when this sequence ('*#') is inadvertently used at the beginning of a line.

To overcome this, two new SmartSpice variables have been added, one to turn the feature on or off, and another to change the sequence of characters that signals an embedded command.

Variable	Type	Default
embedded_commands	boolean	set (true)
embedded_command_string	string	'#'

The default behavior is unchanged: '#' is still recognized as the string that precedes an embedded command (that is, the line must begin with the characters '*#'). To turn off the feature, add the following line to the SmartSpice.ini file:

```
set embedded_commands=false
```

To leave the feature turned on, but to change the string that signals an embedded command, the following line should be added to the SmartSpice.ini file (for purposes of illustration, it is changed to 'SSPICE' here):

```
set embedded_command_string='SSPICE'
```

With this definition, any line beginning with the characters '*SSPICE' will be interpreted as an embedded command, and the remainder of the line (after the '*SSPICE' has been stripped) will be executed as if it were in a .CONTROL block each time the deck is sourced.

5.5 Variables

There are several types of variables:

- Booleans: Have boolean values (the variable is either set or unset).
- Numbers: Have numeric values.
- Strings: Have character strings as values.
- Lists: Have a list of character strings as values.

There are two ways to assign a variable:

- The `set` command.
- The `.OPTIONS` statement in the circuit file (a variable set in this manner is specific to the circuit it appears in).

Table 5-1 lists variables that have a special meaning in SmartSpice. In addition, all circuit variables described in the `.OPTIONS` statement also have special meaning in SmartSpice.

Table 5-1 Variable Table

Variable	Type	Default
`allow_index_op`	boolean	`false`
`altermode`	boolean	false
`appendwrite`	boolean	`unset (false)`
`autotranop`	int	2
`cdl_support`	boolean	unset (false)
`comment`	boolean	`unset (false)`
`compatible_dio`	string	
`compatible_bjt`	string	
`compatible_bsim3v3`	string	
`compatible_bsim4`	string	
`checkwindowwarn`	boolean	unset (false)
`cputime`	number	$+ 10^{+6}$
`createsubcircuitnodemap`	boolean	`unset (false)`
`curplot`	string	
`curplotdate`	string	
`curplotname`	string	
`curplottitle`	string	
`defrawpts`	number	
`dhe_expr`	boolean	`false`

Table 5-1 Variable Table

Variable	Type	Default
dhe_latency_tol	number	1e-3
dhe_level	boolean	false
diff_abstol	number	1e-12
diff_reltol	number	0.001
diff_vntol	number	1e-6
disklimit	number	50
display_form	number	1
dpolydegree	number	2 (in HSPICE mode, 1)
editor	string	*vi* for UNIX
email_address	string	
evaluatemeasureexprasvector	boolean	unset (false)
fftgridsize	number	1024
flattened_dcmonte	boolean	unset (false)
flattened_sweep	boolean	unset (false)
floatingnodeiserrors	boolean	unset (false)
fourgridsize	number	200
fsdb_vprbtol	number	1e-9
fsdb_iprbtol	number	1e-9
fsdb_version	number	2
gridsize	number	0
gridtype	string	xygrid
hcopydev	string	(empty string)
hcopydevtype	string	postscript
height	number	60
helppath	string	
history	number	100
hspicetopologychecker	boolean	false
ignore_dotcards	boolean	unset (false)
ignoreeof	boolean	unset (false)

Table 5-1 Variable Table

Variable	Type	Default
inlinecc	string	;$
iplot_one	boolean	unset (false)
linearize_tstart	number	0
linearize_tstep	number	0
list_all_currents	boolean	unset (false)
list_all_parameters	boolean	unset (false)
logarithm0	number	lg(): lg(DBL_MIN) ln(): ln(DBL_MIN) db(): db.lg(DBL_MIN)
long_names_in_tr0	boolean	true
lpr_format	string	"lpr -P %S%S"
max_num_rawfile_read_errors	number	unlimited
maxvolt/maxamp	number	See MAXVOLT/MAXAMP below
measout	number	See savemeasures below.
measureresultslocal	boolean	unset (false)
meas_fail_default_value	number	1.79769E+308
memorylimit	number	50
msearchauto	number	0
nfreqs	number	10
noasciiplotvalue	boolean	unset (false)
noaskquit	boolean	unset (false)
noaskstopsim	boolean	false
nobreak	boolean	unset (false)
noclobber	boolean	unset (false)
nocreatefurierplot	boolean	unset (false)
noglob	boolean	true
nomoremode	boolean	unset (false)
nonomatch	boolean	unset (false)
nopadding	boolean	unset (false)
noprintscale	boolean	unset (false)

Table 5-1 Variable Table

Variable	Type	Default
notranlinearize	boolean	unset (false)
numdgt	number	4
optionset	number	0
override_post	number	0
parametric_data_in_raw	boolean	false
pathprintmode	number	0
plothistory	boolean	unset (false)
plots (readonly)	list	
plotstyle	string	linplot
pointchars	string	oxabcdefghijklmnopqrstuvwxyz
polydegree	number	1
polysteps	number	10
print_format	number	2
printinfo	boolean	unset (false)
program	string	SmartSpice
prompt	string	'$program!-->'
qtastorelatency	number	3
rawfile	string	SmartSpice.raw
rawfileprec	number	15
rawfiletype	string	binary
readfirst	boolean	unset (false)
renumber	boolean	unset (false)
rtscreen	boolean	unset (false)
safemode	number	0
savemeasures	boolean	true
scaletype	string	linlin
search	list	the current directory
search_always_prepend	boolean	false (in hspice mode, true)
searchorder	number	0

Table 5-1 Variable Table

Variable	Type	Default
simulator	string	SmartSpice
singularsupplycheck	boolean	true
smartspice_verbose	boolean	unset (false)
splitmeasureplots	boolean	unset (false)
stoponfatalerrors	integer	0
strictnumparse	boolean	unset (false)
subcktcheck	boolean	true
subckt_delimiter		. (period)
suppressshortdevicewarnings	boolean	unset (false)
suppressfftoutput	boolean	unset (false)
syntax_form	number	0
term	string	Shell environment variable $TERM (if set on UNIX). Empty string otherwise
ticmarks	boolean	unset (false)
unixcom	boolean	unset (false)
use_display	boolean	set (true)
use_mfiles	boolean	false
use_syntax0_libs	boolean	false
usedegrees	boolean	set (true)
warn_unrecognized	boolean	true (in SmartSpice mode)
waveform_version	string	Empty string
width	number	80

5.5.1 Variable Descriptions

The following section describes each of the variables:

allow_index_op	Setting this variable to true makes it possible to access the individual elements of the vector using the index operator [ ].

altermode	SmartSpice reloads an input deck several times for processing .ALTER statements. To avoid this overhead, setting this variable to true enables a new algorithm. For more details, refer to Section 3.13“Dot Statements”. The default value of altermode is false.

Note: Only .param, and .func/.define statements are supported in the almotermode algorithm.

Note: This algorithm is ignored when parallel .ALTER is used or the variable readfirst is specified.

appendwrite	Append data to the rawfile when a write command is issued (if the rawfile already exists).
autotranop	This option controls the cascaded advanced convergence stepping algorithms. PTA is not invoked if set to 0. PTA is invoked if set to 1. If set to 2, PTA will be used in cascade to find the initial guess for the advanced convergence stepping algorithm. Default is 2.
cdl_support	Setting this variable to true forces SmartSpice to recognize some elements of Circuit Design Language dialect.
checkwindowwarn	Allows print warning messages from the check_window statement after simulation. Default is 0 (OFF).
comment	Unless this variable is specified (in batch mode), and there is at least one .ALTER statement in the input deck, then all comment lines will be ignored and all altered lines will be removed from the input deck. If this variable is specified, then all altered lines will be commented in the input deck listing with the prefix altered:.
compatible_dio	Set COMPATIBLE model parameter for all Junction Diode models (LEVEL=1, 2, 3). The choices for the compatible_dio variable are: smartspice, hspice, pspice, spectre and eldo.
compatible_bjt	Set COMPATIBLE model parameter for all Bipolar Junction Transistor models (LEVEL=1, 2). The choices for the compatible_bjt variable are: smartspice, hspice, pspice, spectre and eldo.
compatible_bsim3v3	Set COMPATIBLE model parameter for all Berkeley BSIM3v3 models (LEVEL=11, 49, 53). The choices for the compatible_bsim3v3 variable are: smartspice, hspice, pspice, spectre and eldo.

compatible_bsim4	Set COMPATIBLE model parameter for all Berkeley BSIM4 models (LEVEL=14, 54). The choices for the compatible_bsim4 variable are: smartspice, hspice, pspice, spectre and eldo.
cputime	The maximum CPU time in seconds allowed for this job. SmartSpice has no job time limitation, by default. The cputime variable is set to -1, by default.
createsub circuitnodemap	Enables subcircuit node map file creation (.subcircuitnodemap.xml file). Default is OFF for stand alone SmartSpice, and is ON for SmartSpice under Gateway/SmartView. This boolean variable needs to be set before the deck is parsed.
defrawpts	This variable sets rawpts value by default, and may be specified in the SmartSpice.ini file or in GUI mode. If the netlist contains the option RAWPTS, it will override the defrawpts value. If a set of input decks contain .OPTIONS RAWPTS=<val>, this setting will be used. For all input decks that do not contain a RAWPTS option value, the defrawpts=<val> will be used by default.
dhe_expr	This variable belongs to the family of DHE (dynamic hierarchical engine) options. DHE is intended to work with netlists containing sophisticated hierarchy structure. This variable allows significant speed up in the parser stage of the devices: A, R, C, L are defined in the underneath the subcircuit's cells, and contain expressions as values.
dhe_latency_tol	Sets the tolerance for dhe_level=1. Default is 1e-03.
dhe_level	Setting this variable to 1(or true) turns on the mechanism for exploiting latency for resistor, capacitor, inductor, and E, F, G, H devices if parameters for the mentioned devices are set using expressions. Also, forces dhe_expr ON. Every expression variable and parameter will be checked on latent status. The v(out), a and b parameters in the *R n1 n2 r='a+b*v(out)'* will be checked on latent status. If the status is "latent", then SmartSpice skips reevaluation of the expression and will reuse the previously calculated value for the expression. It is useful for digital types of simulation when excitations are specified using pwl or pulse specification. There is a significant speed up in these digital type simulations at the expense of extra memory usage due to the underlying mechanism saving the previous state of the circuit (absolute max 7-15% increase). Default is false(OFF).
dhe_verbose	In -fast mode when the variable dhe_verbose is set to 1, SmartSpice outputs device statistics in a progress dialog.
diff_abstol	The absolute current tolerance for the diff command.
diff_reltol	The relative tolerance used by the diff command.
diff_vntol	The absolute voltage tolerance used for the diff command.

disklimit	This variable specifies a minimal disk space when a simulation will be performed. Default is 50 (means 50 MB).

Note: The `disklimit` value is specified in mega bytes.

display_form	This option affects how element and model names are printed: `1` (default) - original form; `2` - element and model names are expanded with subcircuit aliases. When the operating point information is printed, and the `display_form` variable is set to 3, in addition to default functionality, SmartSpice outputs the keyword `TOP` in the line `SUBCKT` for top-level devices. By default (`display_form=1`), or in HSPICE compatible mode (`display_form=2`), the field `SUBCKT` is empty.
dpolydegree	A SmartSpice variable specifying the degree of the polynomial used for derivative calculation. Default is `2` (quadratic). In HSPICE compatibility mode the default is `1` (linear).
editor	The name of the editor to use for the `edit` command.
email_address	The address to which a file will be sent when the `email` command is issued.
evaluatemeasure exprasvector	Setting this variable to `true` makes it possible to evaluate vector expressions in `.MEASURE ...PARAM/EXPR` statements. If evaluation as scalar has failed, SmartSpice uses the last value from the vector. Default is `false`.
fftgridsize	Sets the number of points in the Fourier spectrum when the `.FFT` statement is used. If it is not a power of two, the highest power of two which is less than `fftgridsize` is used.
flattened_dcmonte	If set, SmartSpice will generate a flattened plot for DC Monte Carlo analysis. Also, the `monte_carlo` parameter vector will be added into the plot. The `flattened_dcmonte` mode can be set by either the `.OPTIONS` statement in the input deck, or by the variable in the control block. If set in both places, `.OPTIONS` will override the variable setting.

Note: SmartSpice will always generate flattened DC monte carlo plots for blank DC monte statements (e.g., `.DC monte=10`).

flattened_sweep	Setting this variable to `true` changes the format of the saved simulation results in the TRAN, AC and DC analysis with secondary sweeping parameters. Simulated data are saved in the regular rawfile as one vector (like Berkeley's original dc nested sweep). Each TRAN, AC and DC plot, and corresponding compact chunk of data in the rawfile, will contain a number of vectors which equal to the number of steps in the specified statement.
floatingnodeiserror	Setting this variable to `true` prevents running the simulation when the circuit contains floating nodes (a floating node is a node from which there is no DC path to ground). The variable works only if the variable `stoponfatalerrors` is set. Default is `false`.
fourgridsize	The size of the grid to be created when a `fourier` command is issued.
fsdb_vprbtol	This variable controls plot voltage resolution during writing data to FSDB raw binary file. Due to the fact that FSDB format stores value changes of signal, specified value serves as a tolerance below which value changes are ignored hence controlling the resolution. Reducing tolerance ends up in increased final file size and vice versa. For example: `fsdb_vprbtol=1.2e-6` It means that if absolute signal difference between previous written signal point and current one is less than 1.2uV, then current point will be ignored and skipped. Then absolute difference between same previous point and next point will be calculated and accessed in the same way. Therefore different signals can have different number of points in the plot.
fsdb_iprbtol	This variable controls plot current resolution during writing data to FSDB raw binary file. Due to the fact that FSDB format stores value changes of signal, specified value serves as a tolerance below which value changes are ignored hence controlling the resolution. Reducing tolerance ends up in increased final file size and vice versa. For example: `fsdb_iprbtol=2.3e-9` It means that if absolute signal difference between previous written signal point and current one is less than 2.3nA, then current point will be ignored and skipped. Then absolute difference between same previous point and next point will be calculated and accessed in the same way. Therefore different signals can have different number of points in the plot.

fsdb_version	This variable specifies version of SpringSoft FSDB library to be used for FSDB file generation.
	Possible values are :
	`fsdb_version=1 (FSDB lib v.4.1)`
	`fsdb_version=2 (FSDB lib v.4.2)`
	`fsdb_version=3 (FSDB lib v.4.3)`
	`fsdb_version=5 (FSDB lib v.5.0)`

gridsize	The number of equally-spaced points to use for the y-axis when plotting. Otherwise, the current scale is used (which can not have equally-spaced points). If the current scale is not strictly monotonic, then this option has no effect.
gridtype	This variable is used to determine the type of grid used by the commands: `plot`, `hardcopy` and `asciiplot` if no `gridtype` keywords are given on the command line. Possible values are: • `xygrid`: Draw both `x` and `y` grids. • `xgrid`: Draw only the `x` grid. • `ygrid`: Draw only the `y` grid. • `nogrid`: Do not draw either the `x` or `y` grids.
hcopydev	If this variable is set to a non-empty string, then the `hardcopy` command will send the hardcopy file it creates to the printer named `hcopydev`, with the command specified by the variable `lpr_format`.
hcopydevtype	Specifies the type of the printer output to use in the hardcopy command. Possible values are: `postscript`, `laserjet75`, `laserjet100`, `laserjet150`, `laserjet300` and `penplot`.
height	The height of a page to use when printing the output of `asciiplot` or `print` commands.
helppath	The directory path for help file `SmartSpice.hlp`.
history	The number of events to save in the history list.

hspicetopology checker	This command is for the topology rule checker (TRC). Default is false in the regular SmartSpice mode. In the Hspice mode of SmartSpice, this variable is true. New TRC is supporting checks for MOSFET. If floating gates, shortened source-substrate and drain-substrate are detected, a message "Error: no dc path to ground from node x1.in" is printed when `hspicetopologychecker=true` or in Hspice mode. Also, SmartSpice will abort the simulation with the message "job aborted". In regular SmartSpice mode, when `hspicetopologychecker=false` (default), SmartSpice will continue automatic shunting of floating nodes to ground using a resistor of value `dcpath`. Also in the new TRC "Warning: no connection on node 0:0 defined in subckt 0 ignored" will be issued. SmartSpice will continue the simulation.
ignore_dotcards	In interactive modes (command and window), by default, the program saves only vectors that are specified as arguments of `.SAVE`, `.IPRINT`, `.PRINT`, `.IPLOT`, `.PLOT`, `.PROBE`, `.FOUR` and `.MEASURE` statements. If the variable `ignore_dotcards` is set before a circuit is loaded, these dotcards will be ignored, and all voltages and currents will be saved.
ignoreeof	Ignores the exit command when an EOF (**Ctrl-D**) is typed at the beginning of a line.
ignoremcsteperror	If set to `true`, then errors during Monte Carlo step simulation are ignored and Monte Carlo continues to the next step. Previously, SmartSpice aborted simulations in such cases. By default, this functionality is `false` (OFF), and may be turned ON with either SmartSpice variable, or `.OPTIONS` with the same name.
inlinecc	If this variable is set to contain an asterisk (along with any other desired comment characters), it will be recognized as an inline comment character.
iplot_one	Interactive plots of different sweep steps, and different analyses all appear on one graph. Default is `OFF`.
linearize_tstart	Used with the linearize command, this specifies the time at which linearization of a transient vector should begin.
linearize_tstep	Used with the linearize command, this specifies the interval between successive points in a linearization.
list_all_currents	If this variable is specified when a circuit is loading, then the list of vectors that can be saved during the simulation will contain the currents of all devices. This list can be viewed using the `save` command without arguments.

list_all_parameters	If this variable is specified when a circuit is loading, then the list of vectors that can be saved during the simulation will contain all parameters of all models and devices. This list can be viewed using the save command without arguments.
logarithm0	Value for ln(), log() and db() vector functions at zero argument.
long_names_in_tr0	Historically, nodenames in Hspice rawfiles (.tr0 files) have generally been restricted to a maximum of 15 characters in length, by default. However, recent changes in Hspice have allowed longer names. If set (true), this option will allow newer Hspice .tr0 files to be read. It is planned that this option will eventually become the default. Default=0, short name; default=1, long name.
lpr_format	This command is used to send a hardcopy file to a printer (if the hcopydev variable is set). The default is "lpr_ -P%s %s", where the first %s will be replaced by the printer name specified by the hcopydev variable, and the second %s will be replaced by the hardcopy file name.
max_num_rawfile_read_errors	By setting the variable to a small integer value, a user can set an upper limit on the number of errors reported. On each attempt to read an incomplete rawfile, no more than max_num_rawfile_read_errors will be reported. If the variable is not set, no change in behavior will be seen. If a user, in interactive mode, try to load a rawfile which is still being created by another SmartSpice, a long list of errors may be produced.
maxvolt/maxamp	Allows you to define MAXVOLT and MAXAMP options through a .control block.
measout	Means the same as savemeasures (see below).
measureresultslocal	Controls storage of measure results. When set to true, results will be stored with the circuit and will have the same lifetime. If the circuit is removed, its measure results will be removed too. When set to false, results will be in global storage. Results will exist as long as SmartSpice is running. If several circuits make measurements with the same name, the latest will override previous one. Default is true.
meas_fail_default_value	Sets the user default value of the failed measure. SmartSpice hard coded default for failed measure is 1.79769E+308. You might override the default using this variable, for example set meas_fail_default_value=-1000.
memorylimit	This variable specifies a minimal physical or virtual memory size when a simulation will be performed. Default is 50 (means 50 MB).

Note: The `memorylimit` value is specified in mega bytes.

Example

```
.control
set memorylimit=100
set disklimit=100
set safemode=1
.endc
```

In this example the following setting will be applied:

- memory size limit is 100 MB
- disk space limit is 100 MB
- physical memory and disk will be monitored

and SmartSpice prints information:

```
Total physical memory size: 1024 MB
Available physical memory size: 143 MB (limit 100 MB)
Total virtual memory size: 4555 MB
Available virtual memory size: 1907 MB
Available disk size: 1645250 MB (limit 100 MB)
```

msearchauto	1 - automatically choose the criteria for the upper boundary for the geometry binning. When the option `MSEARCH=1` and no model has been found for the specific instance, SmartSpice will try to use “less and equal” criteria (option `MSEARCH=0`) for the upper boundary for geometry binning
nfreqs	The number of frequencies to compute in a Fourier analysis. The minimum is `1`.
noasciiplotvalue	Suppresses printing the value of the first variable plotted with `asciiplot` on the left side of the graph.
noaskquit	Suppresses confirmation request if there are any suspended circuits or unsaved plots before quitting. Normally, SmartSpice requests confirmation before quitting. Default is `true` in batchmode, and `false` if otherwise.
noaskstopsim	Prevents the confirmation dialog from appearing when the **Stop** button is pressed on the **Runtime** dialog during simulation. By default, the variable `noaskstopsim` is `off` (false), and the confirmation dialog will appear, otherwise SmartSpice stops the simulation process after pressing **Stop**.
nobreak	Suppresses adding page breaks when doing an `asciiplot` or a print.
noclobber	When output is redirected with >, it does not overwrite an existing file.

nocreatefourierplot	Forces SmartSpice to remove the plot with FFT/Fourier results after the analysis is finished.
noglob	Does not expand the characters *, ?, [and] in an input line to match filenames.
nomoremode	Normally, this variable is unset, which means that SmartSpice pauses and waits for you to press the **Enter** key when more output is generated by a single command than will fit on the screen. Along with the **Enter** key, the following commands are also recognized: • q: Discard the rest of the output. • c: Print the rest of the output without pausing. • ?: Print a help message. Setting the nomoremode variable disables this feature and causes all output to be printed without pausing.
nonomatch	If noglob is unset and a global expression cannot be matched, use the global characters *, ?, [and] literally.
nopadding	Suppress complement of plot vectors in case that their length is smaller than length of the longest vector in the plot. Default is off.
noprintscale	Suppresses printing of the scale when a print col command is issued.
notranlinearize	When running in batch mode, the results of a transient simulation are linearized before printing. Setting this variable will suppress this linearization.
numdgt	Specifies the number of digits to print when printing tables of data (fourier, print col). The minimum is 0, the maximum is 16.
optionset=val	Set the group of related options.

optionset value	.OPTIONS Parameter Name	.OPTIONS Parameter Value
1	ACCURATE	1
	COEF1(FT)	0.2
	RELMOS	0.01
	FFT_ACCURATE	
	RELTOL	0.001
	VNTOL	5.e-5
	ABSTOL	1.e-9
	TRTOL	7
	METHOD	TRAP
	VSTALIM	0.8
	-hspice mode	ON
2 Will take effect if -hspice and .OPTIONS ACCURATE are specified directly	COEF1(FT)	0.2
	RELMOS	0.01
	FFT_ACCURATE	
	RELTOL	0.001
	VNTOL	5.e-5
	ABSTOL	1.e-9
	TRTOL	7
	METHOD	TRAP
	VSTALIM	0.8
3	ACCURATE	1
	CNODE(CSHUNT)	1.e-12
	ITL1	5000
	ITL2	5000
4	ACCURATE	1
	CNODE(CSHUNT)	1.e-12
	CONV	1
	ACCEPT	TRUE
	ITL2	5000

optionset=1	An Hspice/accurate compatible option set for simulations.

optionset=2	Sets Hspice default values for accurate mode. Unlike the variable value optionset=1, the optionset=2 will only take effect if you directly specify -hspice flag in the command line of SmartSpice, and .OPTIONS ACCURATE in the netlist.
optionset=3	Recommended for transient analysis. This set of options helps to fix the "Time step too small" problem (CNODE=1.e-12). The option ITL1=5000 allows more regular Newton's iterations for OP calculation. If the regular Newton's iteration failed, SmartSpice automatically goes to advanced stepping algorithm and uses 5000 iterations to find OP. This set of options have to be used if transistor/diode models behave properly, and IV curves are smooth.
optionset=4	Recommended for transient analysis to cover failure during OP calculation due to slow convergence (ITL2=5000), to help to fix the "Time step too small" problem (CNODE=1.e-12), to automatically accept the solution during advanced stepping in the OP calculation (ACCEPT), to improve IV characteristic of transistors and diodes when IV curves are not smooth enough, and use the DCGMIN stepping option (CONV=1) to fix internal device conductances. This set of options is recommended for cell characterization to cover convergency failures.

New Default Value of the .Option DCPATH Under optionset=2 in the hspice Mode

SmartSpice, based on a circuit topology analysis, finds floating nodes and automatically connects them to the ground using a large resistor 1/DCPATH, where the default value of DCPATH is 1e-11. If the variable optionset=2 is set in the SmartSpice.ini, and SmartSpice is run in the hspice compatible mode (smartspice -hspice...), then DCPATH is set to 0 and a warning is printed:

```
"if optionset = 2  and mode is HSPICE then dcpath = 0";
"Zero diagonal value detected at node %s in equation solver, which
might cause convergence problems. If your simulations fail, try
adding a large resistor between node () and ground"
```

As an alternative to avoid a nonconvergence, specify the .IC statement for floating node. Setting DCPATH=0 allows the automatic connection of a large resistor of a floating node to be blocked, and makes SmartSpice compatible in the Hspice way.

override_post=val	This variable gives you the possibility to redefine the type of rawfile produced by SmartSpice. The override_post variable has more priority than the post option and the rawfiletype variable. For value settings see Section 3.14".OPTIONS (Option Specification)", "POST<=val>".

If the option RAWPTS is specified in the input deck, and the variable override_post is set to 0 (which will not save any files), the last setting will be ignored and SmartSpice will generate the rawfile.

SmartSpice only saves analysis measurement results in ascii or hspice_ascii format.

Some file formats can support only a limited number of analyses. For instance, TRAN, DC, AC, NOISE can only be saved in csdf format.

pathprintmode	This variable controls the way the .include and .lib statements, found in input deck during the sorcing, are printed out in output. By default (0), the paths are printed as relative to the main input deck's location. If it set to 1, SmartSpice will output lib/include paths as absolute only if '..' symbols are found in a relative path originating form the location of input deck. If all the include/libs are located in subdirectories of a original directory - then those will be printed in a short form (e.g., relative to the input deck). If it set to 2, the paths will be printed as absolute.
parametric_data_in_raw	Setting this variable to true makes it possible to save the parametric analysis data (.ST, .MODIF, and secondary and nested sweeps) in the regular SmartSpice rawfile. The additional sections with the prefixes modif, sweep and st are added, and the additional vectors TYPE_DEV[PAR] will be inserted in each internal analysis section. These additional sections contain general information about all values of the parameters which are used in parametric analysis. The additional vectors describe concrete values which are used in the current analysis. The vector name TYPE_DEV[PAR] contains: • TYPE: Name of the parametric analysis, which can be param for .ST analysis, modif for .MODIF, sweepAC for .AC, sweepDC for .DC, sweepTRAN for .TRAN, and sweepNESTED for .DC with nested sweep . • DEV: Name of the device that was swept. • PAR: Name of the parameter that was swept.
plothistory	When this variable is set, commands used for plot zooming are included in the history list.
plots	This variable is read-only and contains the names of the plots available. The variable curplot can be set to any plot name; or the word new, in which case it creates a new, empty plot.
plotstyle	Used to determine the plot style if no plotstyle keyword is specified on the command line for the commands plot, hardcopy and asciiplot. Its value can be linplot, to connect points with line segments; combplot, to connect each point to the X-axis; or pointplot, to plot each point as a discrete character.
pointchars	The characters in this string are used to plot successive data values if the pointplot keyword is specified in a plot command.

polydegree	This variable determines the degree of the polynomial that is fit to points when a plot is done. If polydegree is n, then SmartSpice fits a degree n polynomial for every set of n points, and draws 10 intermediate points in between each end point. If the points are not monotonic, then SmartSpice rotates the curve and reduces the degree until a fit is achieved.
polysteps	The number of intermediate points to plot between each actual point used for interpolation. Note that if interpolation is used for plotting, the ticmarks feature is disabled.
print_format	This selects the format in which SmartSpice will print results. Two formats are allowed: engineering and scientific. The output values are printed in scientific form by default. If `print_format` is equal to `0`, then the engineering output format will be used. If `print_format` is equal to `2`, than the scientific output format will be used. The default for `print_format` is `2`. The complete list of suffixes and their values are shown in the Table 5-2.

Table 5-2 Values, Suffix Names and Matching Notations

Value	Name	Notation
10^{12}	tera	t
10^{9}	giga	g
10^{6}	mega	x
10^{3}	kilo	k
1	none	none
10^{-3}	milli	m
10^{-6}	micro	u
10^{-9}	nano	n
10^{-12}	pico	p
10^{-15}	femto	f
10^{-18}	atto	a

printinfo	When this variable is set, SmartSpice prints all info massages. Default is `false`.

program	The full path name of the program.
prompt	The prompt for the SmartSpice command mode. In this string, the character ! is replaced by the current command number.
qtastorelatency	This variable works under variable safemode=3 mode. During raw file saving available quoted space checked for each rawpts segment. qtastorelatency variable stands for damping ratio depending on network load, quoting software used and other unpredictable disk requests efforts. It means “check availability of disk space X times more then next rawpts segment needs to store”. Default is 3.
rawfile	The default name for a rawfile created using the write command.
rawfileprec	Sets the number of significant figures in a rawfile. Default is 15.
rawfiletype	This variable determines the type of rawfile produced by the program. The possible rawfile types are: • ascii • binary • binary_float (float means single precision) • csdf • excel • fsdb (SpringSoft FSDB binary Format) • hspice_binary • hspice_ascii • ssf (SILVACO Standard Structure File) • data (SILVACO Data Format) • tabular • isdb (SimWave ISDB Format) • psf (Cadence PSF ascii Format) • psf_binary (Cadence PSF binary Format) • wsf (Cadence WSF (ascii) Format) • wsf_binary (Cadence WSF (binary)Format)
readfirst	Unless this variable is specified (in batch mode) if there is at least one .ALTER statement in the input deck; or if there are several input decks in one input file, then each deck will be executed immediately after being parsed. If this variable is specified, then all the input decks are parsed first, and then they are executed one at a time. Also, if this variable is not set, SmartSpice will not simulate a circuit in batch mode if there is a .CONTROL portion in the input deck.
renumber	Renumbers input lines when listing a command if a deck has a .INCLUDE statement.

rtscreen	Shows run-time screen during simulation. For window mode, the default is `true`; for command and batch mode, the default is `false`.
safemode	Setting this variable to 1 or 2 makes it possible to check the value of the available memory and the disk space during a simulation. Default value is 0 (which means no checking during simulation). To monitor physical memory size and disk space the `safemode` value must be set to 1. To monitor virtual memory size and disk space the `safemode` value must be set to 2. In special cases when executing environment uses hard disk quota limitation and you can't monitor the remaining quoted space (e.g., due to time expensive simulations consuming days or weeks of modelling time) it can cause loss of data and unreadable raw files in the end of process. To avoid such abnormal behaviour `safemode` value can be set to 3. In this mode additional quota limitation checking mechanism engaged. See the definition of `qtastorelatency` variable.

Syntax

```
set safemode= 0|1|2|3
```

savemeasures	Saves all results of `.MEASURE` statements during simulation, and stores the results in a separate meas plot.
scaletype	Determines the type of scale used by the commands `plot`, `hardcopy` and `asciiplot` if no `gridtype` keywords are given on the command line. Possible values are: • `linlin`: Use linear scales for both axes. • `loglog`: Use log scales for both axes. • `loglin`: Use a log scale for the X-axis. • `linlog`: Use a log scale for the Y-axis. • `polar`: Use polar grid. • `smith`: Use Smith grid.
search	The list of directories that will be searched when a source command (or a command that invokes a script) is specified. The default is the current directory and the SmartSpice directory.
search_always_prepend	A search path specified with `.OPTIONS SEARCH` is appended to the current search path when this variable is set to `true`. Default is `false`.
searchorder	Allows you to change default search order for `.INCLUDE` or `.LIB` statements. Default is 0.

Example

```
set searchorder=1
```

In this example the following search order will be used:

- the directory of the current included file
- the directory of the input (main) deck
- search path

Note: For Hsim compatability use `smartspice -fast`.

---

simulator	Other choices for the simulator variable are: `hspice`, `pspice`, `spectre` and `eldo`. This variable is case independent.
singularsupplycheck	Allows to SmartSpice to skip the check of circuit singular supplies. Singular supply is a an independent voltage source, VCVS and VCCS. They are considered “singular” if not connected to the circuit. Such power supplies might cause a “matrix is singular” issue. SmartSpice automatically terminates singular supplies on ground using 1M resistor and issues a warning message. The default is `true` (do the check) in all SmartSpice modes.
smartspice_verbose	When this variable is set, SmartSpice turns on verbose mode, showing each component as it is invoked. Default is `false`.
splitmeasureplots	When this variable is unset (default), SmartSpice saves all `.MEASURE` results for `.MODIF`/`.ST`/`.MC` analyses in single plot. `splitmeasureplots=true` saves all `.MEASURE` results for different analyses in separated plots. Default is true in -hspice mode.
stoponfatalerrors	Specifies error checking behavior globally (for more details see Section 3.14“.OPTIONS (Option Specification)” `STOPONFATALERRORS`). A particular circuit could override this variable locally with the `.OPTIONS STOPONFATALERRORS`. The variable `stoponfatalerrors` is now set to `1` by default in Hspice-compatibility mode. Otherwise, default is `0`.
strictnumparse	Does not allow trailing characters after a number, unless they are separated from the number with an underscore (“_”).
subcktcheck	Allows SmartSpice to skip the checking of subcircuits which are defined with the same name. The default is `true` (do the check) in all SmartSpice modes, except HSIM mode (do not do the check).
subckt_delimiter	Sets the character used to separate subcircuit levels.

suppressshort devicewarnings	This variable controls the output of the warning message: `"Device 'name' with terminals connected together is ignored."` Default is `off` (`false`).
suppressfftoutput	Suppresses FFT printout. Default is `false` (`off`).
syntax_form	Specifies one of two syntax forms: `0` (default) and `1`. Setting `syntax_form` to `1` will change the parsing scheme for some input statements: • Only one inline comment character (`;`) is supported. By default, two inline characters (`;` and `$`) are supported. • The keyword for the resistor model type is `RES` (instead of the default `R`). • The keyword for the resistor model parameter resistance multiplier is `R` (instead of the default `RES`). • The sub circuit delimiter "`:`" is supported in output statements.
term	Describes a terminal type.
ticmarks	Prints an "o" every ten points for each curve plotted. This variable can also be set to a number, which is the number of points between each ticmark.
unixcom	Specifies if a command is issued that is not a built-in command, then SmartSpice will execute it as a UNIX command (or a DOS command on Windows).
use_mfiles	Forces SmartSpice to save data in multiple files if `.MODIF`, `.TEMP` or `.ALTER` statements are used. Each plot will be stored in a separate file with the file extension indicating the type of plot stored.
use_syntax0_libs	Setting this variable to `true` forces SmartSpice to use the algorithms used by `syntax_form=0` when parsing libraries, even if `syntax_form = 1`.
usedegrees	• `true` or `1`: the angles and phases are output in degrees. • `false` or `0`: the angles and phases are output in radians. Default is `true` or `1`. If `.OPTION USEDEGREES` is specified together with the variable, it will override the variable value. For example, if the variable `usedegrees=true` and the `.option usedegrees=false`, SmartSpice will use radians for the angles and phases.
warn_unrecognized	Checks input decks for unrecognized options (including incorrectly used options), and generates warning messages.
waveform_version	Specifies the version of waveform viewer to plot with. The version number must be enclosed in double quotes.

width	The width of a page to use when printing the output of `asciiplot` or `print` commands. The minimum is `40`, and the maximum is `130`.

Plot-Specific Variables

The following variables are specific to each plot. When the current plot changes, these variables take on new values, and they cannot be altered. In addition to these variables, any variables defined in the rawfile in an `OPTION:` line is associated with the plot and are read-only.

curplot	The name of the current plot.
curplotdate	The date associated with the current plot. This is generally the date of the simulation.
curplotname	The type name of the current plot. This is a description of the type of simulation done.
curplottitle	The title of the circuit associated with the current plot.

Chapter 6
Outputs

6.1 Introduction

The SmartSpice option statements .PRINT, .PLOT, .PROBE, .SAVE, .UNSAVE, .IPRINT, .IPLOT, .GRAPH, .FOUR, .FFT and .MEASURE (and corresponding commands print, plot, etc.) use the notation:

```
outvar1  <outvar2 ...>
```

This notation refers to output variables, macros and expressions. This chapter describes the syntax of output variables, and provides examples of their application. The syntax of macros and expressions is described in Chapter 4 Expressions.

6.2 Output Variable Names

During simulation, SmartSpice calculates the circuit's variables, vectors and element parameters. The basic types of the simulation results are:

- Node voltages, voltage differences between specified nodes, and voltages across two-terminal devices.
- Currents through independent voltage sources and inductances.
- Currents through two-terminal devices.
- Currents through terminals for multi-terminal devices and subcircuits.
- Power dissipation for device, subcircuit and complete circuit.
- Device and/or model parameters.
- Nodal capacitance.

Output parameters provide the appropriate simulation results. Their definition is possible by using the .OPTIONS, .LET and .MEASURE statements, or by referencing specific variable elements.

Syntax

- Node voltage output is specified by:

  ```
  v(nodename)
  ```

  where nodename is either a number or descriptive text string.

- Voltage difference between specified nodes:

  ```
  v(nodename1, nodename2)
  ```

- Current through independent voltage source and inductance are specified by:

  ```
  i(Vxxx), i(Lxxx)
  ```

SmartSpice is used to define the device parameter name in two forms: @devicename [parametername], or by using devicename directly. Normally devicename consists of the letters, numbers, and special symbols, and the first letter of the devicename indicates the device type.

Two-terminal devices:

- A: Analog behavioral devices
- C: Capacitors
- D: Diodes
- E: Voltage-controlled voltage sources
- F: Current-controlled current sources
- G: Voltage-controlled current sources
- H: Current-controlled voltage sources

- I: Independent current sources
- L: Inductors
- R: Resistors
- V: Independent voltage sources

Multi-terminal devices:

- J, Z: JFET / MESFET
- B: MESFET/IBIS buffer depending on the syntax used.
- M: MOSFET
- Q: BJT

Subcircuit definition:

- X

The voltage across a two-terminal device can be specified by:

```
v(devicename)
```

or the equivalent:

```
@devicename[v]
```

If the device and node are found to share the same name, SmartSpice will generate two vectors:

- v(name) for node,
- and @name[v] for device.

The voltage at the terminals of a multi-terminal device might be specified in the form:

```
Vn(devicename)
```

where n is the one-letter terminal name abbreviation (D, G, S, etc. See Table 6-1).

Current through a two-terminal device can be specified by:

```
i(devicename)
```

or the equivalent:

```
@devicename[i]
```

Currents through the terminals of a multi-terminal device might be specified in the form:

```
In(devicename)
Iall(devicename)
```

where n is the node position number in the device statement, or a one-letter terminal name abbreviation (D, G, S, etc. See Table 6-1).

Table 6-1 Multi-terminal Device Description

Device Name	Terminal Name	Node Position Number	Terminal Name Abbreviation
J, B, Z (JFET/ MESFET)	Drain	1	D
	Gate	2	G
	Source	3	S

Table 6-1 Multi-terminal Device Description

Device Name	Terminal Name	Node Position Number	Terminal Name Abbreviation
M (MOSFET)	Drain	1	D
	Gate	2	G
	Source	3	S
	Bulk (Source)	4	B
Q (BJT)	Collector	1	C
	Base	2	B
	Emitter	3	E
	Substrate	4	S
T (Transmission Line, Lossless)	Input port 1 - n1	1	1
	Input port 1 - n2	2	2
	Output port 2 - n3	3	3
	Output port 2 - n4	4	4

For example, `I2` is the current for the second node of a three or four terminal device. If `n` was not specified, SmartSpice assumes the first node.

`Iall` is an alias for BJT, JFET/MESFET/MOSFET and T devices, and is equivalent to:

```
I1(devicename)  I2(devicename)  I3(devicename)  I4(devicename)
```

If `devicename` is defined in a subcircuit, append the subcircuit name and dot to the `devicename`.

Currents through subcircuit terminals are identified by:

```
i(subcktnodename)
```

where `subcktnodename` is the sub-circuit `nodename` for external reference.

For nested subcircuits, expanded names must be used. For example, if a circuit has a call to subcircuit `subcktname1`, and the `subcktname1` definition contains a call to subcircuit `subcircuit2`, then the expanded nodename for the `subcircuit2` terminal with `nodename1` will have the following format:

```
subcktname1.subcktname2.nodename1
```

Device dissipated power is specified by:

```
p(devicename) or w(devicename)
```

Sub-circuit dissipated power is specified by:

```
p(subcktname) or w(subcktname)
```

Circuit dissipated power is specified by:

```
power
```

Device and model parameters of the devices and models are specified by:

```
@devicename[parametername]
@modelname[parametername]
```

Nodal capacitance is specified by:

```
cap(nodename)
```

Examples

```
v(2)        Voltage at node 2
v(r12)      Voltage across resistor r12
i(vin)      Current through voltage source vin
i(r12)      Current through resistor r12
ig(m1)      Current into gate of MOS transistor m1
i(x1.out)   Current at node out of sub-circuit x1
p(r12)      Power dissipated by resistor r12
p(x1)       Power dissipated by sub-circuit x1
@d1[area]   The area of diode d1
cap(2) Nodal capacitance at node 2
```

In some cases (e.g., Monte-Carlo analysis) devices sharing the same model may have different model parameters. In this case, the syntax for accessing the device's model parameters is the same as that used for a normal device parameter, except that the model parameter name is given (i.e., `@device[model_parameter]`). If the device and model parameter have the same name, the device parameter will be returned. To obtain the model parameter, a prefix of `model_` should be added to the parameter to signify that the device's model parameter is required.

Examples

```
@dmod[cjo]
```

The junction capacitance of the diode model dmod. There is no device parameter with the same name.

```
@m1[model_vto]
```

Model parameter `vto` from device `m1`. The name `vto` would be ambiguous, without the `model_` prefix.)

When a device within a sub-circuit is flattened, the flattened name is created by appending the device name to the sub-circuit path. SmartSpice prepends the first character of the device name to the flattened name so that the device type can be identified using only the first character of the flattened name. However, it does not require that you prepend this character when using the access functions. The flattened name of the device `m1` in the sub-circuit `x1` is `m.x1.m1` is used to access device parameters (i.e., `id(m.x1.m1)` or `@m.x1.m1[gds]`). You may use the same syntax to access the data, or use the aliases `id(x1.m1)` and `@x1.m1[gds]`.

When output statements (such as `.PROBE`, `.PRINT`, etc.) are contained into sub-circuit definitions, the output variable names are expanded as part of a sub-circuit definition.

Example:

```
VIN 1 0 DC 1 PULSE ( 1 0 0 1NS 1NS 19NS )
X1 1 0 TEST
C1 1 0 10PF
.TRAN 1.NS 40NS

.SUBCKT Test Input Output
```

```
R1 Input Output 200
.PROBE @R1[res]
.PROBE I(R1)
.ENDS
.END
```

Output will contain `@X1.R1[res]` and `I(X1.R1)` vectors.

Another syntax form for the output of device user-input parameters, state variables, store charges, capacitor currents, capacitances and derivatives is:

```
LVnn(devicename)
```

or

```
LXnn(devicename)
```

where:

LV	Form to obtain output of user-input parameters and state variables.
LX	Form to obtain output of stored charges, capacitor currents, capacitances and derivatives.
nn	Code number for the desired parameter, given in the tables in this section.
devicename	Name of the device.

Example

```
.print ids=lx4(m1) vth=lv9(m1) vdsat=lv10(m1) gmbs=PAR('lx9(ma)')
```

Table 6-2 Device Parameters LV, LX Code Numbers

Parameter name	Alias	Description
Resistor (R)		
GEFF	LV1	Computed effective conductance 1/Reff
RES	LV2	Resistance
TC1	LV3	First temperature coefficient
TC2	LV4	Second temperature coefficient
Capacitor (C)		
CEFF	LV1	Computed effective capacitance
IC	LV2	Initial condition
C	LX1	Current through capacitor
V	LX2	Voltage across capacitor
CAP	LX3	Capacitance

Table 6-2 Device Parameters LV, LX Code Numbers

Parameter name	Alias	Description
Inductor (L)		
FLUX	LX0	Flux in the inductor
VOLT	LX1	Voltage across inductor
C	LX2	Current through inductor
IND	LX3	Inductance
Mutual Inductor (K)		
coefficient	LV1	Mutual inductance
Voltage-Controlled Current Source (G)		
C	LX0	Current through the source
Voltage-Controlled Voltage Source (E)		
V	LX0	Source voltage
C	LX1	Current through the source
Current-Controlled Current Source (F)		
C	LX0	Current through the source
Current-Controlled Voltage Source (H)		
V	LX0	Source voltage
C	LX1	Current through the source
Independent Voltage Source (V)		
DC	LV1	DC/transient voltage
ACMAG	LV2	AC voltage magnitude
ACPHASE	LV3	AC voltage phase
Independent Current Source (I)		
DC	LV1	DC/transient current
ACMAG	LV2	AC current magnitude
ACPHASE	LV3	AC current phase
Diode (D)		
AREA	LV1	Diode area factor
AREAEFF	LV23	Effective area
IC	LV2	Initial voltage across diode

Table 6-2 Device Parameters LV, LX Code Numbers

Parameter name	Alias	Description
VD (VOLTAGE)	LX0	Voltage across diode
ID (C)	LX1	Current through diode
GD	LX2	Diode conductance
CHARGE	LX3	Diode capacitor charge
CD	LX5	Diode capacitance
BJT (Q)		
AREA	LV1	Area factor
ICVBE	LV2	Initial condition for base-emitter voltage
ICVCE	LV3	Initial condition for collector-emitter voltage
M	LV4	Multiplying factor
FT	LV5	Cutoff frequency ft
CS (IS)	LV6	Substrate current
BETA	LV10	Current gain (Ic/Ib)
RB	LV14	Base resistance
VBE	LX0	Base-emitter voltage (internal)
VBC	LX1	Base-collector voltage (internal)
CC	LX2	Collector current
CB	LX3	Base current
GPI (GBI)	LX4	Forward base conductance (gpi) (gbe=dIb/dVbe)
GMU (GBC)	LX5	Reverse base conductance (gmu) (gbc=dIb/dVbc)
GM	LX6	Transconductance (dIc/dVbe)
GO	LX7	Output conductance (dIc/dVce)
QBE	LX8	Charge stored in CBE
QBC	LX10	Charge stored in CBC
QCS	LX12	Charge stored in CCS
QBX	LX14	Charge stored in CBCX
GX	LX16	Base conductance (1/rb)
CBE (CPI, CAPBI)	LX19	Base-emitter junction internal capacitance
CBC (CMU, CAPBC)	LX20	Base-collector junction internal capacitance
CCS (C0)	LX21	Substrate-collector junction capacitance

Table 6-2 Device Parameters LV, LX Code Numbers

Parameter name	Alias	Description
CBCX	LX22	Base-collector external capacitance
VS	LX24	Substrate voltage (external)
JFET (J)		
AREA	LV1	Area factor
ICVDS	LV2	Initial drain-source voltage
ICVGS	LV3	Initial gate-source voltage
VGS	LX0	Internal gate-source voltage
VGD	LX1	Internal gate-drain voltage
IGS	LX2	Gate-to-source current
CD	LX3	Drain current
IGD	LX4	Gate-to-drain current
GM	LX5	Drain to source AC transconductance controlled by vgs
GDS	LX6	Drain to source AC conductance controlled by vds
GGS	LX7	Gate to source AC conductance
GGD	LX8	Gate to drain AC conductance
QGS	LX9	CAPGS charge
CQGS	LX10	Gate-source charge current
QGD	LX11	CAPGD charge
CQGD	LX12	Gate-drain charge current
CAPGS	LX13	Gate to source capacitance
CAPGD	LX14	Gate to drain capacitance
QDS	LX16	Drain-source charge
CQDS	LX17	Drain-source charge current
GMBS	LX18	Bulk transconductance
MESFET (B, Z)		
AREA	LV1	Area factor
ICVDS	LV2	Initial drain-source voltage
ICVGS	LV3	Initial gate-source voltage
VGS	LX0	Internal gate-source voltage
VGD	LX1	Internal gate-drain voltage

Table 6-2 Device Parameters LV, LX Code Numbers

Parameter name	Alias	Description
CD	LX3	Drain current
GM	LX5	Drain to source AC transconductance controlled by vgs
GDS	LX6	Drain to source AC conductance controlled by vds
GGS	LX7	Gate to source AC conductance
GGD	LX8	Gate to drain AC conductance
QGS	LX9	CAPGS charge
QGD	LX11	CAPGD charge
CAPGS	LX13	Gate to source capacitance
CAPGD	LX14	Gate to drain capacitance
GMBS	LX18	Bulk transconductance
MOSFET (M)		
LEFF	LV1	Effective channel length (m)
WEFF	LV2	Effective width (m)
AD	LV3	Drain area
AS	LV4	Source area
ICVDS	LV5	Initial drain-source voltage
ICVGS	LV6	Initial gate-source voltage
ICVBS	LV7	Initial bulk-source voltage
VTH	LV9	Threshold voltage (V)
VDSAT	LV10	Saturation drain voltage
PD	LV11	Drain perimeter
PS	LV12	Source perimeter
BETA	LV21	Transconductance (A/V^2)
GAMMA	LV22	Body effect factor ($V^{1/2}$)
VFB	LV26	Flat band voltage
CAPGSO	LV36	Gate-source overlap capacitance
CAPGDO	LV37	Gate-drain overlap capacitance
CAPGBO	LV38	Gate-bulk overlap capacitance
VBS	LX1	Source to bulk voltage

Table 6-2 Device Parameters LV, LX Code Numbers

Parameter name	Alias	Description
VGS	LX2	Gate to source voltage
VDS	LX3	Drain to source voltage
IDS (ID)	LX4	Drain to source DC current
IBS (CBS)	LX5	Bulk-source junction current
IBD (CBD)	LX6	Bulk-drain junction current
GM	LX7	D-S transconductance controlled by vgs
GDS	LX8	Drain to source conductance controlled by vds
GMBS	LX9	D-S bulk transconductance controlled by vbs
GBS	LX10	Bulk to source conductance
GBD	LX11	Bulk to drain conductance
Charge Conservation Model Parameters		
QB	LX12	Bulk charge
CQB	LX13	Bulk charge current
QG	LX14	Gate charge
CQG	LX15	Gate charge current
QD	LX16	Drain charge
CQD	LX17	Drain charge current
CGGB	LX18	$CGGB = \frac{\partial Q_G}{\partial V_{GB}}$
CGDB	LX19	$CGDB = \frac{\partial Q_G}{\partial V_{DB}}$
CGSB	LX20	$CGSB = \frac{\partial Q_G}{\partial V_{SB}}$
CBGB	LX21	$CBGB = \frac{\partial Q_B}{\partial V_{GB}}$
CBDB	LX22	$CBDB = \frac{\partial Q_B}{\partial V_{DB}}$

Table 6-2 Device Parameters LV, LX Code Numbers

Parameter name	Alias	Description
CBSB	LX23	$CBSB = \frac{\partial Q_B}{\partial V_{SB}}$
QBD	LX24	Bulk-drain charge
QBS	LX26	Bulk-source charge
CAPBS	LX28	Bulk-source capacitance
CAPBD	LX29	Bulk-drain capacitance
CDGB	LX32	$CDGB = \frac{\partial Q_D}{\partial V_{GB}}$
CDDB	LX33	$CDDB = \frac{\partial Q_D}{\partial V_{DB}}$
CDSB	LX34	$CDSB = \frac{\partial Q_D}{\partial V_{SB}}$

6.2.1 Wildcards

SmartSpice supports wildcards '*' and '?' for output variable with `v(nodename)` and `i(devicename)` names. The character '*' means any number (0 or more) of any characters. The character '?' can be used to refer to any single character.

The wildcard functionality has been improved to support subcircuit as the filter reference.

Examples

```
.print i(*)
.print i(x1.*.c*)
.print id(m.x*.*)
```

In the first example, SmartSpice prints currents for devices within netlist.

The second example prints all capacitor currents for this x1 subcircuit instance and below.

The third example prints drain currents for all MOSFET devices within all subcircuits.

Examples

```
.PRINT V(ABC*)
.PRINT V(X?Z)
```

The first statement will print out the voltages at any node whose name begins with the character ABC followed by any number of characters, or none at all. The second statement will print voltages at nodes whose names consist of the letter X, followed by any single character (number or digit), or the letter Z.

Wildcards can be used with the following commands:

```
.save, .unsave, .probe, .print, .iprint, .plot, .iplot, .graph,
```

```
.fft,
```

and in the following types of analysis: TRAN, AC and DC.

The following macros accept the wildcard characters:

```
vr(x),   vi(x),   vm(x),   vp(x),   vgd(x),   vdb(x),
v(x,y), vr(x,y), vi(x,y), vm(x,y), vp(x,y), vgd(x,y), vdb(x,y).
```

Example

```
DEMO EXAMPLE 1: INPUT FILE *
VIN 1 0 DC 0 SIN(0 0.1 5MEG) AC 1
VCC andrpi 0 DC 10
VEE 9 0 DC -12
RS1 1 2 1K
RS2 opout 0 1K
RC1 pash andrpi 10K
RC2 4 andrpi 10K
RBIAS 7 andrpi 20K
CLOAD pash 4 5PF
Q1 pash 2 6 QNL
Q2 4 opout 6 QNL
Q3 6 7 9 QNL
Q4 7 7 9 QNL

* Save magnitude and phase for all vectors in AC analyses
.probe ac vm(*) vp(*)
*Print real and imaginary parts of all vectors in AC analyses
.print ac vi(*) vr(*)
* Saves real part of voltages at nodes whose name ends in 'i',
* and magnitude of voltages at nodes whose name contains the letter
'p',
* in AC analyses
.probe ac vr(*i) vm(*p*)
* Saves real part of voltages between node 2 and all nodes whose
name ends
* in the letters 'pi', for AC analyses
.probe ac vr(*pi,2)
```

Transistor's Current Macro Templates

- `id, ig, is, iall`: For JFETs. Templates must start with `B/J/Z`.
- `id, ig, is, ib, iall`: For MOSFETs. Templates must start with `M`.
- `ic, ib, ie, is, iall`: For bipolar transistors. Templates must start with `Q`.

Examples

```
.PRINT IB(Q*) $$
```

Base current for all bipolar transistors.

```
.SAVE id(m.x1*) $$
```

Drain current for all subcircuit x1 MOSFETs.

```
.SAVE iall(m*) $$
```

Drain, Gate, Source and Bulk(Source) currents for all MOSFETs.

T (Transmission Line, Lossless) Element Current Macro Templates

- `i1, i2, i3, i4, iall`: For lossless transmission line element. Templates must start with T.

Examples

```
.PRINT I1(t*) $$
```

The node n1 at input port 1current for T elements.

```
.PROBE iall(t*) $$
```

The nodes n1, n2 at input port 1 and the nodes n3, n4 at output port 2 currents for T elements.

Instance Parameters Macro Templates

Support for templates in instance parameters. Templates can be used in place of the instance name:

```
@q*[area]
```

6.2.2 UNIX Basic Regular Expressions

```
v(/<RegExp>/), v(/<RegExp1>/,/<RegExp2>/), i(/<RegExp>/)
```

A significant difference from UNIX's basic regular expressions is that `\\` must be used instead. If you plan to use the `$` symbol in a regular expression, you must exclude it from inline comments (`inlinecc` variable) first.

Example

```
V1 1 0 1
X1 1 0 Test
X2 1 0 Test
X3 1 0 Test
.subckt Test In Out
R1 In Internal 1
R2 Internal Out 1
R3 Internal Out 2
.ends
.tran 1n 5n
.print v(/^x[12]\\.Internal/)
.print i(/^r\\.x3\\.R./)
.option nomod
.end
```

6.2.3 Bus Notation

A new notation has been introduced into SmartSpice to allow a compact expression of a multiple bit wire bus to be used. From this expression, simply state the members of a wire bus that you want to generate vectors, or probe for. This syntax can be used in conjunction with the `.SAVE` and `.PROBE` statements.

```
Bus_name<n:m:i>
Bus_name[n - m - i]
```

`Bus_name`	An ASCII character string naming the set of wires.
`N, M, I`	A positive integer number.

< > or **[ ]**	Indicates the expression to be expanded.

```
Syntax       Num. Bit's   Expanded form
B<0:2>         3              b<0>,b<1>,b<2>
B<0:2:1>       3              b<0,1,2>
B<3:0:2>       2              b<3,1>
B<0:1,2:2>     3              b<0,1,2>
```

DATA<2,1,0> represents DATA<2>,DATA<1> and DATA<0>.

Bus Expression Expanded to the single bit level:

- DATA<0:3:2> indicates a 2-bit bus containing elements:
  DATA<0> and DATA<2>
- DATA<1:3:2> indicates a 2-bit bus containing elements:
  DATA<1> and DATA<3>
- DATA<0:3> indicates a 4-bit bus containing elements:
  DATA<0>, DATA<1>, DATA<2> and DATA<3>
- DATA<2:0> indicates a 3-bit bus containing elements:
  DATA<2> , DATA<1> and DATA<0>

The following 2 examples show how the different forms of this abbreviation for a set of bus wires can be used.

Example 1

```
* Test bus notation in .COMMANDS

V1 1      0      1
R1 1      bus<1> 1
R2 bus<1> bus<2> 1
R3 bus<2> bus<3> 1
R4 bus<3> bus<4> 1
R5 bus<4> 0      1

.options nomod

.tran 1n 5n

.save v(bus<1-4>)
.print v(bus<1:4>)
.print v(bus<1-4-2>)
.print v(bus<4-1-2>)
.print v(bus<1,3:4>)

.end
```

Example 2

```
* Test bus notation in shell commands

V1 1      0      1
R1 1      bus[1] 1
R2 bus[1] bus[2] 1
R3 bus[2] bus[3] 1
R4 bus[3] bus[4] 1
R5 bus[4] 0      1
```

```
.options nomod

.control
save v(bus[1-4])
tran 1n 5n
print v(bus[1:4])
print v(bus[1-4-2])
print v(bus[4-1-2])
print v(bus[1,3:4])
.endc

.end
```

SmartSpice allows you to use collapsed bus notations with X-calls, .SUBCKT and YVLG device statements.

The following two examples show how to use bus notations.

Example

```
* Bus notation in subcircuits (subckt) and X-calls
m_m1 out sb<0> vdd mpa l=6u w=12u
m_m2 out sb<0> 0 mna l=6u w=10u
rload out 0 1meg
vdd vdd 0 3
vin1 sb<1> 0 dc 2.0 sin (2.0 0.6 50e6 0 0 90)
vin2 sb<2> 0 dc 2.0 sin (2.0 0.4 55e6 0 0 90)
xr21 sb<0:2> out sr2
xr11 sb<0:2> out 0 sb<2:1> sr1 m1=1 m2 = 1
xr12 sb<0,1:2> out 0 sb<2:1> sr1 params: m1=1 m2 = 1
.subckt sr1 bs<0> bs<1> bs<2> n1 n2 bss<2:1> m1=1
+ Rsiga_cf = 'm1+1.20u/(0.67+0.33*1.2u)'
r1 bs<1> bs<0> 1
r2 bs<2> bs<0> 1
r3 bss<1> bss<2> 1
r4 n1 n2 1
xr21 bs<2:0> n1 sr2
.ends sr1
.subckt sr2 bs2<0:2> n2 FCC='fc1'
R1 bs2<1> bs2<0> 1
R2 bs2<2> bs2<0> 1
R3 bs2<2> n2 1
.ends sr2
.end
```

Bus notations can use .PARAM parameters in its ranges. Parameters can be used with or without apostrophes.

Example

```
* Using bus notation with Verilog-A device and using parameters in
bus notation
.verilog "delayGenerator.va"
* PWL Reference
.param size = 1024
.param first_bit = 0
.param last_bit = 'size - 1'
.param print_bits = 10
```

```
VpwlIn pwlIn 0 PWL 0 0 5n 1 10n 1 15n -1 20n -1 25n 1
XVLG_delayGenerator        pwlIn        SCAN[‘first_bit’:last_bit]
delayGenerator
.save v(scan[0:last_bit]) v(pwlIn)
.print v(scan[0:print_bits]) v(pwlIn)
.end
```

6.2.4 MACROS

SmartSpice supports the following macros in output statements:

ALL	All independent voltages and currents.
ALL(I) and **ALL(V)**	These macros cause SmartSpice to save all currents and voltages during simulation. Macro ALL_SUB(I) saves all subcircuit currents during simulation.
TOP(I) and **TOP(V)**	These macros cause SmartSpice to save node voltages and branch currents only for circuit top-level nodes and branches (ignoring subcircuit levels). TOP(I) is equivalent to HEIR0(I). TOP(V) is equivalent to HEIR0(V)
<SUBCKT>.ALL_SUB(I)	This macro saves all terminal currents for the subcircuit SUBCKT, and for all dependent subcircuits.
<SUBCKT>.ALL(V)	This macro saves all device currents and node voltages for subcircuit SUBCKT, and all dependent subcircuits, as does SUBCKTNAME.ALL(I).
ALL_SUB(V)	This macro can be used by the .SAVE and .PROBE statements, and causes SmartSpice to save all subcircuit terminal voltages during simulation.
<SUBCKT>.ALL_SUB(V)	This macro saves all terminal voltages for the subcircuit instance <SUBCKT>, and for all contained subcircuits. <SUBCKT> is defined as XP1.XSS.
HIER*(V)	Causes SmartSpice to save subcircuit terminal voltages for any specific circuit hierarchy level.
HIER*(I)	Causes SmartSpice to save subcircuit terminal currents for any specific circuit hierarchy level.

HIER*(V | I)

Example

```
.save HIER1(V)
.save HIER2(I)
```

In the first example where V is specified, the terminal voltages will be saved for all subcircuits with the hierarchy level that equals 1.

In the second example where I is specified, the terminal currents will be saved for all subcircuits with the hierarchy level that equals 2.

6.2.5 Optional Settings for Output Dot Statements

Syntax

```
.PRINT <anytype> outvar1 <outvar2 ...> <subckt = sub_name>
+ <level = val1> <filter = pattern>
```

The following optional settings are supported in this release:

subckt=sub_name	The output applies to the specified node name(s) and/or element name(s) within all instances of the specified sub-circuit name. This `subckt` setting is equivalent to placing any output statements (`.print`, `.save`, `.plot` and `.probe`) within the sub-circuit definition.
level=val	This setting is effective only when the wildcard character is specified in the output variable. The level value `val` specifies the number of hierarchical depth levels when the wildcard node/element name matches. • When `val` is set to 1, the wildcard match applies to the same depth level where the output statement is located. • When `val` is set to 2, it applies to the same level and to one level below the current level where output is located. • When `val` is set to -1, the wildcard match applies to all the depth levels below and including the current level of output statement. • The default value of `val` is -1.
filter=pattern	This setting is effective only when the wildcard character is specified in the output variable. Nodes/elements that match the pattern specified in the filter clause will not be printed.

Examples

```
.print v(x1.*) v(q*) subckt=sub1
.save v(*) level=1
.print v(x1.*) level=3
.print v(x1.x2.*) filter= x1.x2.n*
```

In the first example, SmartSpice prints node voltages for all nodes which match `x1.*` and `q*` in all instances of subcircuit `sub1`.

The second example saves all top-level node voltages.

The third example prints node voltages within sub-circuit instance `x1`. The printout includes nodes within `x1`, and the nodes in two levels below `x1`. The nodes located more than two levels deeper than the level of instance `x1` are excluded.

The fourth example prints the voltages of all nodes in subckt `x1.x2` that do not start with `n`.

6.3 Output Variables for Analyses

The types of outputs generated are different for different types of simulation. The following sections describe the output variables for the standard OP, DC, AC, Transient, Distortion, Pole-Zero, NET, TF, and Noise analyses.

6.3.1 .OP, .DC, .AC and .TRAN Analyses

OP, DC, AC and Transient analyses generate the following types of output:

- OP analysis generates real scalars (vectors of length one).
- DC analysis generates real vectors with a dimension that depends on `.DC` statement arguments, and with a variable specified on the `.DC` statement as a scale.
- AC and network analyses generate complex vectors with a dimension that depends on a specified frequency range, with frequency as a scale.
- Transient analysis generates real vectors with a dimension that depends on a specified time range, with time as a scale.

6.3.2 .DISTO Analysis

If `f2overf1` (see Section ".DISTO (Distortion Analysis)" in Chapter 3 Statements) is not specified, the `.DISTO` statement performs harmonic analysis and produces two `DISTO` plots: one for harmonic frequency *2f1*, and one for harmonic frequency *3f1*.

Both plots contain AC node voltages and currents through independent voltage sources with frequency as a scale. Vectors are complex and their dimension depends on specified frequency ranges.

If *f2overf1* is specified, the `.DISTO` statement performs a spectral analysis, and produces three plots for intermodulation product frequencies: *f1+f2, f1-f2* and *2f1-f2*. All three plots contain node voltages and currents through independent voltage sources with frequency as a scale, and all vectors are complex.

Examples

If small-signal distortion analysis is specified using the statement:

```
.DISTO DEC 10 1K 10MEG  ;  f2overf1 not specified
```

then the statement:

```
.PRINT DISTO V(4)
```

prints the voltage at node `4` at harmonic frequencies *2f1* and *3f1* (two printouts), where *f1* is the frequency swept from `1K` to `10MEG`, 10 points per decade.

If small-signal distortion analysis is specified using the statement:

```
.DISTO DEC 10 1K 10MEG 0.9 ;  f2overf1 specified
```

then the statement:

```
.PRINT DISTO V(4)
```

prints the voltage at node `4` at intermodulation product frequencies *f1+f2, f1-f2,* and *2f1-f2* (three printouts), where *f1* is the frequency swept from `1K` to `10MEG`, 10 points per decade, and *f2* is kept fixed at the value *f2=f2overf1 fstart=0.91K=0.9kHz*.

6.3.3 .PZ Analysis

The Pole-zero analysis generates poles and/or zeros of transfer functions as the output variables. All outputs are complex scalars (vectors of length one) with types pole for poles, and zero for zeros. They are named `pole(1), pole(2), ...`, and `zero(1), zero(2), ....`

Examples

The statement:

```
.pz 1 0 3 0 vol pz
```

calculates poles and zeros of transfer function *v(3)/v(1)*. To print all poles and zeros, issue the statement:

```
.print pz all
```

or, to print the first two poles and first two zeros, the following statement is used:

```
.print pz pole(1) pole(2) zero(1) zero(2)
```

6.3.4 .NET (Two Ports) Analysis

The small-signal network analysis generates 16 complex output vector variables:

```
s11,  s12,  s21,  s22    (s-parameters, type: s-param)
h11,  h12,  h21,  h22    (h-parameters, type: h-param)
z11,  z12,  z21,  z22    (z-parameters, type: z-param)
y11,  y12,  y21,  y22    (y-parameters, type: y-param)
```

with frequency as a scale. The dimensions of the vectors depend on the specified frequency range.

Examples

To print/plot results of network parameter calculation specified by the statement:

```
.net v(9,11) vin dec 10 1k 1G
```

the following statements can be used:

```
.print net all
.print net s11 h12 z22
.print net mag(s12) ph(y22)
.plot net mag(z12) mag(z22)
```

6.3.5 .TF Analysis

Transfer-function analysis generates three output variables: transfer function, input resistance and output resistance. All variables are real scalars (vectors of length 1) of the type *notype*.

Examples

The `tf` analysis specified by:

```
.tf v(5,3) vin
```

produces the following output variables:

- `tf` (transfer function)
- `zin_at_vin` (input resistance at `vin`)
- `zout_at_5_3` (output resistance across the nodes `5` and `3`)

To print the results, one of the following two statements can be used:

```
.print tf all
.print tf tf zin_at_vin zout_at_5_3
```

The TF analysis specified by the statement:

```
.tf v(4) iin
```

produces the following output variables:

- `tf` (transfer function)
- `zin_at_iin` (input resistance at `iin`)
- `zout_at_4` (output resistance at node 4)

The following statement can be used to print the output variables:

```
.print tf tf zin_at_iin zout_at_4
```

6.3.6 .NOISE Analysis

The `.NOISE` statement produces two plots:

- `NOISES`: For the noise spectral density curves.
- `NOISET`: For total integrated noise over the specified frequency range.

The plot `NOISES` contains at least three real vectors:

- `frequency` of type frequency for a scale.
- `onoise_s` of type onoise-spectrum for total output noise spectral densities.
- `inoise_s` of type inoise-spectrum for equivalent input noise.

The length of the vectors depends on the specified frequency range, and on the parameter `INTER` (see Section ".NOISE (Noise Analysis)").

The plot `NOISET` contains at least three scalars (vectors of length 1):

- `number` of type notype for scale.
- `onoise_t` of type onoise-integrated for total integrated output noise.
- `inoise_t` of type inoise-integrated for integrated equivalent input noise.

If the parameter `INTER` is specified in the `.NOISE` statement, then both plots contain a number of additional vectors and scalars, representing the contribution of each device noise source to the total output or integrated noise. The vector/scalar names for passive devices (resistors, voltage-controlled switches, and current-controlled switches) are created by:

```
onoise_s.devicename, onoise_t.devicename, inoise_t.devicename
```

The vector/scalar name for active devices consist of the additional suffix `noisetype`:

```
onoise_s.devicename.noisetype,
onoise_t.devicename.noisetype,
inoise_t.devicename.noisetype.
```

If parameter `RX <=val>` is specified in the `.NOISE` statement, the `NOISES` plots contain a number of the additional vectors, representing transfer function coefficients to the output for the corresponding noise source in the circuit. The vector names for all the devices are created by the same rule, but the prefix `RX.` is used instead of `onoise_s.`:

```
RX.devicename,  RX.devicename.noisetype.
```

value of noisetype is defined in the table below.

Table 6-3 Noise Types

Device	Noise Type	Meaning	RX = val
BJT	ib	Noise due to ib	1
	1overf	Flicker noise (1/f)	1
	ic	Noise due to ic	2
	rb	Noise due to rb	3
	rc	Noise due to rc	4
	re	Noise due to re	5
DIODE	id	Noise due to id	1
	1overf	Flicker noise (1/f)	1
	rs	Noise due to rs	2
JFET MESFET MOSFET	id	Noise due to id	1
	1overf	Flicker noise (1/f)	1
	rd	Noise due to rd	2
	rs	Noise due to rs	3

All vectors in the NOISES plot are sorted in alphabetical order. The scalars in the NOISET plot are sorted in alphabetical order, and then integrated equivalent input and output noises are listed in decreasing order.

Examples

The statement:

```
.NOISE v(5,3) iin dec 10 1 10g
```

does not specify INTER. Generated plots will contain the following vectors and scalars:

```
NOISES: frequency, inoise_s_i, onoise_s
NOISET: inoise_t_i, number, onoise_t
```

If INTER is specified in the statement:

```
.NOISE v(2) v1 dec 10 1k 100Meg 10
```

then the two generated plots will contain many vectors. The number of vectors depends on the number of passive and active devices in the circuit.

The NOISES plot contains the vectors onoise_s and inoise_s_v, representing total output and equivalent input noise;

- If the circuit contains the resistor rc1, then the plot will contain the vector onoise_s.rc1.
- If the circuit contains the bipolar transistor q1, the plot will contain the following vectors:

```
onoise_s.q1
onoise_s.q1.1overf
onoise_s.q1.ib
```

```
onoise_s.q1.ic
onoise_s.q1.rb
onoise_s.q1.rc
onoise_s.q1.re
```

The NOISET plot contains the scalars onoise_t and inoise_t_v, representing the total integrated output and equivalent input noise:

- If the circuit contains the resistor rc1, the plot will contain the scalars onoise_t.rc1 and inoise_t.rc1.
- If the circuit contains the bipolar transistor q1, the plot will contain the following vectors:

```
onoise_t.q1
onoise_t.q1.1overf
onoise_t.q1.ib
onoise_t.q1.ic
onoise_t.q1.rb
onoise_t.q1.rc
onoise_t.q1.re
```

and

```
inoise_t.q1
inoise_t.q1.1overf
inoise_t.q1.ib
inoise_t.q1.ic
inoise_t.q1.rb
inoise_t.q1.rc
inoise_t.q1.re
```

If INTER and RX=5 are specified in the statement:

```
.NOISE v(2) v1 dec 10 1k 100Meg 10 RX=5
```

the NOISES plot will contain the vectors onoise_s and inoise_s_v, representing total output and equivalent input noise:

- If the circuit contains the resistor rload, then the plot will contain the vectors:

```
onoise_s.rload,
```

representing contribution of thermal noise due to resistor rload to the total output noise, and

```
RX.rload,
```

representing the transfer function coefficient to the output for noisy resistor rload.

- If the circuit contains the MOSFET mm1, the plot will contain the following:

```
onoise_s.mm1
onoise_s.mm1.1overf
onoise_s.mm1.id
onoise_s.mm1.rd
onoise_s.mm1.rc,
```

representing contributions of all the internal transistor noise sources to the total output noise, and vectors

```
RX.mm1.id
RX.mm1.rd
RX.mm1.rs,
```

representing the transfer function to the output coefficients for the corresponding noise source in the transistor.

6.3.7 Transient Noise Analysis

The Transient noise analysis calculates the noise correlation matrix of circuit variables (voltages at the nodes, which are connected to capacitors) *K(t)* at each time point during the transient process:

$$K(t) = \varepsilon\left[x_{noise}(t) \times x_{noise}(t)^{T}\right]$$

Correlation matrix element $K_{i,j}(t)$ is the noise voltage variance (mean-squared noise power) or the noise voltage correlations between *i* and *j* noise node voltages. The `.PRINT`, `.PLOT`, `.IPRINT`, `.IPLOT` and `.MEASURE` statements can be used to generate, print, plot and measure noise voltage variance vectors.

Noise voltage variances (or noise voltage correlations) between nodes `nodename1` and `nodename2` are specified by:

```
k(nodename1, nodename2)
```

Noise voltage auto-correlation (correlation matrix diagonal elements $K_{i,i}(t)$) at node with name `nodename` is specified by:

```
k(nodename)
```

or

```
k(nodename, nodename)
```

Examples

`k(2)`	Noise voltage variance at node `2`.
`k(out)`	Noise voltage variance at node `out`.
`k(4,5)`	Noise voltage variance between nodes `4` and `5`.

6.4 Verilog-A Port I/V Access

Verilog-A devices are treated differently to instantiated device elements or subcircuits containing primitive device elements. To get currents flowing into a Verilog-A module through the electrical port p in SmartSpice, the following special syntax should be used.

Variant 1 - More Common

a) `@YVLGflatname[I(<p>)]`

where:

YVLGflatname	The instance `flatname` (e.g., `YVLG.XCAP.YVLG_co`)
I(< ... >)	Is used to access the current flowing into the module through the electrical port p.
p	port name (like used in module definition).

Example 1:

```
@yvlg.xcap.yvlg_c0[I(<p>)]
```

b) `@YVLGflatname[I(p,n)]`

where:

(p,n)	The branch from net/port p to net/port n.

This syntax allows the access to the branch current flowing between two nets or ports in the form `I(...)`.

Example 2: Where the module has two ports p and n, the following function accesses the branch current flowing between the ports p and n:

```
I(p,n)
```

Example 3:

```
@yvlg.xcap.yvlg_c0[I(p,n)]
```

Variant 2 - Less Common

a) `i(YVLGflatname.p)`

where:

YVLGflatname	Instance `flatname` (e.g., `YVLG.XCAP.YVLG_co`)
p	Port name (as used in module definition).

Example 4:

```
i(yvlg.xcap.yvlg_c0.p)
```

Note: The same syntax is used to get access to the potential on nets, ports or branches by placing V (instead of I) as the access function in the case of an electrical discipline.

6.5 Context-dependent Output Mechanism

The SmartSpice statements .PRINT, .IPRINT, UNIPRINT, .PLOT, .IPLOT, .UNIPLOT, .GRAPH, .PROBE, .PROBE/CSDF, .SAVE, .UNSAVE, .SAVEBIAS, .FOUR, .FFT, .LET, .MEASURE can be performed in two different ways:

- To all analyses contained in an input deck, independently on a position, and a specified type of analysis. This is the default behavior.
- Context-dependently: The statement performance depends on the statement position, specified type of analysis, and structure of the “statement flow”. This SmartSpice behavior set by the .OPTIONS CONTEXTCMDS statement, or by command-line flag ‘-hspice’.

Context-dependent output mechanism (when input deck contains the .OPTIONS CONTEXTCMDS statement) is illustrated in the following examples:

```
.AC DEC 10 10KHZ 10GHZ
.print v(andrpi)
.probe v(opout)
```

The .PRINT statement applies only to the .AC DEC 10 10KHZ 10GHZ analysis, and not to any other analyses in the input deck:

```
.dc VIN 0 5 0.1
.print v(9)
.probe v(3)
.probe v(andrpi)
.let ac vr5=vr(5) ;’ac’ indicates block of commands connected to DC
is over
.print v(7)
.print v(opout)
```

The three commands after the DC analysis (.PRINT, .PROBE, .PROBE) will be performed only for that analysis. The .LET statement will apply to all AC analyses in the input deck. The remaining two commands (two .PRINT statements) will be performed for all analyses in the deck. It is the presence of the .LET statement with its ’AC’ marker that interrupts the ’flow’, and indicates that the block of commands associated with the .DC analysis is broken:

```
.dc VIN 0 5 0.1
.print v(9)
.probe v(3)
.probe v(andrpi)
.let dc v5=v(5)   ;  ’dc’ is same as last analysis type, block is
unbroken
.print v(7)
.print v(opout)
```

Here, all the .PRINT and .PROBE statements will be performed for the DC analysis. There is no break in continuity since the .LET statement has a ’DC’ marker, which is the same as the analysis immediately preceding:

```
.dc VCC -12 0 1 ; analysis block starts here
.dc VIN 0 5 0.1 ; analysis
.TRAN 5NS 500NS ; analysis
.DISTO DEC 10 1k 10MEG ; analysis block ends here
.print v(9) ; block of post-processing starts here
.print v(7) ; block of post-processing ends here
```

Here, both .PRINT statements will be performed by all preceding analyses. In general, postprocessing statements which do not specify an analysis type are applied to the unbroken block of analysis statements that immediately precedes them:

```
.print ac v(1)  ;  free
.print v(9)   ;   free
.let asdf=v(opout)  ;  free
.print v(6)  ;  free
.dc VIN 5 10 1
.dc VCC -12 0 1
.print v(4)
```

All post-processing which does not specify any analysis type will be performed for all analyses in input deck. The statement .print v(4) applies only to both preceding DC analyses:

```
.fft v(pash) window =hamm ; free FFT will be attach to all
.probe fft vi(pash)
.AC DEC 10 10KHZ 10GHZ
.DC VIN -0.25 0.25 0.025
.probe v(2)
.TRAN 5NS 6NS
.probe tran v(6)
.fft v(7) window =hamm
.plot fft vr(opout)
.TRAN 5NS 6NS
.fft v(4) window =bart
.plot fft vi(9)
```

The first FFT statement (.fft v(pash) window =hamm) is a 'free' (un-associated) .FFT analysis and will be associated with both .TRAN statements, and also with the .PROBE fft vi(pash) statement. The second FFT statement (.fft v(7) window=hamm) is associated with the first .TRAN statement, and the third FFT statement is associated with the second .TRAN:

```
.four 1MEG 20 v(pash)
.probe four vi(pash)
.AC DEC 10 10KHZ 10GHZ
.DC VIN -0.25 0.25 0.025
.probe v(2)
.TRAN 5NS 1000NS
.probe tran v(6)
.four 1MEG 20 v(7)
.plot four vr(opout)
.TRAN 5NS 1010NS
.four 1MEG 20 v(4)
.plot four vi(9)
```

In this input deck fragment, the first .FOUR statement is un-associated, and is therefore applied to each and every analysis in the deck: the two .TRAN statements, and the .PROBE. The second and third .FOUR statements are associated with the .TRAN statements that immediately precede them.

Another thing to remember when using output statements is the relation between a statement itself and an analysis type. Here is an example:

```
.model nm nmos level=49

v1 d0 0 dc 5
v2 g 0 dc 0
r1 d0 d 1k
m1 d g 0 0 nm l=0.5u w=20u

.DC LIN V2 0 6 1

.measure dc aaa find id(m1) at=0
.print vvv1=v(g)
.print aaa
```

Both .PRINT statements will be performed for DC analysis. SmartSpice supports printing of measure results. Thus both .PRINTs will be performed for the .MEASURE plot as well. As a result we will get couple of errors “no such vector as”.

The first .PRINT statement will search vvv1 vector in resulting DC plot and will print it successfully. But it will also search vector inside resulting MEASURE plot. Since there is no such vector inside MEASURE plot we'll receive an error.

The second .PRINT statement will process aaa vector in the same way. DC plot doesn't contain such a vector so it will generate an error. For MEASURE plot vector aaa will be printed.

To make a clear output and suppress those errors it is recommended to use .PRINT statements with the analysis specified. The following example gives the same results without errors:

```
.DC LIN V2 0 6 1

.measure dc aaa find id(m1) at=0
.print dc vvv1=v(g)
.print meas aaa
```

This behavior is common for all output statements.

6.6 Multi-Sweep Output Example

```
****************** MULTI_SWEEP_TRAN ********************
.param a=1
.param b=1
.param c=1.0e-12
v1 1 0 PULSE (0 3 0.1NS 0.1NS 10NS 15NS)
r1 1 OUT a
r2 0 OUT b
c1 0 OUT c
.data scale_c c 1.0e-12 2.0e-12 3.0e-12
.enddata
.data scale_b b 1000 2000 3000
.enddata
*********************************************************
.TRAN 1ns 10ns sweep a LIST 3 1000 1100 1200 sweep data=scale_b
+ sweep data=scale_c
.meas TRAN QWE CROSS V(OUT) VAL=1.0 NEST=0
.OPTIONS POST=1
.save all
.end
```

Example

```
Variables:       0 a notype
                 1 qwe notype
Values:
0                1.000000000000000e+03
                 6.059355050241965e-10
```

SmartSpice saves all nested swept parameters in `.measure` plot.

Example

```
Variables:       0  a     notype
                 1  b     notype
                 2  c     notype
                 3  qwe   notype
Values:
0                1.000000000000000e+03
                 1.000000000000000e+03
                 1.000000000000000e-12
                 6.059355050241965e-10
```

6.7 Measure Statement

The .MEASURE statement calculates circuit performance measures and their electrical specifications for output variables defined by the user. This statement is used to store measurement results and retrieve those results for future calculations. This feature makes the .MEASURE statement a powerful utility for parametric analysis. The .MEASURE statement is used for both function and performance measure circuit optimization with the usual .MODIF statement stop conditions or with Optimizer target specifications.

Numerical parameter values of the .MEASURE statement can be numbers, names of previous measurements, parameter labels, algebraic expressions of measurement names, labels, and numbers. The output variables can be specified with simple output variables (node voltages, currents, etc.), or with algebraic expressions of simple output variables and parameter labels. Expressions must be in single quotation marks.

Arguments must be monotonic for CROSS, WAVE, DELAY, AVG, RMS, DERIV, INTEGRAL, FIND and POINT measurements. Their corresponding measurement intervals must have at least two points (except for POINT measurement). A measurement performed on complex vectors will be applied to the magnitude of the vector (excluding the FIND measurement).

If an analysis statement contains the keyword SWEEP, SmartSpice creates a set of output vectors for each value of the second swept parameter. By default (NEST=0), the measurement is performed for all sets. If NEST is set to a value greater than 0, the measurement is performed only on the specified step.

If a nested DC sweep is performed using the .DC statement without the keyword SWEEP, SmartSpice creates one set of output vectors with many segments. In this case, the argument is non-monotonic. By default (NEST=0), this measurement is calculated and the results are printed for each segment of the sweep, and the final result is stored.

For MAX, MIN, AMAX, AMIN, ERR, ERR1, ERR2 and ERR3 measurements (if NEST is set to a value of -1) the measurement is calculated for the entire argument without dividing it into segments. It is also possible to calculate measurements for a single segment. Single-segment calculation is useful for model parameter optimization.

If the extended output format of any measurement statement (except RMS) consists of three lines, the first line contains the name of the measurement, and the names of the variables used in calculations. The second line contains the result value of the measurement, which is used later in other measurements. The number of digits of the result value is defined by the option NUMDGT in the .OPTIONS statement. The third line contains additional information, such as the argument value for the function maximum (MAX), or the number of points within the measurement interval (ERR, ERR1, ERR2, ERR3). The output of the RMS measurement contains only the first and second lines. By default, a more compact format is used. The extended format is used when the option FORMAT in the .OPTIONS statement is defined as FORMAT=0.

Note: If NOISE is specified, the corresponding measurement will be computed for the NOISES (spectral density) plot only.

6.7.1 Format of .MEASURE Statements

All .MEASURE statements begin with the following four words:

```
.MEASURE analysis_type resname meas_type <print=val>
```

where:

- analysis_type is one of AC, DC, FFT, FOUR, NET, NOISE, PZ or TRAN.
- resname is the name of the result of the measurement, and may be used for references and output.
- meas_type is the type of measurement to be performed.
- print=val:

  print=0 does not print the measure result into the measure raw file.

  print=1 (default) prints the measure result into the measure raw file.

What follows these initial words varies according to meas_type, but certain keywords are allowed with all measurements:

NEST	Works for nested sweeps • **NEST=0:** Default value. SmartSpice computes the measurement for each value of the second sweep parameter and stores the last result. • **NEST=step:** SmartSpice calculates the measurement for the specified step of the second sweep. The value of step must not be greater than the number of steps in the sweep.
OFF	Suppresses result printing

The following keywords are allowed only with certain types of measurement, as shown in Table 6-4. They have the same meaning for all measurement types that accept them.

FROM	Argument value where measurement begins. Defines the first point of the user-specified interval. The default value is the first point of the analysis interval. (Allowed with all but EXPR/PARAM measurements).
TO	Argument value where the measurement ends. Defines the last point of the user-specified interval. The default value is the last point of the analysis interval. (Allowed with all but EXPR/PARAM measurements).
RISE	Specifies which rising edge of the vector must be used for the measure calculation. A zero value is used to measure the last rising edge of the vector.
FALL	Specifies which falling edge of the vector must be used for the measure calculation. A zero value is used to measure the last falling edge of the vector.
CROSS	Specifies which crossing point must be used for the measure calculation. A zero value is used to measure the last crossing point.

Finally, there are those keywords whose precise meaning depends on the particular measurement type with which they are used:

ARG0 **VAL1** **ARG1** **COEF** **TARG** **DIFF**	Allowed only with DELAY.

See the specific keywords for details on how these keywords should be used.

Many keywords may be specified with either a numerical value or a symbolic parameter name. An example is the FROM keyword in the following statement:

```
.MEASURE TRAN avcur_rload AVG i(rload) FROM=120n
```

or

```
.MEASURE TRAN avcur_rload AVG i(rload) FROM='2.7*t1'
```

which measures the average current at the node rload during a transient analysis starting at the point specified by the FROM keyword. In the first example, the measurement begins at *time=120n*. In the second case, the measurement begins at the time obtained by evaluating the expression *'2.7*t1'*. Allowed variables in the expression are the names of previously calculated measurements, parameter labels, or an algebraic expression of measurement names, labels, and numbers. Expressions must be put in single quotation marks.

Table 6-4 Keywords Allowed with .MEASURE Statements

Measurement Type	NEST	OFF	FROM	TO	OCCUR	RISE	FALL	CROSS	VAL	VAL0	VAL1	ARG0	ARG1	COEF	TARG	DIFF	AT	WHEN	TD
AMAX	•	•	•	•															
AMIN	•	•	•	•															
AVG (MEAN	•	•	•	•															
CROSS (WHEN)	•	•	•	•	•	•	•		•										•
DELAY (TRIG)	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•		•
DERIVATIVE (DERIV)	•	•	•	•	•					•		•							
ERR, ERR1, ERR2, ERR3	•	•	•	•					•										
EXPR (PARAM)	•	•																	
INTEGRAL (INTEG)	•	•	•	•															
FIND	•	•	•	•		•	•	•									•	•	
MAX	•	•	•	•															
MIN	•	•	•	•															
POINT	•	•	•	•								•							
PP	•	•	•	•															
RMS	•	•	•	•															
WAVE	•	•	•	•		•	•			•	•	•	•	•					

6.7.2 AVG (MEAN), RMS, PP, INTEGRAL (INTEG)

Syntax

```
.MEASURE analysis_type resname AVG <POS> <NEG> |MEAN <POS> <NEG>
|RMS|PP|INTEGRAL|INTEG outvar
+ <FROM=val|name> <TO=val|name>
+ <OFF> <NEST=0|step>
```

outvar1	Name of output variable.
AVG <POS> <NEG> | MEAN <POS> <NEG>	(Average) Computes the area under the variable outvar divided by the length of the user-specified time interval. Flag POS or NEG should be specified in case of calculation the average of the positive or negative part of the output variable.
RMS	(Root Mean Square) Computes the square root of the area under the variable outvar divided by the length of the user-specified time interval.
PP	(Peak to Peak) Computes the maximum value minus the minimum value of the variable outvar within the user-specified time interval.
INTEGRAL	(Integration) Computes the integral of the variable outvar1 within the user specified time.

Examples

```
MEASURE TRAN AVGPOS_V1 AVG POS V(1) from 10n to 10u
```

This measure computes the average of the positive part of the output variable V(1) during the time interval from 10n to 10u.

```
.MEASURE AC PP_VM1 PP VM(1)
```

This example computes the difference between the maximum and minimum values of the magnitude VM(1) during the frequency interval from 10MHz to 100GHz (see Figure 6-1).

```
PP_VM1 = MAX - MIN
```

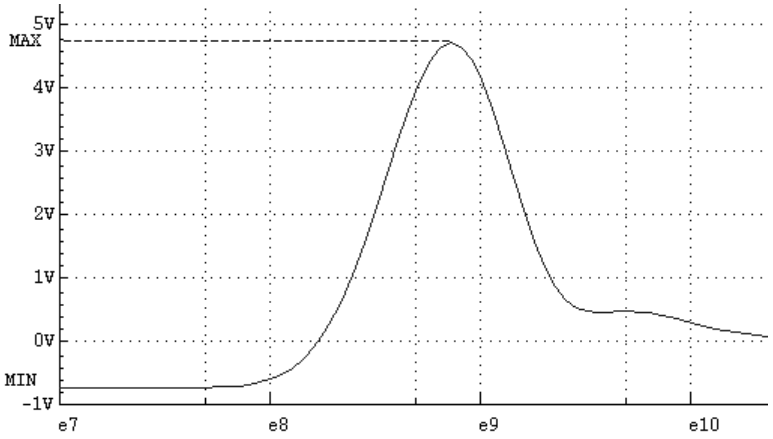


Figure 6-1 PP Calculations

6.7.3 CROSS (WHEN)

Note: The keyword WHEN may be used as a synonym for CROSS.

Calculates the argument value at the point of the intersection of outvar1 and outvar2, or outvar1 and VAL during the specified measurement interval.

Syntax

```
.MEASURE analysis_type resname
+ CROSS outvar1 outvar2 | CROSS outvar1 VAL=val|name
+ <TD=number>
+ <FROM=val|name> <TO=val|name>
+ <OCCUR=number | RISE=number | FALL=number>
+ <OFF> <NEST=0|step>
```

or

```
.MEASURE analysis_type resname
+ WHEN outvar1=outvar2 | WHEN outvar1=val|name
+ <TD=number>
+ <FROM=val|name> <TO=val|name>
+ <OCCUR|CROSS=number | RISE=number | FALL=number>
+ <OFF> <NEST=0|step>
```

outvar1, outvar2	Names of output variables.
VAL	Calculates the cross of outvar1 and VAL (instead of outvar2).
OCCUR, RISE, FALL	Specifies which cross of outvar1 and outvar2, or outvar1 and VAL must be calculated. A zero value is used to measure the last crossing point. If neither RISE, FALL, CROSS or OCCUR keywords are specified in the .MEASURE statement (CROSS, FIND, TRIG/TARG) then CROSS or OCCUR is used by default. Default is 1.

Examples

```
.MEASURE TRAN CROSS_1_2 CROSS V(1) V(2) OCCUR=2
.MEASURE TRAN CROSS_2_VAL CROSS V(2) VAL=1.5
```

In the first example, SmartSpice finds the second cross point of the output variables V(1) and V(2), and stores the calculated time value T0 under the name CROSS_1_2 (see Figure 6-2).

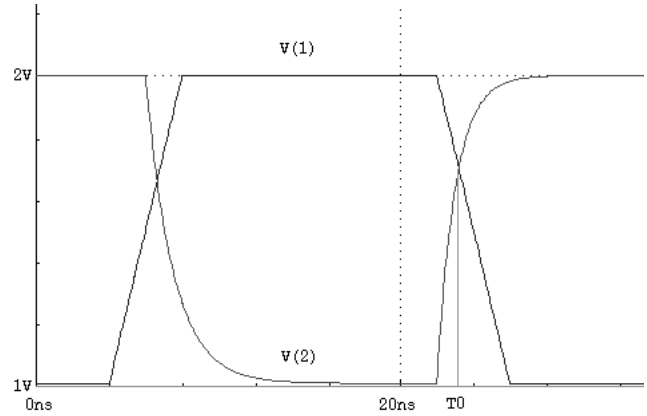


Figure 6-2 Cross Calculation

In the second example, SmartSpice calculates the first time point where the node voltage V(2) reaches a value of 1.5 (see Figure 6-3).

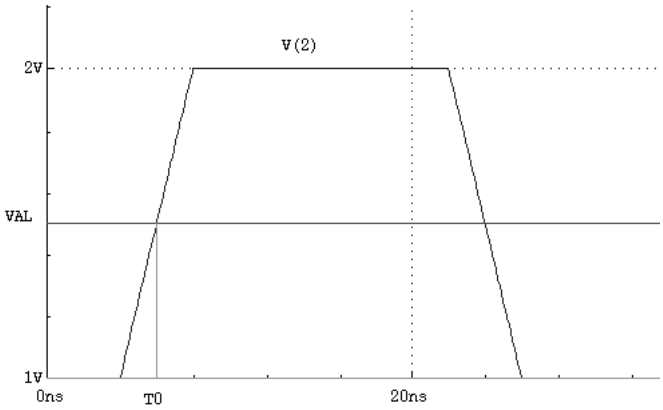


Figure 6-3 Cross Calculation

6.7.4 DELAY (TRIG)

Note: The keyword TRIG may be used as a synonym for DELAY.

DELAY calculates the propagation delay between the middle points of the rise or fall of outvar2, and the rise or fall of outvar1 during the specified measurement interval. The middle points can either be specified directly using VAL, or calculated as a function of the low and high values on either side of the middle point.

Syntax

```
.MEASURE analysis_type resname
*** Trigger section
+ DELAY|TRIG outvar1 <FROM=val|name> <TO=val|name>
+ <COEF=val|name> <DIFF=val|name> <TD=number>
+ RISE=number | FALL=number | CROSS=number   <VAL=val|name> |
+ <<<ARG0=val|name> <VAL0=val|name>> | <<ARG1=val|name> |
+ <VAL1=val|name>>> + <AT=val|name>
*** Target section
+ TARG outvar2 | TARG=outvar2
```

```
+ <TD=number>
+ RISE=number | FALL=number |CROSS=number
+ <COEF=val|name> <DIFF=val|name> <VAL=val|name> |
+ <<<ARG0=val|name> | <VAL0=val|name>
+ <<ARG1=val|name> | <VAL1=val|name>>>
+ <OFF> <NEST=0|step>
```

Note: RISE, FALL or CROSS specifications may appear after the arguments and values (ARG0, VAL0, ARG1 and VAL1).

outvar1, outvar2	Names of output variables.
VAL	The values of outvar1 and outvar2 where the counter for rises and falls during delay calculation is incremented by one.
VAL0, VAL1	The values of outvar1 and outvar2 that are used as levels for either the rise or the fall calculations (instead of VAL).
ARG0, ARG1	Argument values that correspond to VAL0 and VAL1 that can be given instead of VAL, VAL0 and VAL1. ARG0 and ARG1 must be within the specified measurement interval.
COEF	Actual levels for either the rise or fall calculations are $VAL0 + COEF(VAL1 - VAL0)$ instead of VAL0 and $VAL1 - COEF(VAL1 - VAL0)$ instead of VAL1. COEF must be greater than or equal to 0 and less than 0.5. Default is 0.1.
DIFF	The absolute value of the difference between VAL0 and VAL1 must be greater than DIFF. Default is 0.0.
AT	Start point of measurement. If the parameter AT is specified, outvar1 must not be used.
TARG	Indicates where information about the second variable begins.
OCCUR, RISE, FALL	Specifies which cross of outvar1 and outvar2, or outvar1 and VAL must be calculated. A zero value is used to measure the last crossing point. If neither RISE, FALL, CROSS or OCCUR keywords are specified in the .MEASURE statement (CROSS, FIND, TRIG/TARG) then CROSS or OCCUR is used by default. Default is 1.

TD=number	Amount of simulation time that must elapse before the measurement is enabled. The number of crossings, rises or falls is counted only after the TD value. Default is 0. When the parameter TD is specified in TRIG, and is not specified in the TARG section of the measurement statement, the TRIG TD value will be used for both the TRIG and TARG functions when counting crossing points.

Examples

```
VEE 1 0 PULSE (1.05 1.65 2n 3n 3n 9n 40n)
*
.MEASURE TRAN DEL_1_2 DELAY
+ V(1) RISE=1 VAL=1.35 TARG=V(2) RISE=1 VAL0=1.7 VAL1=2.3
*
.MEASURE TRAN DEL_1_2 DELAY
+ V(1) RISE=1 ARG0=0 ARG1=12NS TARG=V(2) RISE=1 VAL0=MIN_V2
+ VAL1=MAX_V2
```

In the first example, SmartSpice calculates the propagation delay of the first rise of the output variable V(2) with respect to the first rise of the input signal V(1) (see Figure 6-4).

```
T10 = 1.7 + 0.1 * (2.3 - 1.7)= 1.76
T11 = 2.3 - 0.1 * (2.3 - 1.7)= 2.24
DEL_1_2 = ((T11 + T10)/2) - T00
```

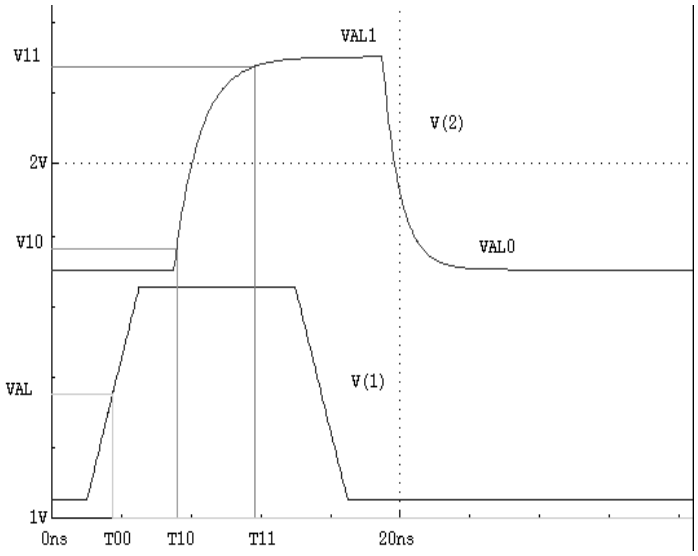


Figure 6-4 Delay Calculation

In the second example, SmartSpice computes the same delay using voltage values at 0 and 12ns for the input signal V(1), and previously calculated measures MIN_V2 and MAX_V2 for the output variable V(2).

6.7.5 DERIVATIVE (DERIV)

If ARG0 is specified, then the value of the derivative of outvar1 at the point ARG0 is calculated. Otherwise, SmartSpice calculates the value of the argument where the derivative equals VAL0. The specified measurement interval is considered.

Syntax

```
.MEASURE analysis_type resname
+ DERIVATIVE|DERIV outvar1
+ <FROM=val|name> <TO=val|name>
+ <VAL0=val|name <OCCUR=val|name>> | <ARG0=val|name>
+ <OFF> <NEST=0|step>
```

outvar1	Name of output variable.
VAL0	The derivative value for which the corresponding argument value is to be calculated.
OCCUR	Shows what argument value must be calculated. Default is 1.
ARG0	Argument value that determines where to calculate the derivative of outvar1.

6.7.6 ERR, ERR1, ERR2, ERR3

These functions compute various errors between the arguments specified.

Syntax

```
.MEASURE analysis_type resname ERR|ERR1|ERR2|ERR3 outvar1 outvar2
+ <FROM=val|name> <TO=val|name> <VAL=val|name> <OFF>
+ <NEST=-1|0|step>
```

outvar1, outvar2	Names of output variables.
VAL	If an outvar1 value is less than VAL, then the difference between outvar1 and outvar2 is divided by VAL instead of by outvar1. Default is *1E-12*.
NEST=-1	SmartSpice calculates the measurement for nested DC sweep without dividing the entire argument into segments of the sweep.
ERR	Calculates the square root of the sum of the squares: $$ERR = \left(\frac{1}{N} \sum_{i=1}^{N} \left(\frac{M_i - C_i}{\max(VAL, M_i)} \right)^2 \right)^{0.5}$$ • M_i is the ith measurement value (outvar1). • C_i is the corresponding calculated (outvar2) value. • N is the number of points within the specified measurement interval.

ERR1	Calculates the average relative error: $$ERR1 = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{M_i - C_i}{\max(VAL, M_i)}\right)$$
ERR2	The average absolute relative error: $$ERR2 = \frac{1}{N}\sum_{i=1}^{N}\left\lvert\frac{M_i - C_i}{\max(VAL, \lvert M_i\rvert)}\right\rvert$$
ERR3	The plus sign (+) is used for the positive M/C ratio, and the minus sign (-) for the negative M/C ratio. $$ERR3 = \frac{1}{N}\sum_{i=1}^{N}\frac{\pm\log\left\lvert\frac{M_i}{C_i}\right\rvert}{\left\lvert\log(\max(VAL, \lvert M_i\rvert))\right\rvert}$$

Examples

```
VGOAL3 VGOAL3 0 PWLDC (-0.25v 12 -0.2v 11.97 -0.15v 11.9 -0.1v 11.3
-0.05v 8.2 0v
+ 5.6 0.05v 2.1 0.1v 0.8 0.15v 0.4 2v 0.31 2.5v 0.3)
*
.MEASURE DC ERR_V3 ERR2 V(VGOAL3) V(3) VAL=1
```

In this example, SmartSpice computes a value ERR2, which is the difference between the DC transfer curve V(3) and V(VGOAL3) (see Figure 6-5).

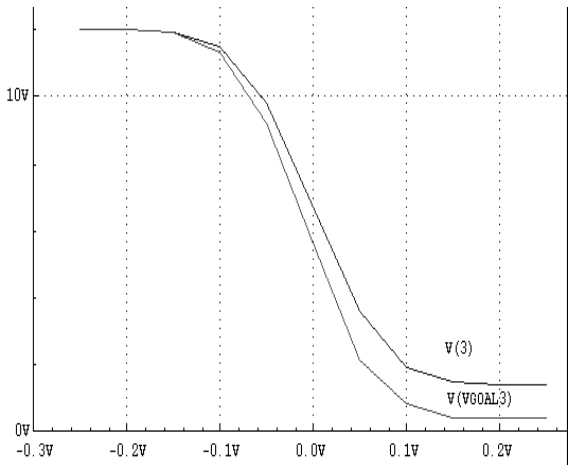


Figure 6-5 ERR Calculation

6.7.7 EXPR (PARAM)

The keyword PARAM may be used as a synonym for EXPR.

This statement computes the value of VAL. It provides a convenient way to calculate expressions of any complexity in order to use them in other measurements and in the .MODIF statement.

The PARAM measurement statement supports the use of ternary operators in expressions:

&& (and), || (or), and ! (not).

Syntax

```
.MEASURE analysis_type resname
+ EXPR VAL=val|name | PARAM=val|name
+ <OFF>
```

VAL	Specifies the expression to be evaluated. It also can contain the device and/or model parameters as part of the expression.

SmartSpice supports PARAM/EXPR measurements with vector arguments (e.g., .MEASURE TEST PARAM='2*v(1)'). To evaluate such expressions, the last values from vector arguments are used.

To enable this feature, set the variable evaluatemeasureexprasvector to TRUE.

Examples

```
.MEASURE TRAN RSDEL DELAY V(IN) RISE=1 VAL=2.5 TARG=V(OUT) RISE=1
VAL0=0
+ VAL1=5
*
.MEASURE TRAN FLDEL DELAY V(IN) FALL=1 VAL=2.5 TARG=V(OUT) FALL=1
+ VAL0=0 VAL1=5
*
.MEASURE TRAN DELDIFF EXPR VAL="abs(fldel-rsdel)"
```

In these examples, SmartSpice calculates the absolute value of the difference between two propagation delays of the node voltages V(IN) and V(OUT).

```
.MEASURE TRAN Instance EXPR VAL='@R1[res]'
.MEASURE TRAN Model param='@RM1[tnom]'
```

In these examples, SmartSpice calculates the device parameter "resistance" for the resistor named R1, and the model parameter "nominal temperature" for the resistor model named RM1.

6.7.8 FIND

Calculates the value of outvar1 when outvar2 intersects outvar3, or when outvar2 intersects a constant value or a timepoint.

Syntax

```
.MEASURE analysis_type resname
+ FIND outvar1 WHEN outvar2=outvar3|val|name
+ <TD=val>
+ <RISE=number | FALL=number | CROSS=number> <FROM=val|name>
+ <TO=val|name>
+ <OFF>
+ <NEST=0|step>
```

or

```
.MEASURE analysis_type resname FIND outvar1 AT=val|name <OFF>
<NEST=0|step>
```

outvar1 **outvar2** **outvar3**	Names of output variables.
val	Calculates the cross of outvar2 and val (instead of outvar3).
AT	Indicates the argument value where outvar1 is to be calculated.
OCCUR, RISE, FALL	Specifies which cross of outvar1 and outvar2, or outvar1 and VAL must be calculated. A zero value is used to measure the last crossing point. If neither RISE, FALL, CROSS or OCCUR keywords are specified in the .MEASURE statement (CROSS, FIND, TRIG/TARG) then CROSS or OCCUR is used by default. Default is 1.
TD	Time at which Measurement starts.

Examples

```
.MEASURE TRAN FND_1 FIND V(1) WHEN V(2)=V(3)
+ RISE=2
*
.MEASURE TRAN FND_2 FIND V(OUT) WHEN V(IN)=2.5
+ FALL=1
*
.MEASURE TRAN FND_3 FIND V(5) AT=10ns
```

In the first example, SmartSpice calculates the point of the second intersection of V(2) and V(3) when V(2) is rising. The value of V(1) is calculated at that point. This measurement statement is equivalent to the combination of the following CROSS and POINT statements:

```
.MEASURE TRAN M1 CROSS V(2)=V(3) RISE=2
.MEASURE TRAN M2 POINT V(1) ARG0=M1
```

In the second example, SmartSpice computes the value of V(OUT) at the point where V(IN) crosses the value 2.5V for the first time, while decreasing.

In the third example, SmartSpice calculates the value of V(5) at 10ns.

6.7.9 MAX, MIN, AMAX, AMIN

Syntax

```
.MEASURE analysis_type resname MAX|MIN|AMAX|AMIN outvar1
+ <FROM=val|name>
+ <TO=val|name>
+ <OFF> <NEST=-1|0|step>
```

outvar1	Name of output variable.
NEST=-1	SmartSpice calculates the measurement for a nested DC sweep without dividing the entire argument into segments of the sweep.

MAX	Calculates the maximum value of outvar1 during the user-specified measurement interval.
MIN	Calculates the minimum value of outvar1 during the user-specified measurement interval.
AMAX	Calculates the argument value of the maximum of outvar1 during the user-specified measurement interval.
AMIN	Calculates the argument value of the minimum of outvar1 during the user-specified measurement interval.

6.7.10 POINT

Calculates the value of outvar1 at the point ARG0.

Syntax

```
.MEASURE analysis_type resname POINT outvar1 <FROM=val|name>
+ <TO=val|name>
+ ARG0=val|name <OFF> <NEST=0|step>
```

outvar1	Name of the output variable.
ARG0	Argument value that determines where to calculate the value of outvar1.

6.7.11 WAVE

It calculates the duration of the rise or fall of outvar1 over a specified interval. A rise is defined on the measurement interval as outvar1 changes from $VAL0 + COEF(VAL1 - VAL0)$ to $VAL1 - COEF(VAL1 - VAL0)$. A fall is defined on the measurement interval as outvar1 changes from $VAL1 - COEF(VAL1 - VAL0)$ to $VAL0 + COEF(VAL1 - VAL0)$. ARG0 and ARG1 can be used instead of VAL0 and VAL1. The units of the measurement interval are whatever outvar1 is scaled against (i.e., *x*-axis units).

Syntax

```
.MEASURE analysis_type resname WAVE outvar1 <FROM=val|name> |
+ <TO=val|name>
+ RISE=number | FALL=number <<ARG0=val|name> | <VAL0=val|name>>
+ <<ARG1=val|name> | VAL1=val|name>> <COEF=val|name>> <OFF>
+ <NEST=0|step>
```

outvar1	Name of the output variable.
VAL0, VAL1	The values of outvar1 and outvar2 that are used as levels for either the rise or the fall calculations.
ARG0, ARG1	Argument values which correspond to VAL0 and VAL1 values that could be given instead of VAL0, VAL1. ARG0 and ARG1 must be within the specified measurement interval.

COEF	Proportion of the difference between VAL0 and VAL1 over which to calculate the measurement interval. COEF must be greater than or equal to 0, and less than 0.5. Default is 0.1.

Examples

```
.MEASURE TRAN RISE_V1 WAVE V(1) RISE=1 VAL0=1 VAL1=2
.MEASURE TRAN RISE_V1 WAVE V(1) RISE=1 ARG0=0 ARG1=12NS
.MEASURE TRAN MIN_V1 MIN V(1)
.MEASURE TRAN MAX_V1 MAX V(1)
.MEASURE TRAN FALL_V1 WAVE V(1)
+  FALL=1 VAL0=MIN_V1 VAL1=MAX_V1
+  FROM=10NS TO=20NS
```

In the first example, the time of the first rise from 1V to 2V of the node voltage V(1) is calculated (Figure 6-6). The default value of COEF is used.

```
V0 = 1 + 0.1 * (2 - 1)= 1.1
V1 = 2 - 0.1 * (2 - 1)= 1.9
RISE_V1 = T1 - T0
```

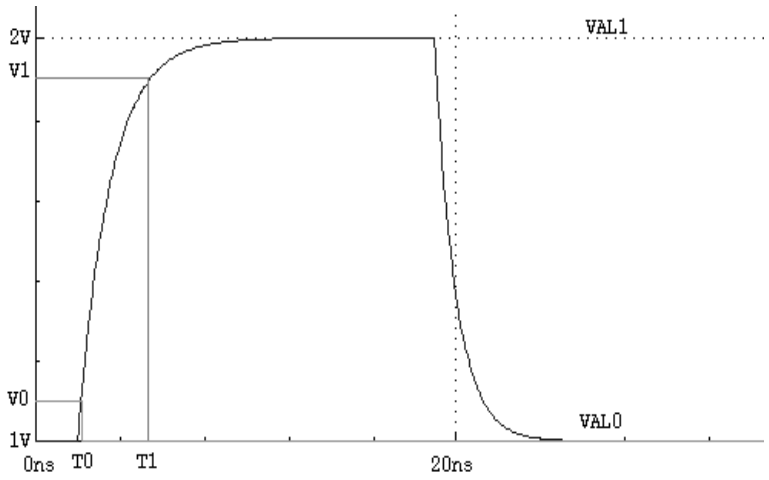


Figure 6-6 Wave Calculation

In the second example, voltages at 0 and 12ns are used to calculate the rise time of V(1).

The third and fourth examples define MIN_V1 and MAX_V1 for the fifth example.

The fifth example computes the time of the first fall during the time interval from 10ns to 20ns. It uses the previously calculated measures MIN_V1 and MAX_V1.

6.7.12 Saving Measure Statement Results

By default, the program immediately prints all results of the .MEASURE statement and will save all results as vectors in a separate plot. This plot will have type meas and a name prefixed with meas.

In batch mode, the program creates one measure plot to store all .MEASURE results. In interactive mode, a new measure plot is created each time the run command is issued. This means that vectors in a measure plot will have dimensions equal to the number of times the corresponding .MEASURE statement is executed. If the input deck contains .MODIF, .ST,

.TEMP or SWEEP statements, the vectors in a measure plot will be multidimensional. For simulations with only one pass, vectors in the measure plot will be one dimensional.

If the input deck contains .MODIF, .ST, .TEMP or SWEEP statements, the measure plot will contain additional vectors. If the input deck contains .ST or .TEMP statements, one additional vector will be created by saving the values of the sweep variable of .ST or .TEMP statements. This vector will be the scale of the measure plot. If the input deck contains a .MODIF statement, one additional vector is created for each variable that is swept through the .MODIF statement, and the first of these vectors becomes the scale of the measure plot.

The vectors from the measure plot are printed or plotted like the vectors from any other plot. In batch mode, they are referenced using the keyword MEAS for the analysis type in .PRINT and .PLOT statements:

```
.PRINT MEAS vecname1 ...
.PLOT MEAS vecname1 ...
```

In interactive mode, if the current plot is set to a measure plot, these vectors can be referenced directly:

```
print vecname1 ...
plot vecname1 ...
```

or, if the measure plot is not the current plot, the vectors can be referenced using the notation:

```
print plotname.vecname1 ...
plot plotname.vecname1 ...
```

where plotname is the name of the measure plot (for example: meas1, meas6, ...).

To control .MEASURE output, the variable splitmeasureplots can be used. splitmeasureplots=FALSE (default) saves all .MEASURE results for .MODIF/.ST/.MC analyses in a single plot. splitmeasureplots=TRUE saves all .MEASURE results for different analyses in separated plots. Saving all of the results of the .MEASURE statement is controlled by the .OPTIONS control statement .OPTIONS SAVEMEASURES and set command (or variable) savemeasures. .OPTIONS SAVEMEASURE=0 allows you to turn off the option of saving all results of the .MEASURE statement.

Examples

Consider the following input deck:

```
*save measures example
VIN 1 0 SIN(0 0.1 5MEG)
vcc 1 11 dc 2
r1 11 0 1 rmod
.model rmod r (tc1=2 tc2=0.005)
.MODIF TEMP=-50 vcc(dc)=4 all@r(res)=10k
+ MODIF LOOP=1
+ vcc(DC)+=0.2 proff MODIF TEMP=0 vcc(dc)=4
+ all@r(res)=6k
+ MODIF LOOP=1 vcc(DC)=0.2 proff
+ MODIF TEMP=27 vcc(dc)=4
+ all@r(res)=1k
+ MODIF LOOP=1 vcc(DC)=0.2 proff
*
.MEASURE TRAN maxVIN MAX i(r1)
*
.TRAN 5n 500n CALLV SAVEV
*
```

```
.print tran V(1)
.plot meas maxvin vs vcc(dc)
.END
```

Simulating this input deck generates six `tran` plots (`tran1, ..., tran6`). The program will also create the plot `meas1`, with the following vectors:

```
ALL@r(res) : notype, real, 6 long
maxvin : notype, real, 6 long
temp : notype, real, 6 long [scale]
vcc(dc) : notype, real, 6 long
```

The vectors have dimension 6. Only one vector (`maxvin`) is the result of saving the `.MEASURE` statement, and the other three vectors (`temp`, `vcc(dc)` and `all@r(res))` are generated as sweep variables from the `.MODIF` statement.

6.8 Output Files

SmartSpice produces output files:

Table 6-5 SmartSpice output files and suffixes

Output file type	Extension
Output listing	`.out` or user specified
Analysis results	`.raw`, `.dat`, `.fsdb`
Analysis measurement results	`.measure.raw#`
Subcircuit node map	`subcircuitnodemap.xml`
Hardcopy graph data(postscipt format)	`hc#####.tmp#`
LSTB analysis results	`.cx#`

Special symbol # is a sweep number or hardcopy file number.

Output listing are placed in `inputname.out` file, or in a file with a user-specified file extension. This file includes the following information:

- name and version of simulator used
- license details
- input file name
- copy of the input deck file
- model parameters
- results of the `.OP` statement
- low resolution plots originating from the `.PLOT` statement
- results of `.PRINT` statements
- results of `.MEASURE` statements
- results of `.OPTIONS` statements
- runtime statistics

Analysis results are written to `inputname.raw` (rawfile). This file contains the results that are produced by all input deck analysis statements, and the associated post-processor commands that will be needed. Rawfile contains one or more (in sweep case) header/data segments, which are presented in ASCII/binary format. Header is always in an ASCII format.

The rawfile segment structure is presented in Table 6-6. SmartSpice rawfile format is based on the Berkeley SPICE rawfile format. For more details, see the Berkeley SPICE 3 source code.

LSTB analysis results are written to `inputname.cx#` file. The `*.cx#` file is a general HSPICE file format for the complex outputs.

Table 6-6 Rawfile Segment Structure

Field number	Field type	Keyword and Data Content	Mode
1	header	**Title:** `netlist_name\n`	ASCII
2	header	**Inputdeck file name:** `path\file_name\n`	ASCII
3	header	**Date:** `date_time\n`	ASCII
4	header	**Plotname:** `performed_analysis, temperature step name_swept_parameter = value\n`	ASCII
5	header	**Temperature:** `temp_value\n`	ASCII
6	header	**Sweepvar:** `name_var; secondary_var\n`	ASCII
7	header	**Sweepmode:** `sweepmode\n`	ASCII
8	header	**Flags:** `type padding\n`	ASCII
9	header	**No. Variables:** `nvars\n`	ASCII
10	header	**No. Points:** `npoints\n`	ASCII
11	header	**Dimensions:** `size_1, size_2, ..., size_8\n`	ASCII
12	header	**Source:** `name_of_simulator\n`	ASCII
13	header	**Version:** `number_of_ version\n`	ASCII
14	header	**Variables:** `list_of_variables\n`	ASCII
15	header	**Binary:** `binary_type \n` or **Value:** `\n` (for `.OPTIONS POST=2`)	ASCII
16	data	simulation results	ASCII or binary

where:

`netlist_name`	First netlist string.
`path\file_name`	The name of the file contained in netlist.
`data_time`	Data and time information (e.g. Wed Sep. 12 15:12:05 2001).
`performed_analysis`	The name of the analysis currently performed (e.g. Transient analysis).

temperature	Temperature during simulation (e.g. 25 °C).
name_swept_parameter	Swept parameter name.
value	Current swept parameter value.
temp_value	Temperature during simulation (e.g. 25 °C).
name_var	Variable name to be swept during simulation (e.g. time for Transient analysis, frequency for AC analysis).
secondary_var	Name of the second (nested) sweep parameter, start, stop and increment values.
sweepmode	One of `0`, `-1`, `1(DECADE)`, `2(OCTAVE)`, `3(LINEAR)`, `4(LIST)`.
type	Real or complex.
padding	Padded or un-padded.
nvars	Number of variables are stored in the rawfile.
npoints	Number of calculated timepoints during simulation.
size_1, size_2, ..., size_8	The sizes and number of components in a vector. It will be in use only if the number of vector components are greater than one.
name_of_simulator	The name of the simulator that is used.
number_of_version	The version of simulator that is being used.
list_of_variables	The list of variables, which is printed as number, name, and type for each one.
binary_type	Binary format type. Table 6-7 contains a list of binary format types for simulation results data writing.
simulation results	The result of simulation. Contains for each time, frequency or DC sweep variable point the following information: • number of points (for ASCII format only); • current value of swept variable; • values of the output variables (in the same order, as in the `list_of_variables`). For complex variables, value will be written as a real part, then imaginary. When “Transpose binary” format is used, the output variables are written as vectors: • vector of the swept variable; • vectors of the output variables (in the same order, as in the `list_of_variables`).

Table 6-7 Table Simulation Data Binary Formats

binary_type content	Writing mode	POST value	Output function
	binary	1	`fwrite()`
	ASCII	2	`fprintf()`
Float	Single precision binary	3	`fwrite()`

When `.OPTIONS POST=2` is specified, the simulation results will be written in ASCII format and organized in columns. For complex output variables, real and imaginary parts are separated by a comma.

Example of Transient and AC analyses in ASCII format (POST=2)

```
Title:      DEMO EXAMPLE 1: INPUT FILE
Input deck file name: /home/Example/ex1_orig.in
Date: Wed Oct. 3 15:13:50 2001
Plotname: Transient Analysis, 27 deg C, qnl@bf
Temperature: 27
Sweepvar: time
Sweepmode: -1
Flags: real padded
No. Variables: 7
No. Points: 98
Source: SmartSpice
Version: 1.9.4.R
Variables:  0  time  time
            1  i(vcc)  current
            2  i(vee)  current
            3  i(vin)  current
            4  v(0)    voltage
            5  v(1)    voltage
            6  v(2)    voltage
            7   v(3)   voltage
Values:
 0   0.000000000000000e+00
    -2.298600788991904e-03
     2.312970466630704e-03
    -7.184838819556505e-06
     0.000000000000000e+00
     0.000000000000000e+00
    -7.184838819556505e-03

 1   1.000000000000000e-10
    -2.298600384564064e-03
     2.313032508790734e-03
    -7.463021459517354e-06
     0.000000000000000e+00
     3.141587485879564e-04
    -7.148862710929398e-03
...
Title:      DEMO EXAMPLE 1: INPUT FILE
Input deck file name: /home/Example/ex1_orig.in
```

```
Date: Wed Oct. 3 17:12:46 2001
Plotname: AC Analysis, 27 deg C
Temperature: 27
Sweepvar: frequency
Sweepmode: 1
Flags: complex padded
No. Variables: 7
No. Points: 61
Source: SmartSpice
Version: 1.9.4.R
Variables: 0  frequency  frequency  grid=4
           1  i(vcc)     current
           2  i(vee)     current
           3  i(vin)     current
           4  v(0)       voltage
           5  v(1)       voltage
           6  v(2)       voltage
           7  v(3)       voltage
Values:
 0   1.000000000000000e+04,  0.000000000000000e+00
    -1.007628145584596e-05, -1.414101670296514e-07
     1.033991645580937e-05,  1.708270571413179e-07
    -1.066113008995576e-04, -4.623650532164639e-06
     0.000000000000000e+00,  0.000000000000000e+00
     1.000000000000000e+00,  0.000000000000000e+00
     8.933886991004424e-01, -4.623650532164640e-03
 1   1.258925411794167e+04,  0.000000000000000e+00
    -1.007646892519157e-05, -1.780243989540594e-07
     1.033993886601264e-05,  2.150585311373695e-07
    -1.066652930704029e-04, -5.819450787179803e-06
     0.000000000000000e+00,  0.000000000000000e+00
     1.000000000000000e+00,  0.000000000000000e+00
     8.933347069295970e-01, -5.819450787179803e-03
...
```

If the input deck contains `.MEASURE` statements, the additional header/data segments (always in ASCII format) will be attached to the rawfile. This segment has the structure, as described above, with the field `Plotname: Measure Plot`, and the field `Variables` filled by variables from measurement statements.

The file with the extension `.dat` is created by the output statement `.PROBE` only in `-pspice` mode (`set simulator=pspice`), and has the same structure as rawfile. The field `Variables` is filled by variables from the `.PROBE` statements.

Analysis measurement results are written to `inputname.measure.raw#` when rawfile is not created. This file contains only the measurement results, which are always presented in ASCII format. The file has the same structure as the rawfile measurement header/data segment.

Example of Measurement Header/Data Segment

```
Title:      DEMO CIRCUIT2. INPUT FILE
Input deck file name: /home/valerys/my_examples/Example/ex2.in
Date: Wed Oct. 3 13:23:35 2001
Plotname: Measure Plot
Temperature: 0
Sweepvar: rbias(res)
Sweepmode: 0
```

```
Flags: real padded
No. Variables: 4
No. Points: 3
Source: SmartSpice
Version: 1.9.4.R
Variables:  0  rbias(res) notype
            1  cload(cap) notype
            2  max_tr_v3  notype
            3  min_tr_v3  notype
            4  fall_1_tr_v3notype
Values:
 0  2.500000000000000e+03
    7.500000000000000e-13
    1.202632883161306e+01
    3.075050652129222e+00

 1  2.250000000000000e+03
    4.500000000000001e-13
    1.202140632842488e+01
    2.085075835872537e+00

 2  2.025000000000000e+03
    2.700000000000001e-13
    1.201515786287814e+01
    9.436110771098196e-01
```

Use of .XML Output File

A file called `subcircuitnodemap.xml` is produced when a spice input deck is parsed, and the boolean variable `createsubcircuitnodemap` is set to `1` (Default=`0` since version 1.9.5.R of SmartSpice for a stand alone SmartSpice. Default=`1` when used under Gateway/ SmartView). This `.xml` file contains a header, subcircuit node map and alias information. The contents of this file can be easily seen by any browser, such as internet explorer.

An example deck is shown below:

```
* Use subcircuit node names in main level devices

V1 0 1 DC 2
R1 0 X1.Output 1K
X1 0 1 Test

.SUBCKT Test Input Output
R1 Input 1 2K
R2 1 Output 3K
.ENDS

.control
nodelist
.endc

.END
```

And the associated `.xml` file content is:

```
<?xml version="1.0" standalone="yes" ?>
  <!DOCTYPE subcircuitnodemap (View Source for full doctype...)>
- <subcircuitnodemap>
  <alias name="0" node="0" />
```

```
<alias name="gnd" node="0" />
<alias name="gnd!" node="0" />
<alias name="ground" node="0" />
<alias name="x1.input" node="0" />
<alias name="x1.output" node="1" />
</subcircuitnodemap>
```

6.8.1 FSDB Binary Raw File Generation

SmartSpice supports writing data in FSDB binary raw format. FSDB format is a SpringSoft (NOVAS) proprietary compressed binary raw file format. In case of multiple plots stored in single file, each can have a different number of points and unique time axis. Inside FSDB format, signals are represented as signal value changes, not signal values. This allows for a significantly reduced amount of saved data, especially with the use of resolution control options described below. For example, for the straight line plot (like DC voltage), it will store just 2 signal/time pairs - begin and end.

Resolution Control

Due to the fact that FSDB stores data by value changes, it is possible to control amount of data or roughness of a signal to be written. In other words, FSDB allows resolution control of a plot to be stored in a binary file. There are two options for resolution control, `FSDB_VPRBTOL=val` and `FSDB_IPRBTOL=val` for voltage and current signals, respectively. Setting a tolerance value `val` influences the criteria by which each signal plot point is accessed. If a signal difference between a previously written signal point and the current one is less than tolerance `val`, specified in the option, then the current point is discarded and the next signal point will be accessed in same way. If signal difference between a previously written point and the currently accessed one is equal or greater than specified tolerance value, the signal difference is to be written to the FSDB file together with respective time axis value.

Version Control

SmartSpice supports FSDB library version control for enhanced compatibility with existing FSDB readers and waveform viewers. The option `FSDB_VERSION=val` sets the version of proprietary SpringSoft(NOVAS) FSDB library to be loaded. Possible values are listed in the following table 6.xx. Default value is 2. (v.4.2)

Table 6-8 FSDB Version Option Values

FSDB_VERSION option value	FSDB library version
1	v.4.1
2	v.4.2
3	v.4.3
5	v.5.0

Example

```
.OPTIONS POST=20 FSDB_VPRBTOL=1.2E-6 FSDB_IPRBTOL=2.3E-9
.OPTIONS FSDB_VERSION=1
```

In this example, SmartSpice will set a FSDB binary raw file output using SpringSoft (NOVAS) FSDB library v.4.1. Waveforms will be written to the file with voltage resolution 1.2uV and current resolution 2.3nA (i.e., signal difference between adjacent plot points).

6.8.2 Support for PSF File Format in SmartSpice

PSF File Generation

1. By default, the PSF files are placed into a `psf` directory one level above the netlist. You can specify the name and location of the `psf` directory by using the command line switch `-psf_dir`. Usage:

   ```
   smartspice -psf_dir <dir>
   ```

   The basic reference element of PSF file set is the file `logFile`. This file contains links to all simulation data files within the `psf` directory.

2. The PSF library supports 2 types of PSF formats: ASCII and binary. To save output data in PSF ASCII format, specify `.OPTION PSF=1` in the netlist. If `.OPTION PSF=2` is specified, the output data will be saved in PSF binary format.

3. PSF library supports 2 versions of PSF binary file format:
   - SPECTRE 5 compatible (default)
   - SPECTRE 7 compatible (for SPECTRE 7.x and higher)

   When first executed the library is asking for environment variable `SPECTRE_V` and switches to SPECTRE 7 compatibility mode if it finds version number 7.x, 8.x, 9.x, or 10.x. Typical content of `SPECTRE_V` environment variable:

   ```
   SPECTRE_V 'sub-version  7.0.0.038'
   ```

4. SmartSpice supports generation of the following INFO files:
   - Elements (input parameters for instances of all components)
   - Models (input parameters for models of all components)
   - Output (effective and temperature-dependent parameter values)
   - Parameters (top level circuit parameters and their values)
   - Primitives (model parameters, oppoint parameters, output parameters, instance parameters, region parameters, and terminal names of a primitive, but no values)
   - Subcircuits (subcircuit parameters and terminal names)

   The INFO files are saved in PSF ASCII format. Detailed information on the `.INFO` statement can be found in Chapter 3 Statements ".INFO".

5. SmartSpice provides the ability to read PSF data files. SmartSpice has a **Load PSF Data…** menu item in the **File** menu under GUI mode operation. Then the **Open Spectre Log** dialog appears and offers you a choice of appropriate log files. The loaded data is then available in the **Vector** window.

SmartSpice allows saving of the following analyses output in PSF format:

- TRAN (`tran_1.tran, tran_2.tran, …`)
- DC (`dc_1.dc, dc_2.dc, …`)
- AC (`ac_1.ac, ac_2.ac, …`)
- STB (`stb.stb, stb.margin.stb`)
- DCOP (`dcOp_1.dc, dcOp_2.dc, …`)
- OP (`oppoint_1.info, oppoint_2.info, …`)
- NOISE (`noise_1.noise, noise_2.noise, …`)

6.9 Saving and Recovering of a Transient Analysis Simulation

SmartSpice is capable of recovering from an unexpected program termination during a transient analysis simulation. This functionality is performed through so called checkpoint files and is disabled by default. It can be activated through the environment variable or command line option `+checkpoint`.

Example

```
smartspice +checkpoint <filename>
```

<filename>	Input deck file name.

The command line option `-checkpoint` will disable the Save/Recover functionality in case it is already enabled with the `+checkpoint` environment variable.

Example

```
smartspice -checkpoint <filename>
```

<filename>	Input deck file name.

Note: The simulation, saved in batch mode, cannot be recovered in GUI mode, and vice versa. A warning will be issued if this recovery method is tried.

To actually control the creation of checkpoint files, additional options have to be set in the input deck (see Section 3.14“.OPTIONS (Option Specification)”). These are:

1. The `.OPTION CKPTCLOCK`

Example

```
.OPTION CKPTCLOCK=period,
```

where `period` is the regular time interval in seconds when the checkpoint files will be saved. This option will work for all transient analysis statements, except those containing the `ckptperiod` in the `.TRAN` statement.

2. The parameter `CKPTPERIOD` in `.TRAN` statement

Example

```
.tran 0.1n 150n ckptperiod=period,
```

where `period` is the regular time interval in seconds when the checkpoint files will be saved. This option only works for transient analysis, where the parameter `ckptperiod` is defined.

To recover a previously aborted simulation, SmartSpice has to be started with the `+recover` command line option along with input deck paths to an existing checkpoint file. To ensure the existence of a checkpoint file, look for the file with the `.sav` extension, and check that the name and location is the same as in the input deck.

Example

```
smartspice +checkpoint +recover <filename>
```

<filename>	Input deck file name.

Upon successful completion of the simulation run, all checkpoint files will be automatically deleted.

The multiple savings of simulation are implemented. The statement format is as following:

Syntax

```
.tran ... ckptlist <points_num> <point1> <point2> .. <point_num>
...
```

ckptlist	a keyword
points_num	Specifies a number of savings during simulation.
point1..	Time points when simulation will be saved.

6.9.1 MultiSave Mode

The multisave mode can be activated through the command line option `+multisave`. Each time the simulator produces the unique `.sav` file with the filename:

```
input_deck_file_name(sim_time).sav
```

where:

input_deck_file_name	input deck name
sim_time	Time point when simulation has been saved.

It is possible to restore a simulation at the specific time point using the command line option `+recover:time`.

6.9.2 SR_ABSTOL, SR_RELTOL, SR_VNTOL, SR_LTERATIO, SR_LVLTIM

Overrides corresponding option values for `ABSTOL`, `RELTOL`, `VNTOL`, `LTERATIO` or `LVLTIM` initially after `RESTORE` when the `SAVE/RESTORE` feature is enabled.

Syntax

```
.OPTION OPTIONNAME = val
```

where:

```
OPTIONNAME = { SR_ABSTOL | SR_RELTOL | SR_VNTOL | SR_LTERATIO |
SR_LVLTIM }
```

If `SR_RELTOL=1e-4` is specified in the netlist and the feature `SAVE/RESTORE` is used, SmartSpice will override current `RELTOL` value with the value specified in `SR_RELTOL (1e-4)` initially after `RESTORE` and will continue simulation with this new tolerance.

Example

```
.OPTION SR_RELTOL = 1e-4
```

Modifying accurate/timestep Control Options after RESTORE

To do the rest of the simulation from the restore point (e.g., 620n) to the final point (700n) with the new option value (e.g., RELTOL) is equal to 1e-4 after RESTORE when the SAVE/RESTORE feature is used, the following steps should be done.

1. The original netlist should contain the .OPTION SR_RELTOL=1e-4, and the .TRAN statement should be adjusted accordingly to the restore point (in this example, one save/restore point at 620.0n, step is 1e-11, stop is 700n) as follows:

   ```
   .TRAN 1e-11 0.7e-6 ckptlist 1 620.0n
   ```

2. In batchmode SmartSpice should be launched the first time with the next command line:

   ```
   -b filename.in +checkpoint +multisave
   ```

After first simulation, the SAVE/RESTORE file filename(620.0n).sav and, if the correspondent options are set in the netlist or through the command line flags, the *.raw and the output files will be created on disk at the simulation directory. During the first full simulation the RELTOL value will not be changed from the original.

Note: The multisave command line argument allows to use more than one save/restore points if ckptlist list contains more than one timepoint

---

To do the rest of the simulation from the restore point (620n) to the final point (700n) with the new option RELTOL is equal to 1e-4 after RESTORE at 620n SmartSpice should be run the second time with the next command line

```
-b filename.in +checkpoint +recover:620.0n
```

During the second simulation the RELTOL value will be changed from the original to 1e-4 right after RESTORE at 620n and simulation will continue from 620n to 700n with this changed value.

Note: During the second simulation the original RAW file will be appended and the output file will be overwritten if it were created during the first simulation.

Chapter 7
IBIS Model Support

7.1 Introduction

The Input/Output Buffer Information Specification (IBIS) is a standard for electronic behavioral models based on I/V and V/T curve data. It is being developed by the IBIS Open Forum, which is affiliated with the Electronics Industry Alliance (EIA). These models are suitable for high-speed designs of digital systems to evaluate Signal Integrity issues (deformation of electronic signals, cross-talk, power/ground bounce, transmission lines...) on printed circuit boards (PCBs).

IBIS standard offers a way to provide fast and accurate models of I/O buffers without divulging any proprietary technology process. As it protects IP, it is now widely used by semiconductor vendors as a replacement for SPICE netlists.

The IBIS standard specifies only what kind of information is provided, how this information is presented in ASCII files, and how some data is derived from measurements or simulations. How this data is used and processed by a simulator is not part of the standard. The purpose of this documentation is to present the IBIS model support in SmartSpice.

The reader who is not familiar with the IBIS standard, or would like to learn more, may refer to the Web site of the IBIS Open Forum at:

```
http://www.eda.org/pub/ibis
```

where numerous documents are available for download, including introductions, slide shows, articles and complete specifications (from the original v1.0 to the latest v4.1 of January 2004).

7.2 IBIS Buffer Equivalent Circuit

A buffer is implemented as a new element in SmartSpice. Even though different types are available to cover a wide range of functions and technologies, most are based on the same equivalent circuit, which is shown in Figure 7-1 below.

Several elements and terminals are optional depending on the buffer type: Input (node IN) and Enable (node EN) high-impedance inputs, Output (node OUT) voltage source and conductance, and Pullup/Pulldown (nodes PU/PD) Voltage-Controlled Current Sources (VCCS). Only Input/Output buffers have all elements and terminals available.

The Gnd node corresponds to the SPICE ground node, also called node 0. All connections to Gnd are internal (`C_comp`, `Vout`...) so this node is not available as a terminal of the buffer device.

The die capacitance `C_comp` specified in IBIS models is usually connected between IO node and ground. However, if die capacitances `C_comp_pc`, `C_comp_gc`, `C_comp_pu` and `C_comp_pd` are specified in the related IBIS model instead of `C_comp`, these four capacitances are connected between IO and PC, GC, PU and PD nodes, respectively.

PC/PU and GC/PD terminals are supposed to connect power and ground rails, respectively. By default, they are connected to internal voltage sources (not shown on the circuit diagram) and should not be connected to any other elements in the netlist (especially voltage sources). However, the instance parameter `power` may be used to allow connections to external elements or power supplies. Please see below for further information about this instance parameter.

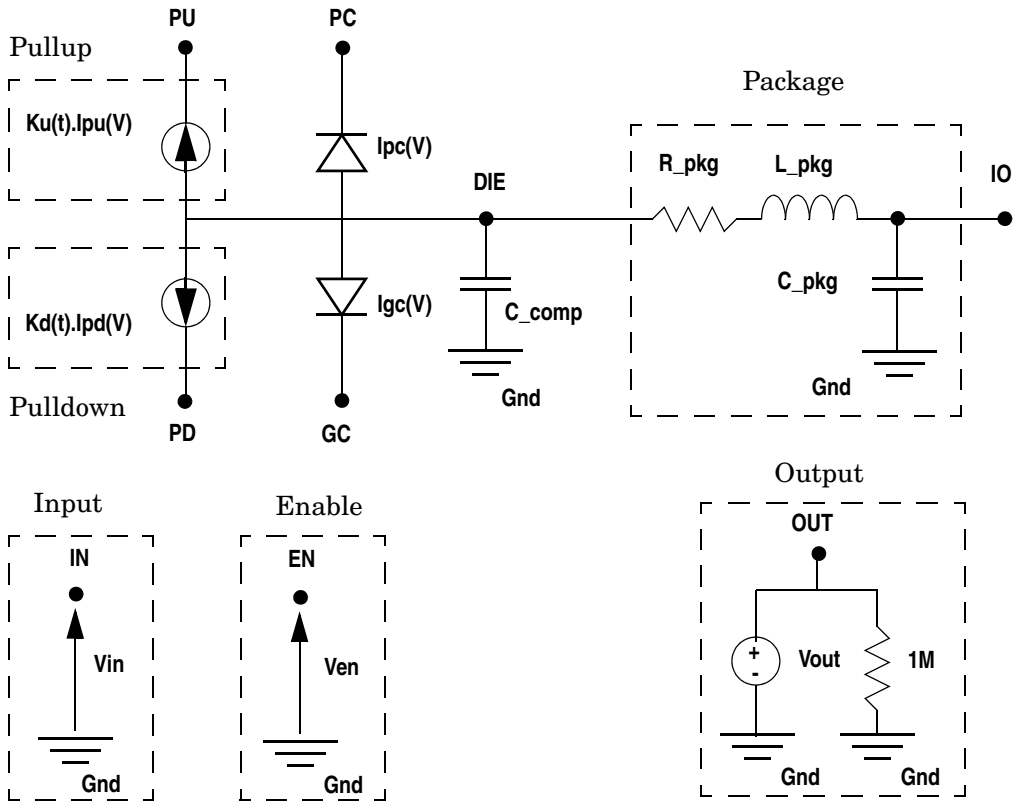


Figure 7-1 IBIS Buffer General Circuit Diagram

7.2.1 Buffer Types and Related Circuitry Descriptions

Table 7-1 gives details about the topology of the equivalent circuit for the 17 buffer types currently supported in SmartSpice:

Table 7-1 Buffer Types and Related Circuitry Description

Type/Number	Terminals (min/max)	Input	Enable	Output	Pullup	Pulldown
`input/1`	4/4	no	no	yes	no	no
`output/2`	4/6	yes	no	no	yes	yes
`input_output/3`	6/8	yes	yes	yes	yes	yes
`three_state/4`	5/7	yes	yes	no	yes	yes
`open_drain/5`	4/6	yes	no	no	no	yes
`io_open_drain/6`	6/8	yes	yes	yes	no	yes
`open_sink/7`	4/6	yes	no	no	no	yes
`io_open_sink/8`	6/8	yes	yes	yes	no	yes

Table 7-1 Buffer Types and Related Circuitry Description

Type/Number	Terminals (min/max)	Input	Enable	Output	Pullup	Pulldown
open_source/9	4/6	yes	no	no	yes	no
io_open_source/10	6/8	yes	yes	yes	yes	no
input_ecl/11	4/4	no	no	yes	no	no
output_ecl/12	3/5	yes	no	no	yes	yes
io_ecl/13	5/7	yes	yes	yes	yes	yes
three_state_ecl/14	4/6	yes	yes	no	yes	yes
series/15	2/2	NA	NA	NA	NA	NA
series_switch/16	2/3	NA	NA	NA	NA	NA
terminator/17	3/3	no	no	no	no	no

`series` and `series_switch` buffers have a totally different equivalent circuit. `terminator` buffers also account for extra parameters corresponding to passive elements. These elements are not shown in Figure 7-1. The complete equivalent circuits for `series`, `series_switch` and `terminator` buffers are given below with the information specific to these types.

7.2.2 Optional Package Circuitry

Usually, buffers correspond to [Model] descriptions in IBIS files, which do not include package parasitics. The corresponding RLC elements may be added manually in the netlist using the values specified in [Package] or [Pin] keywords in IBIS files. However, SmartSpice offers a simple alternative to automatically account for this information without adding any external elements in the netlist. The optional package circuitry is internally created between an IO terminal and a DIE internal node when the instance parameters component and/or pin are specified on the device line. If component and pin are not specified, DIE and IO nodes are connected together to ensure compatibility with other simulators.

7.2.3 Optional Transit Time Capacitances

SmartSpice can account for optional transit time parameters [TTgnd] and [TTpower]. If specified among IBIS model data, the corresponding capacitances (not shown on Figure 7-1) are connected in parallel with ground clamp and power clamp diodes. Their values are computed as a function of DC conductances and transit times using the following expressions:

$$C_{GC} = TTgnd \cdot \frac{\partial}{\partial Vgc} Igc$$

$$C_{PC} = TTpower \cdot \frac{\partial}{\partial Vpc} Ipc$$

7.2.4 Optional [Model Spec] Data

Even though [Model Spec] data are not fully supported, SmartSpice systematically checks whether a [Model Spec] section is defined for each model and gets Vinl/Vinh values if specified. According to IBIS specifications, the sub-parameters Vinl/Vinh specified in a [Model Spec] section overrides the corresponding values specified in the [Model]

description. As Vinl/Vinh correspond to ranges in [Model Spec] instead of single values in [Model], they are used when min/max values of thresholds are required to improve accuracy. Thresholds are accounted for only for input and IO buffers.

7.3 IBIS Buffer Device Line

Using buffers in SmartSpice is identical to using other elements, like passive or semiconductor devices.

The general syntax of a buffer statement is given by:

```
Bname term1 term2 [term3 [term4 [term5 [term6 [term7 [term8]]]]]]
+ file = 'filename'
+ [model = 'modelname'] | [component = 'componentname'
+ [pin = 'pinname']]
+ [typ = {typ|min|max|fast|slow}] [power = {on|off}]
+ [interpol = {1|2}] [buffer = {number|type}]
+ [ramp_rwf = {0|1|2}] [ramp_fwf = {0|1|2}] [fwf_tune = value]
+ [rwf_tune = value]
+ [c_comp_pc = value] [c_comp_gc = value] [c_comp_pu = value]
+ [c_comp_pd = value]
+ [nowarn] [mac = value] [risedly = value] [falldly = value]
```

7.3.1 Bname

The buffer element name must begin with B followed by optional alphanumeric characters.

7.3.2 Terminals

The number and the name of a terminal depends on the buffer type:

- Input and Input_ecl buffers:
  `B_input PC GC IO OUT`
- Output, Open_drain, Open_sink, Open_source buffers:
  `B_output PU PD IO IN [PC [GC]]`
- Three_state buffers:
  `B_three_state PU PD IO IN EN [PC [GC]]`
- Input_output, IO_open_drain, IO_open_sink and IO_open_source buffers:
  `B_input_output PU PD IO IN EN OUT [PC [GC]]`
- Output_ecl buffers:
  `B_output_ecl PU IO IN [PC [GC]]`
- IO_ecl buffers:
  `B_io_ecl PU IO IN EN OUT [PC [GC]]`
- Three_state_ecl buffers:
  `B_three_state PU IO IN EN [PC [GC]]`
- Series buffers:
  `B_series IN OUT`
- Series_switch buffers:
  `B_series_switch IN OUT [EN]`
- Terminator buffers:
  `B_terminator PC GC IO`

Open_drain, IO_open_drain, Open_sink and IO_open_sink buffers have no pullup circuitry, but PU terminal must be specified even though they are not connected to internal

elements. Open_source and IO_open_source buffers have no pulldown circuitry, but PD terminal must be specified even though they are not connected to internal elements.

Ouput_ecl, Three_state_ecl and IO_ecl buffers have pullup and pulldown circuitry, but not a PD terminal because this latter node is internally connected to a PU node.

When power=off, no internal voltage sources are created. For this case, PC and GC terminals must be specified to supply power clamp and ground clamp diodes.

When power=on (default), there is usually no need to connect PU, PD, PC and GC terminals to other nodes in the circuit but they must be specified on the device line to correctly parse other terminals. That's why SmartSpice offers an alternative method to specify terminals from specific instance parameters, which makes PU, PD, PC and GC optional terminals for all buffer types. Please refer to Section 7.8.1"Alternative Terminal Specification Method" for a complete description of this feature.

7.3.3 Required Parameters

Specifying the file parameter is required to define the location of the IBIS file containing the IBIS model description for this buffer. In SmartSpice, this parameter is also used to decide if a B statement corresponds to a MESFET device or to an IBIS buffer. See Section 7.11"Backward Compatibility" for further details.

In order to find the [Model] section in the IBIS file it is necessary to properly set the model parameter. However, if component and pin are given, model becomes optional. According to IBIS specifications, a valid pin description always contains the name of the associated model, which is used in SmartSpice as the default value of model.

Note: The model name specified on any [Pin] line may also correspond to a [Model Selector] instead of a [Model]. For this case, the first entry in the [Model Selector] list is used as the model name. For convenience, the instance parameter model may also correspond to a valid [Model Selector].

---

component and pin are used to account for package parasitics. If component is specified, SmartSpice first checks whether a [Component] description named componentname exists in the IBIS file, then gets the values of R_pkg, L_pkg and C_pkg from the corresponding [Package] description. R_pkg, L_pkg and C_pkg sub-parameters correspond to ranges in IBIS files, so the values used for the simulation also depend on the value of typ. If pin is also specified, SmartSpice checks whether a pin named pinname exists in the [Pin] list associated to the selected component. The first column of a [Pin] list contains pin names. A [Pin] list is a sub-parameter of a [Component] description in IBIS files. Consequently pin should not be specified if component is missing on the device line, or if componentname does not match a valid [Component] name. If the optional parameters R_pin, L_pin and C_pin are available for the selected pin, these values override the default package values _pkg, L_pkg and C_pkg, respectively. For this case, R_pin, L_pin and C_pin do not correspond to ranges and so do not depend on the value of typ.

When only file and component are specified, SmartSpice creates several buffers according to the [Pin] list and may connect power and ground supplies to POWER and GND buses

according to the optional [Pin Mapping] list. This feature is documented in Section 7.9“IBIS Components”.

‘filename’	Case-sensitive and must correspond either to the absolute path to the IBIS file or the relative path. For this latter case, IBIS files are searched from the directory where the simulator run, from the directory of the netlist containing the corresponding B statement (which may be different from the current working directory), and from the paths specified in the option ‘d_ibis’. For convenience, the characters ’\’ and ’/’ are valid path delimiters in file and d_ibis arguments, regardless of the operating system.
‘modelname’ ‘componentname’ ‘pinname’	Case-sensitive and must match one of the [Model], [Component] and [Pin] names defined in the IBIS file.

In brief, at least two parameters must be specified to create IBIS buffers or components. The syntax of all B statements must match one as follows:

- Single buffer without parasitics:

```
Bname [term1 ... [term8]]
+ file=‘filename’ model=‘modelname’
+ [optional parameters]
```

- Single buffer with package parasitics:

```
Bname [term1 ... [term8]]
+ file=‘filename’ model=‘modelname’ component=’component_name’
+ [optional parameters]
```

- Single buffer with pin parasitics:

```
Bname [term1 ... [term8]]
+ file=‘filename’ model=‘modelname’ component=’component_name’
+ pin=’pin_name’
+ [optional parameters]
```

- Single buffer with pin model and parasitics:

```
Bname [term1 ... [term8]]
+ file=‘filename’ component=’component_name’ pin=’pin_name’
+ [optional parameters]
```

- Component (multiple buffers):

```
Bname term1 [term2 ... [termN]]
+ file=‘filename’ component=’component_name’
+ [optional parameters]
```

7.3.4 Optional Parameters

typ must be set to select what column of IBIS range is used during the simulation: TYP (default), MIN, MAX, SLOW or FAST. If FAST or SLOW are specified, the column MIN or MAX is selected depending on the IBIS parameter as defined in Table 7-2:

Table 7-2 Min/Max Combinations for Slow/Fast Conditions

IBIS Parameter/Data	Fast	Slow
C_comp	min	max
C_comp_pc	min	max
C_comp_gc	min	max
C_comp_pu	min	max
C_comp_pd	min	max
Voltage_range	max	min
Pullup_reference	max	min
Pulldown_reference	min	max
Power_clamp_reference	max	min
Gnd_clamp_reference	min	max
Pulldown	max	min
Pullup	max	min
Gnd_clamp	max	min
Power_clamp	max	min
Ramp	max	min
Rising_waveform	max	min
Falling_waveform	max	min
V_fixture	max	min
R_pkg	min	max
L_pkg	min	max
C_pkg	min	max
TTgnd	min	max
TTpower	min	max
Vinh (in [Model Spec])	min	max
Vinl (in [Model Spec])	max	min

This is especially useful for best case/worst case analysis. If min or max values are not available in the IBIS model for a given parameter, typ values are used.

power is used to select how the buffer is powered from PC, GC, PU and PD nodes (if these latter nodes exist for the given buffer type).

- If power is set to on (default), these nodes are internally connected to voltage sources whose values are taken from the IBIS parameters: [POWER Clamp Reference], [GND Clamp Reference], [Pullup Reference], [Pulldown Reference] (or [Voltage Range] if preceding parameters are missing). For this case, terminal names specified on the element card may be useful to print out the voltage values.
- If power is set to off, internal voltage sources are not created and PC, GC, PU and PD nodes must connect to external voltage sources either directly, through passive devices like RLC networks, or transmission lines.

interpol is the interpolation method selector.

- If interpol is set to 1 (default), I/V curves are interpolated using linear interpolation.
- If interpol is set to 2, quadratic bi-spline interpolation is used. This latter method is not recommended and useless if IBIS data is accurate.

buffer is used to specify the type of buffer. This value overrides the corresponding IBIS parameter Model_type. However, it is not recommended to specify a different value. This may lead to unpredictable results. Integer values are allowed to select a buffer. The correspondence with literal names for all types currently supported in SmartSpice is given in Table 7-1.

ramp_fwf and ramp_rwf selectors allow you to choose the calculation method of multipliers Ku(t) and Kd(t). These parameters are totally independent and may have different values.

- If ramp_fwf (or ramp_rwf) is set to 0, only the ramp data is used to derive multipliers for the falling (or rising) transition.
- If ramp_fwf (or ramp_rwf) is set to 1, the first falling (or rising) waveform table available in IBIS model is used to derive the corresponding multipliers.
- If ramp_fwf (or ramp_rwf) is set to 2 (default), the first two falling (or rising) waveform tables available in IBIS model are used to derive the corresponding multipliers.

This latter option is highly recommended to get accurate results in transient analysis. However, if the required data is not available in the IBIS model, the value of ramp_fwf (or ramp_rwf) is decremented and a warning message is issued. For example, if ramp_fwf=2 and only one waveform table is given, then ramp_fwf is set to 1; if ramp_fwf=1 and only ramp data is given, then ramp_fwf is set to 0.

fwf_tune and rwf_tune factors are control parameters for ramp_fwf=0, 1 and ramp_rwf=0, 1 algorithms, respectively. When only ramp data or one waveform is available, it is necessary to impose an additional condition to compute multipliers. It is assumed that Ku(t)+K(t)=1, which was demonstrated to be unrealistic because the circuitry that goes from ON to OFF undergoes this transition faster than the circuitry that goes from OFF to ON.

By setting fwf_tune or rwf_tune to a value between 0 and 1 (default 0.1), it is possible to get more accurate transitions by using the following assumption: if deltaT is the duration of a complete transition, the multiplier K(t) corresponding to the circuitry that goes from ON to OFF decreases linearly from 1 to 0 between t=0 and t=fwf_tune * deltaT (or t=rwf_tune * deltaT depending on the transition). Thus, the other multiplier is uniquely determined from an IBIS ramp or one IBIS waveform. The multiplier computation methods are described in several articles whose references are given at the end of this section (1 and 2).

C_comp_pc, C_comp_gc, C_comp_pu and C_comp_pd are dimensionless die capacitance partitioning factors. They do not override the IBIS parameters with the same names, which correspond to actual die capacitances. If these latter capacitances are specified in the IBIS model, the dimensionless factors are useless and ignored if given on the element card. If only C_comp is available in the IBIS model, it may be desirable to split it into several parts for simulating power/ground bounce. This is achieved by specifying the fractions of C_comp connected between IO node and PC, GC, PU, PD nodes. If given, the values of instance parameters C_comp_pc, C_comp_gc, C_comp_pu and C_comp_pd should be between 0 (default) and 1. It is also expected that their sum equals 1.

nowarn is a flag and may be set to turn off all non-critical warnings issued when reading an IBIS file.

mac is a selector used to turn on (mac=1, default) and off (mac=0) the DC/AC Mismatch Auto Correction algorithm. Please refer to Section 7.8.3“DC/AC Mismatch Auto Correction Algorithm” for further information. This parameter is ignored for buffer types with no pullup/pulldown stages.

risedly and falldly may be specified to delay rising and falling transitions, respectively. By default, they are set to 0s so that a transition of the output voltage starts as soon as a voltage change is detected on input (or enable) pins. They are ignored for buffer types with no pullup/pulldown stages.

7.4 Buffer Logical State

The logical state of a buffer is controlled by the voltage of IO, IN and/or EN nodes relative to ground and noted Vio, Vin and Ven, respectively. The way the logical state is set is type-dependent. Series and terminator buffers have no internal logical state because they have no control terminals. Series_switch buffers have an internal logical state which corresponds to the state of the switch (ON or OFF). Please refer to the related paragraph below for further details.

For buffers with no controlling signals (no IN or EN nodes), the state is a function of Vio, the IBIS parameters Vin_l, Vin_h (thresholds), Polarity and the previous state if any.

If Polarity=Non-Inverting

- Initially (t=0 in transient analysis or first computed point of a DC sweep), state is set to LOW if Vio<(Vin_h+Vin_l)/2 or to HIGH in the opposite case.
- If state=HIGH then it goes to LOW only if Vio<Vin_l.
- If state=LOW then it goes to HIGH only if Vio>Vin_h.

else Polarity=Inverting

- Initially (t=0 in transient analysis or first computed point of a DC sweep), state is set to LOW if Vio>(Vin_h+Vin_l)/2 or to HIGH in the opposite case.
- If state=HIGH then it goes to LOW only if Vio>Vin_h.
- If state=LOW then it goes to HIGH only if Vio<Vin_l.

For buffers with only one controlling signal (IN node), the state is a function of Vin, the IBIS parameter: Polarity and the previous state if any. Here thresholds are constant built-in parameters.

If Polarity=Non-Inverting

- Initially (t=0 in transient analysis or first computed point of a DC sweep), state is set to HIGH if Vin>0.5 or to LOW in the opposite case.
- If state=HIGH then it goes to LOW only if Vin<0.2.

- If state=LOW then it goes to HIGH only if Vin>0.8.

else Polarity=Inverting

- Initially (t=0 in transient analysis or first computed point of a DC sweep), state is set to HIGH if Vin<0.5 or to LOW in the opposite case.
- If state=HIGH then it goes to LOW only if Vin>0.8.
- If state=LOW then it goes to HIGH only if Vin<0.2.

For buffers with two controlling signals (IN and EN nodes), the state is a function of Ven, Vin, Vio, the IBIS parameters: Vin_l, Vin_h (thresholds), Polarity, Enable and the previous state if any. The enable signal Ven supersedes the input signal Vin, and is used to determine whether the buffer is in ENABLE or DISABLE state:

If Enable=Active-High

- Initially (t=0 in transient analysis or first computed point of a DC sweep), buffer is ENABLE if Ven>0.5 or DISABLE in the opposite case.
- If buffer=ENABLE then it goes to DISABLE only if Ven<0.2.
- If buffer=DISABLE then it goes to ENABLE only if Ven>0.8.

else Enable=Active-Low

- Initially (t=0 in transient analysis or first computed point of a DC sweep), buffer is ENABLE if Ven<0.5 or DISABLE in the opposite case.
- If buffer=ENABLE then it goes to DISABLE only if Ven>0.8.
- If buffer=DISABLE then it goes to ENABLE only if Ven<0.2.

If a buffer is ENABLE, the state is controlled by Vin according to the rules defined above for buffers with only one controlling signal (IN node).

If a buffer is DISABLE, there are two possible behaviors depending on the type:

- For buffers without output circuitry (no OUT node), the state is just locked till the buffer returns to ENABLE. Three-state buffers belong to this family.
- For buffers with output circuitry (OUT node), the state is controlled by Vio according to the rules defined above for the buffers with no controlling signals. Input-output buffers belong to this family.

The logical state can be printed out if the output circuitry (OUT node) is available:

- If state=HIGH then Vout=1.0V. If state=LOW then Vout=0.0V.

OUT node can also connect to external elements, especially IN or EN nodes of other buffers. These nodes offer a simple way to create complex digital blocks in a SPICE netlist.

7.5 Output Variables

The following internal variables can be printed out using the SmartSpice syntax `@B_name[variable_name]`:

Variable Name	Definition
ku	Pullup transient current multiplier
kd	Pulldown transient current multiplier
cio	Input/Output terminal current
cpc	Power Clamp terminal current
cgc	Ground Clamp terminal current
cpu	PullUp terminal current
cpd	PullDown terminal current
cin	Input terminal current
cen	Enable terminal current
cout	Output terminal current

7.6 Series and Series_Switch Buffers

`series` and `series_switch` buffers are two terminal devices. Their equivalent circuit is totally different from other IBIS buffers as shown in Figure 7-2:

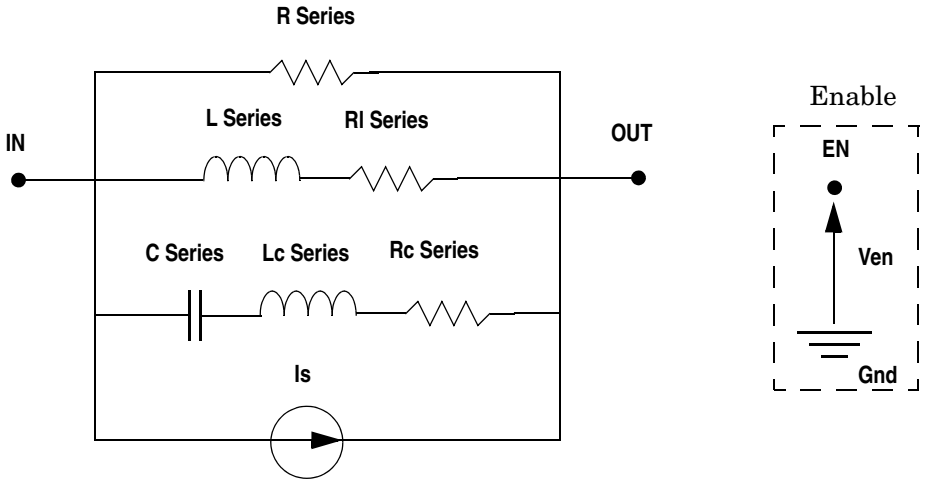


Figure 7-2 Series and Series_switch Buffers Circuit Diagram

All elements connected between `IN` and `OUT` terminals are optional. So the most simple series buffer is just a resistor. The value of passive elements is directly taken from `[Model]`

descriptions in IBIS files. If a value is not available or is unrealistic (negative resistance, inductance or capacitance), the corresponding element is simply removed from the circuit. More, [L Series] must be set to a positive value otherwise the entire branch L/Rl is not created, even though [Rl Series] is set to a positive value. Similarly, [C Series] must be set to a positive value otherwise the entire branch C/Lc/Rc is not created, even though [Lc Series] and/or [Rc Series] are set to a positive value.

The internal current source Is depends on data specified in [Series Current] and [Series MOSFET] tables. Please refer to IBIS specs for a complete description of these data.

[R Series], [L Series], [Rl Series], [C Series], [Lc Series], [Rc Series] and the elements of [Series Current] and [Series MOSFET] tables correspond to ranges in IBIS files, so the values used for the simulation also depend on the value of typ. For fast and slow values, the column is chosen according to Table 7-3.

Table 7-3 Series data: Min/Max combinations for Slow/Fast conditions

IBIS Parameter/Data	Fast	Slow
R Series	max	min
Rl Series	max	min
Rc Series	max	min
L Series	min	max
C Series	min	max
Lc Series	min	max
Series Current	max	min
Series MOSFET	max	min

The optional control terminal EN is used to switch between On and Off states defined in [On] and [Off] sections of IBIS files (for series_switch buffers only). If unspecified, the series_switch buffer is assumed to be in the On state, otherwise the state depends on the control voltage according to the following rules:

- Initially (t=0 in transient analysis or first computed point of a DC sweep), the switch is ON if Ven>0.5 or OFF in the opposite case.
- If the switch is ON, then it turns OFF only if Ven<0.2.
- If the switch is OFF, then it turns ON only if Ven>0.8.

Unlike other buffer types, there is no optional package circuitry. So specifying the instance parameters component and/or pin is useless. Only file and model are required. The other instance parameters nowarn and interpol are useless and simply ignored if specified.

7.7 Terminator Buffers

A terminator is similar to an input buffer, except that it has no internal logical state (no output terminal, `Vinh` and `Vinl` not required) and may include optional passive elements as shown in Figure 7-3.

`Rpower`, `Rgnd`, `Rac` and `Cac` sub-parameters correspond to ranges in IBIS files, so the values used for the simulation also depend on the value of `typ`. For `fast` and `slow` values, the column is chosen according to Table 7-4. If a value is not available or is unrealistic (negative resistance or capacitance), the corresponding element is simply removed from the circuit. Cac must be set to a positive value otherwise the entire branch `Rac/Cac` is not created, even though `Rac` is set to a positive value.

Like other buffer types, the optional package circuitry is internally created between the `IO` terminal and `DIE` internal node when the instance parameters `component` and/or `pin` are specified.

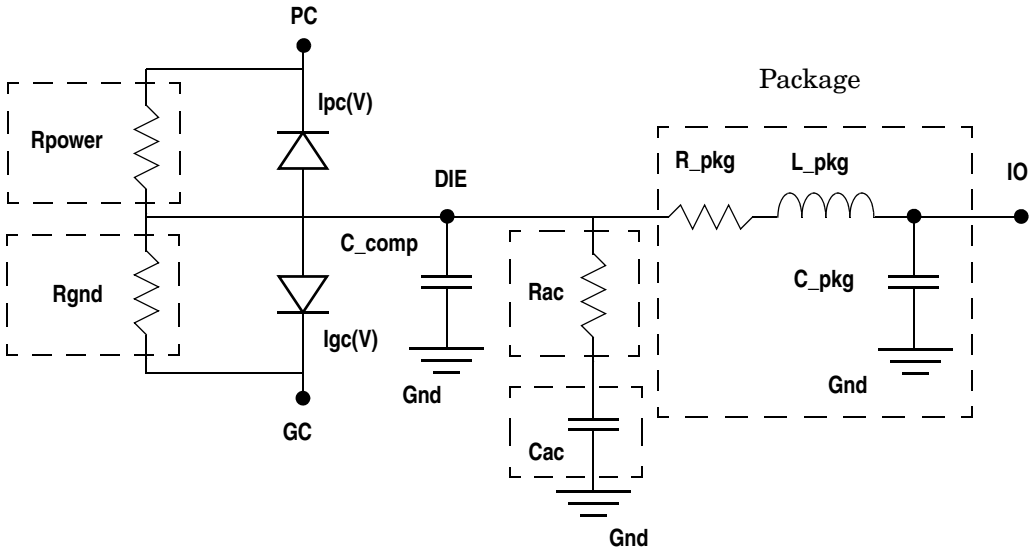


Figure 7-3 Terminator Buffer Circuit Diagram

Table 7-4 Min/Max Combinations for Slow/Fast Conditions

IBIS Parameter/Data	Fast	Slow
`Rpower`	max	min
`Rgnd`	max	min
`Rac`	max	min
`Cac`	min	max

7.8 Silvaco Improvements

7.8.1 Alternative Terminal Specification Method

SmartSpice offers another way to specify terminals using dedicated instance parameters. So, instead of specifying node names in a given order after the buffer name, node names can be specified using the following instance parameters: pc, gc, pu, pd, in, en, io and out.

The alternative syntax of a buffer statement is given by:

```
Bname
+ [pc=term1] [gc=term2] [pu=term3] [pd=term4]
+ [in=term5] [en=term6] [io=term7] [out=term8]
+ standard required parameters ...
+ [standard optional parameters ...]
```

These special instance parameters may be specified anywhere among other standard IBIS instance parameters. They are automatically used if no node names are specified after the device name, otherwise all are ignored and the standard method for specifying terminal nodes is used. They may be required, optional or simply ignored, depending on the buffer type and the value of power as described below:

- Input and Input_ecl buffers:

  ```
  B_input IO=term1 OUT=term2 [PC=term3] [GC=term4]
  ```

- Ouput, Open_drain, Open_sink, Open_source buffers:

  ```
  B_output IO=term1 IN=term2 [PC=term3] [GC=term4] [PU=term5]
  [PD=term6]
  ```

- Three_state buffers:

  ```
  B_three_state IO=term1 IN=term2 EN=term3 [PC=term4] [GC=term5]
  [PU=term6] [PD=term7]
  ```

- Input_output, IO_open_drain, IO_open_sink and IO_open_source buffers:

  ```
  B_input_output IO=term1 IN=term2 EN=term3 OUT=term4
  + [PC=term5] [GC=term6] [PU=term7] [PD=term8]
  ```

- Output_ecl buffers:

  ```
  B_output_ecl IO=term1 IN=term2 [PC=term3] [GC=term4] [PU=term5]
  ```

- IO_ecl buffers:

  ```
  B_io_ecl IO=term1 IN=term2 EN=term3 OUT=term4 [PC=term5] [GC=term6]
  [PU=term7]
  ```

- Three_state_ecl buffers:

  ```
  B_three_state IO=term1 IN=term2 EN=term3 [PC=term4] [GC=term5]
  [PU=term6]
  ```

- Series buffers:

  ```
  B_series IN=term1 OUT=term2
  ```

- Series_switch buffers:

  ```
  B_series_switch IN=term1 OUT=term2 [EN=term3]
  ```

- Terminator buffers:

  ```
  B_terminator IO=term1 [PC=term2] [GC=term3]
  ```

The main interest of this alternative method is that the four terminals PC, GC, PU, PD, may be unspecified when power=on (default), which is not possible with the standard syntax.

7.8.2 Extended Parsing Capabilities

The IBIS parser implemented in SmartSpice allows you to read `.ibs` files whose syntax is not fully compliant with IBIS specifications. However, warnings are issued to identify non-standard files.

For example:

- All sub-parameters and arguments in `.ibs` files should be case-sensitive, but SmartSpice can recognize sub-parameters and arguments regardless of their case.
- File names should be lower-case, but SmartSpice accepts file names containing upper-case letters.

7.8.3 DC/AC Mismatch Auto Correction Algorithm

Ideally, `V/t` waveforms in IBIS models should contain data consistent with `I/V` curves. That's why the Golden Parser checks for `DC` and `AC` endpoints. If a mismatch less than 2% is detected, a warning is issued. Such a small discrepancy should never affect the overall accuracy of transient results. If a mismatch above 2% is detected, an error is issued. For this case the IBIS file should not be used, especially when accuracy is needed. However, SmartSpice has an internal Mismatch Auto Correction algorithm which may help to get accurate results. The instance parameter `mac` is a selector used to invoke the Mismatch Auto Correction algorithm. If `mac=0`, no correction is performed and IBIS raw data are used. If `mac=1` (default), SmartSpice assumes that `I/V` data are accurate and adjusts `V/t` data so that endpoints (which correspond to steady-states) match `DC` points by keeping the global shape of waveforms. By turning the `MAC` algorithm on and off, it is possible to see the effect of the mismatch on transient results. If there is no (or small) mismatch between IBIS data, no significant differences should be observed.

7.8.4 Accuracy Maintained For Overclocked Buffers

IBIS models are not suitable to simulate the behavior of buffer outputs when input signals change before a transition is complete (overclocking). Usually this may lead to unrealistic results, especially when short pulses are applied on control terminals. In this case, SPICE equivalent circuits are recommended to get accurate results.

However, the computation of multipliers in SmartSpice has been improved to also account for overclocked buffers. So that results are now closer to those obtained when running the equivalent SPICE circuit of the IBIS buffer. The loss of accuracy is usually low enough to allow correct signal integrity simulations.

7.9 IBIS Components

One way to define an IBIS component in a netlist is to add one B statement for each pin listed in the [Component] description if the model_name matches a valid [Model] definition. This solution is not convenient for components with a large number of pins (100+ is a common value for modern interface circuits). That's why SmartSpice can also create a component with only one B-statement by using the following syntax:

```
Bname term1 [term2 ... [termN]]
+ file = 'filename' component = 'componentname'
+ [typ = {typ|min|max|fast|slow}] [interpol = {1|2}]
+ [ramp_rwf = {0|1|2}] [ramp_fwf = {0|1|2}] [fwf_tune = value]
+ [rwf_tune = value]
+ [c_comp_pc = value] [c_comp_gc = value] [c_comp_pu = value]
+ [c_comp_pd = value]
+ [nowarn] [mac = value] [risedly = value] [falldly = value]
```

Here file and component are required parameters, but model and pin must be unspecified. The number of terminals (N) specified after the device name must match the number of pins listed in the [Pin] list of the [Component] description in .ibs file. A new buffer named Bname.pin_name is created for every pin whose model_name is not a reserved word, such as POWER, GND or NC. For this case, the pin corresponds to a power bus, a ground bus, or is unconnected. If the pin is unconnected, the related node 'termX' is ignored (and does not appear in the node list of the circuit) but must be specified. If the pin is a bus, the related node 'termX' is implicitly connected to the terminals PU/PD/PC/GC of the created buffers, according to [Pin Mapping] data, or is ignored if [Pin Mapping] data are not available for this component. If the pin is a buffer, the related node 'termX' is implicitly connected to the terminal IO of the newly created buffer. Depending on the type, other terminals may be required. If terminals IN, EN or OUT are needed, they are implicitly connected to nodes named 'Bname.pin_name.in', 'Bname.pin_name.en' or 'Bname.pin_name.out', respectively. Usually, these nodes must be connected using .connect statements to reflect the internal circuitry of the integrated circuit (see below for an example).

All optional parameters available for IBIS buffers are also supported and are common to all buffers created by the B statement. Only buffer and power are ignored if specified. The value of buffer is internally set for each buffer according to the value of Model_type in the related [Model] definition. The value of power is internally set to ON or OFF for each created buffer depending on the availability of the optional [Pin Mapping] table. If no [Pin Mapping] data are available, power is set to ON and terminals PU/PD/PC/GC are connected to internal voltage sources, whose values are defined in [Model] definitions. If [Pin Mapping] data are available, power is set to OFF and terminals PU/PD/PC/GC are connected to power/ground pins according to the [Pin Mapping] table.

Example

The file `interface.ibs` contains the description of a digital circuit consisting in 4 non-inverting buffers with 3-state outputs controlled by a common enable signal:

```
...
[Component]     LineDriver
[Manufacturer]  Silvaco
[Package]
|               typ         min          max
R_pkg           0.036       0.026        0.041
L_pkg           6.747nH     5.492nH      9.423nH
C_pkg           0.752pF     0.392pF      1.110pF
|
[Pin]   signal_name  model_name  R_pin     L_pin       C_pin
|
1       ce1          IC_IN       0.041     8.654nH     1.110pF
11      I1           IC_IN       0.041     8.654nH     1.110pF
10      I2           IC_IN       0.041     8.654nH     1.110pF
9       I3           IC_IN       0.041     8.654nH     1.110pF
8       I4           IC_IN       0.041     8.654nH     1.110pF
2       O1           IC_OUT      0.041     8.654nH     1.110pF
3       O2           IC_OUT      0.041     8.654nH     1.110pF
4       O3           IC_OUT      0.041     8.654nH     1.110pF
5       O4           IC_OUT      0.041     8.654nH     1.110pF
6       GND          GND         0.041     7.469nH     0.899pF
12      GND          GND         0.041     7.469nH     0.899pF
7       VCC          POWER       0.035     6.006nH     0.748pF
...
```

The following statement:

```
...
bc en in in in in out1 out2 out3 out4 gnd1 gnd1 vcc1
+component='LineDriver' file='interface.ibs'
...
```

creates buffers named `bc.1`, `bc.2`, `bc.3`, `bc.4`, `bc.5`, `bc.8`, `bc.9`, `bc.10` and `bc.11`. No buffers are created for pins `6`, `7` and `12` because `model_name` is a reserved word. Assuming that `IC_IN` and `IC_OUT` are input and three-state models, respectively, several nodes are automatically created for each buffer. For input buffers, the required terminals `'out'` are connected to nodes `'bc.pinname.out'`, for three-state buffers, the required terminals `'en'` and `'in'` are connected to nodes `'bc.pinname.en'` and `'bc.pinname.in'`, respectively. As IBIS specs do not provide any information about the internal circuitry of the component, these nodes must be explicitly connected by adding the following statements in the netlist:

```
...
.connect bc.1.out  bc.2.en  bc.3.en  bc.4.en  bc.5.en
.connect bc.11.out bc.2.in
.connect bc.10.out bc.3.in
.connect bc.9.out bc.4.in
.connect bc.8.out bc.5.in
...
```

If no [`Pin Mapping`] data are given for this component, no nodes are created for `gnd1` and `vcc1` terminals, and all buffers use their internal voltage sources.

If the [`Component`] description contains the following data:

```
...
[Pin Mapping] pulldown_ref pullup_ref gnd_clamp_ref power_clamp_ref
|
1             NC           NC         GND_BUS       PWR_BUS
2             GND_BUS      PWR_BUS    GND_BUS       PWR_BUS
3             GND_BUS      PWR_BUS    GND_BUS       PWR_BUS
4             GND_BUS      PWR_BUS    GND_BUS       PWR_BUS
5             GND_BUS      PWR_BUS    GND_BUS       PWR_BUS
6             GND_BUS      NC
7             NC           PWR_BUS
8             NC           NC         GND_BUS       PWR_BUS
9             NC           NC         GND_BUS       PWR_BUS
10            NC           NC         GND_BUS       PWR_BUS
11            NC           NC         GND_BUS       PWR_BUS
12            GND_BUS      NC
...
```

gnd1 and vcc1 nodes must be connected to appropriate voltage sources to supply all buffers from the terminals PC/GC/PU/PD:

```
Vcc  vcc1 gnd 3.3V
Vgnd vgnd1 gnd 0V
```

7.10 Options

GMIN/DCGMIN conductances are connected in parallel with PC and GC diodes, and with PU and PD Voltage-Controlled Voltage Sources if they exist (type-dependent), to ensure better convergence of buffer devices in particular situations.

The option VZERO=2 can also be specified in netlists containing IBIS buffers. This may help to speed-up computation in some cases.

A new option 'd_ibis' has been added to specify the location of IBIS files.

7.11 Backward Compatibility

In older releases, when IBIS buffers were not supported, a B device was only used to define a MESFET device (as an alias of Z). Now B may be used to define either MESFET devices or IBIS buffers. When a B statement is encountered in a netlist, SmartSpice first checks whether the IBIS-specific parameter file is specified in the element card. As this parameter is required to create an IBIS buffer, SmartSpice creates a MESFET device if it is missing, so that backward compatibility is maintained.

7.12 Limitations

Only DC, Transient and AC analysis are supported for netlists containing IBIS buffers.

The Silvaco IBIS parser is based on the most recent “Golden Parser” source code, which was partially incorporated into SmartSpice. However only *.ibs files are currently supported. *.pkg and *.ebd files are not supported yet. Even though the Silvaco IBIS parser is more permissive than the “Golden Parser” (regarding case-sensitivity and parameter check for example), it is not recommended to use IBIS files that are not fully compliant with IBIS specs (up to version 4.0). All IBIS files should be systematically verified with the most recent release of the “Golden Parser”, which is freely available as an executable on the IBIS Open Forum web site.

Not all data available in IBIS files is actually used by SmartSpice. Basically only [Model] descriptions are required to perform a simulation. However several advanced (but optional)

features (like [Driver Schedule] or [Add Submodel]) are currently ignored by the simulator. This will be implemented in future releases.

7.13 References

1. Peivand F. Tchrani, Yuzhe Chen, Jiayuan Fang, “Extraction of transient behavioral model of digital I/O buffers from IBIS”, *46th IEEE Electronic Components&Technology Conference*, Orlando, May 28-31, 1996, pp 1009-1015
2. Ying Wang, Han Ngee Tan, “The development of analog SPICE behavioral model based on IBIS model”, *Ninth Great Lakes Symposium on VLSI*, pp.101-104, 1999

Chapter 8
Verilog-A Simulation Flow

8.1 Introduction

This chapter describes Verilog-A language, the analog subset of the Verilog-AMS language. Verilog-A belongs to the Analog Hardware Description Language (AHDL) class of computer languages. AHDLs are intended to help design analog systems in high level behavioral forms for continuous systems.

The SmartSpice Verilog-a Interface provides the capability to include in a netlist one or several modules described in Verilog-A.

SmartSpice Interface supports the Verilog-A language. The Verilog-A language allows you to write behavioral models to support either top down design, or trade off speed and accuracy during verification.

8.1.1 Requirements

The Verilog-A Interface (`libVLG`) is supported on Linux Redhat Enterprise 4, 5 and 6 (64 bit). Verilog-A is also supported on Windows XP, Windows Vista and Windows 7. GCC compiler (Mingw GCC) is used in the Verilog-A simulation flow for speedup purposes. The intention of this manual is not to explain in detail all the features of the Verilog-A language. For further language information, consult the Verilog-AMS Language Reference Manual distributed with SmartSpice, found under the SmartSpice **Help** menu.

8.2 Modeling with the Verilog-A Language Overview

The Verilog-A language is a high-level Analog Hardware Description Language (AHDL) that uses modules as the basic component to describe the structure and behavior of analog systems and their components. With the analog statements of Verilog-A, you can describe a wide range of conservative systems and signal-flow systems; such as electrical, mechanical, fluid dynamic, and thermodynamic systems. To simulate systems that contain Verilog-A components, you must have the SmartSpice simulator installed on your system.

To describe a system, specify both the structure of the system and the behavior of its components. In Verilog-A with the SmartSpice circuit simulator, the structures are defined at different levels. At the highest level, you can define the overall system structure in a netlist. At the lower, more specific levels, you can define the internal structure of modules by defining the interconnections among submodules.

8.2.1 Representing a System

A system is a collection of interconnected components that, when acted upon by a stimulus, produce a response. A hierarchical system is a system in which the components are also systems. A primitive component (leaf component) is a component that has no subcomponents. Each primitive component connects to zero or more nets. Each net connects to a signal which can traverse multiple levels of the hierarchy. The behavior of each component is defined in terms of the values of the nets to which it connects. The Verilog-A language allows analog and mixed-signal systems to be described by a set of components or modules.

A signal is a hierarchical collection of nets which, because of port connections, are continuous. The nets for a signal are in the discrete domain called a digital signal. The nets that make up a signal are in the continuous domain; the signal is an analog signal. The net that consists of signals from the continuous and discrete domain is called a mixed signal.

The components interconnect through ports and nets to build a hierarchy, as illustrated in Figure 8-1.

8.2.2 Nets and Nodes

Nets in Verilog-A connect analog signals that are assigned values from a continuous domain. A node is a point of physical connection between nets of continuous-time descriptions. Analog signals are also referred to as nodes.

Nodes obey conservation-law semantics.

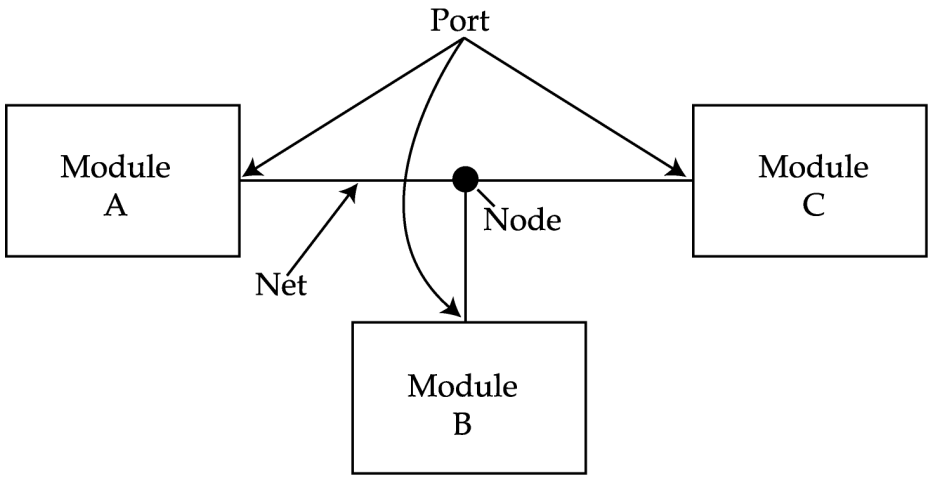


Figure 8-1 Nets and Nodes

8.2.3 Verilog-A Systems

Two types of analog systems can be described with Verilog-A: conservative and signal-flow systems. A conservative type of system, which includes those described by conventional SPICE, incorporates a set of constraints within the system that insures conservation of charges (fluxes and so forth) within the system. Signal flow systems employ a different level of formulation, which focuses only on the propagation of signals throughout the system.

8.2.4 Conservative Systems

A conservative system is a system where two values are assigned to every node: a potential value and a flow value. The potential of a node is shared by all ports and nets connected to it. The flow of a node is such that the sum of all continuous nodes are equal to zero. For this reason, the conservation laws, such as Kirchoff 's Potential Law (KPL) and Kirchoff's Flow Law (KFL), can be applied to every node. KPL and KFL are generalizations of KVL and KCL for electrical systems which allow the conservation laws to be applied to any conservative system.

Conservative Systems	Electrical Systems
KPL	KVL
KFL	KCL

8.2.5 Kirchhoff's Laws

In formulating continuous system equations, Verilog-A uses two sets of relationships. The first are the constitutive relationships which describe the behavior of each component. Constitutive relationships can be kept inside the simulator as built-in primitives, or they can be provided by Verilog-A's module definitions.

The second set of relationships (interconnected relationships) describe the structure of the network. Interconnected relationships, which contain information on how the components are connected to each other, are only a function of the system topology. They are independent of the nature of the components.

SmartSpice Verilog-A simulator uses Kirchhoff's Laws to define the relationships between the nodes and the branches. Kirchhoff's Laws are typically associated with electrical circuits that relate voltages and currents. However, by generalizing the concepts of voltages and currents to potentials and flows, Kirchhoff's Laws can be used to formulate interconnection relationships for any type of system.

Kirchhoff's Laws provide the following properties relating the quantities present on nodes and branches, as shown in Figure 8-2.

- Kirchhoff's Flow Law (KFL): The algebraic sum of all flows out of a node at any instant is zero (0).
- Kirchhoff's Potential Law (KPL): The algebraic sum of all the branch potentials around a loop at any instant is zero (0).

These laws imply that a node is infinitely small; so there is negligible difference in potential between any two points on the node and a negligible accumulation of flow.

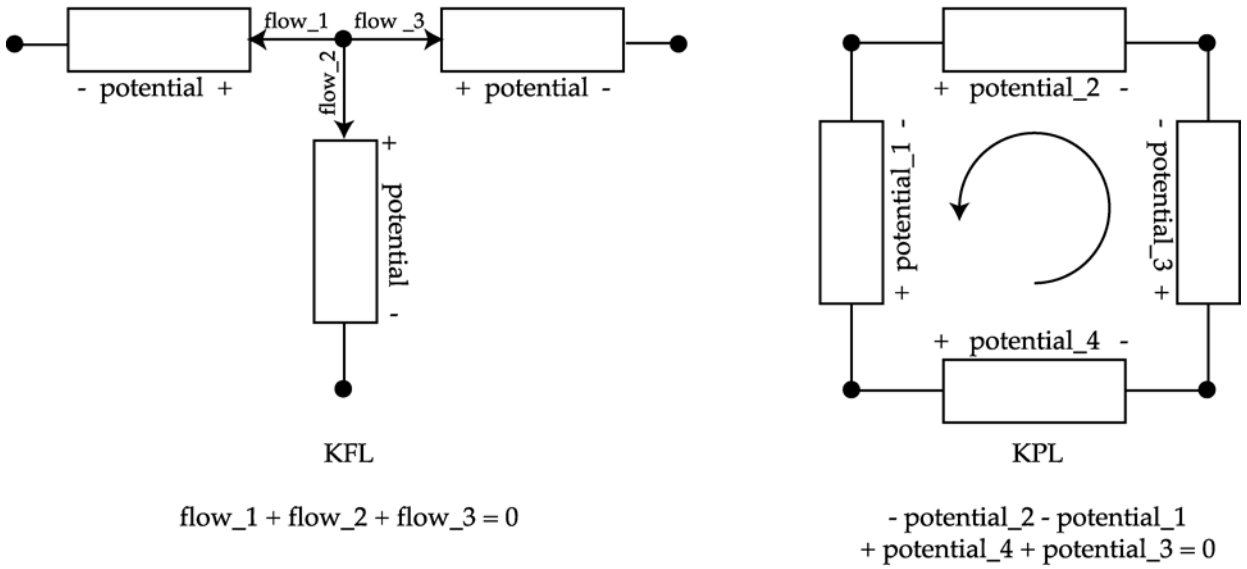


Figure 8-2 Kirchhoff's Flow Law (KFL) and Potential Law (KPL)

8.2.6 Reference Nodes

The potential of a single node is defined with respect to a reference node. The reference node, called ground in electrical systems, has a potential of zero.

8.2.7 Reference Directions

Each branch has a reference direction for the potential and flow (see Figure 8-2). With the reference direction shown, the potential in this schematic is positive whenever the potential of the terminal marked with a plus sign is larger than the potential of the terminal marked with a minus sign. Verilog-A uses associated reference directions. Consequently, a positive flow is defined as one that enters the branch through the terminal marked with the plus sign, and exits through the terminal marked with the minus sign.

8.2.8 Signal-Flow Systems

Signal-flow systems associate only a single value with each node. As a result, a signal-flow port must be unidirectional. If the component has two ports, one port is the input and the other one must be the output.

8.2.9 Mixed Conservative and Signal-Flow Systems

You can model systems that contain a mixture of conservative nodes and signal-flow nodes when practicing the top-down design cycle. It allows the flexibility for the designers to initially use signal-flow models easily in the design cycle, and gradually convert component models to conservative forms as the design progresses.

8.3 Simulation Flow and Configuration Silvaco Verilog-A

8.3.1 Simulation Flow

The main stages of the Verilog-A interface are described below:

1. **Compilation Phase:** During the sourcing of a SmartSpice netlist, Verilog-A files referred to by the `.verilog` command cards are compiled by the Verilog-A Compiler. As a result of the compilation, a C file is produced for each module parsed.
2. **Linkage Phase:** The C files are compiled by a `gcc` compiler, a Dynamically Linkable Library (a `.so` file for Unix or `.dll` for Windows) is produced. This library is then linked to the SmartSpice executable.
3. **Simulation Phase:** Once all Verilog-A modules have been incorporated, the simulation in SmartSpice is done as usual. The compilation, and the linkage are in fact transparent. Simulating a netlist with Verilog-A module is still done the same way as with a traditional netlist: first the netlist is sourced, then a simulation is run, and the results output in a text or graphical form.

The simulator flow with an example of editing Verilog-A file(s) and a SmartSpice input deck, is shown in Figure 8-3.

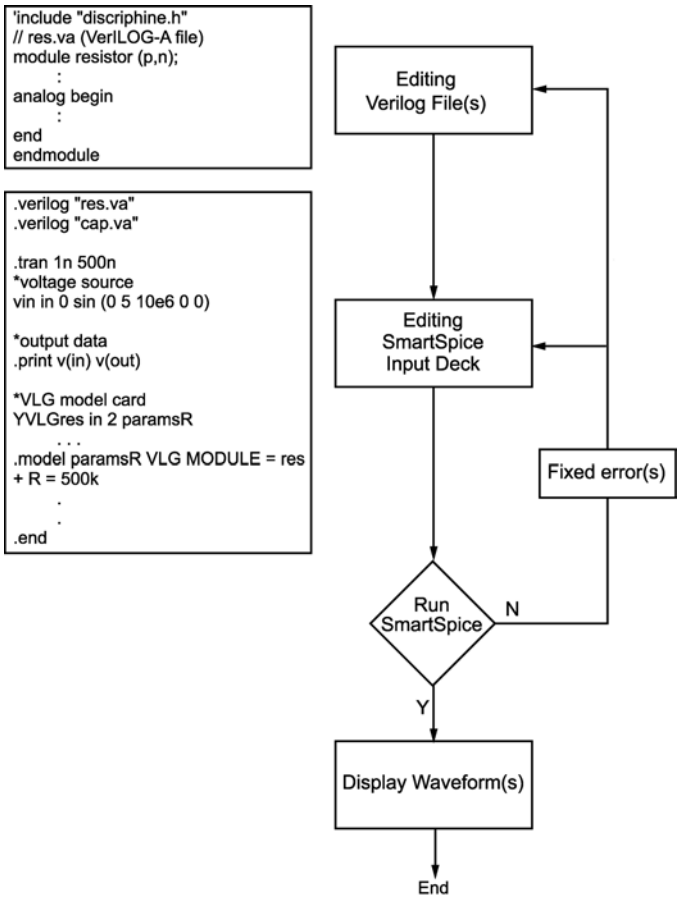


Figure 8-3 Simulation Flow

8.3.2 Encrypted Verilog-A Libraries

To support standard Silvaco automatic understanding of encrypted files, simplify and better organize the Verilog-A device libraries, the Verilog-A translator (application Verilog-A) has been moved to a static library libVerilog-A. Now it’s part of the libVLG library, which is a SPICE dynamic library to support the SmartSpice interface to Verilog-A translator. It’s linked with libVerilog-A.

8.3.3 Support for Encrypted Verilog-A Source Files

libVLG supports both fully encrypted and partially encrypted Verilog-A source files. If Verilog-A source files are encrypted using the standard SmartSpice encryption program SENCRYPT (see the SENCRYPT User's Manual), SmartSpice will be able to automatically understand these files. No special license is necessary.

8.3.4 Default Compiler

GCC is included in the SmartSpice package and is used on Linux platforms, and theMinGW C/C++ compiler is included in the SmartSpice package for use on Windows platforms.

8.4 Interfacing Verilog-A Modules within the SPICE Input Deck

8.4.1 Introduction

Verilog-A modules can be used inside an input deck as follows:

1. `.verilog` statement + Verilog-A module:

```
.verilog “file.va”   The    name of the file that contains the
Verilog-A module(s) that will be used within this input deck.

YVLGpart_name  <port pins>  module_name <parameter values> that are
contained within the file.va.
```

or

2. `.model` card + Verilog-A module instantiation:

```
YVLGpart_name  <port pins>  model_card_ID_name

.MODEL model_card_ID_name VLG MODULE = module_name
+ <parameter values
```

3. X call + Verilog-A module.

```
Xpart_name <port pins> va_module_name
```

4. X call + .model card with Verilog-A module instantiation

```
Xpart_name <port pins> model_card_ID_name

.MODEL model_card_ID_name va_module_name
+ <parameter values>
```

Examples

1.

```
*SPICE deck with module instantiation (with VLG model card)

VIn In0 0 PWL(0,0v .3ns,0v .6ns,1v 2ns,1v 2.3ns,0v )

YVLGtline1 In0 In1 Out1 DUTY
.verilog "tline.va"
.MODEL DUTY VLG MODULE = mva_tline1
ri1 In1 0 1e-12
ro1 Out1 0 1e-12
.tran .1ns 10ns
.save all
.probe v(In0) v(In1) v(Out1)
.end
```

2.

```
*SPICE deck without VLG Model Card

.verilog "tline.va"
VIn In0 0 PWL(0,0v .3ns,0v .6ns,1v 2ns,1v 2.3ns,0v )

YVLGtline1 In0 In1 Out1 mva_tline1
ri1 In1 0 1e-12
```

```
ro1 Out1 0 1e-12
.save all
.tran .1ns 10ns
.probe v(In0) v(In1) v(Out1)
.end
```

8.5 Instantiating Analog Primitives (SmartSpice Specific)

These sections describe how to instantiate some SmartSpice-specific SPICE primitives in the Verilog-A modules, and shows the primitives available with their name, port names and parameter names.

8.5.1 SPICE Model Card Primitives

The following shows an example of how a model card can be instantiated from Verilog-A.

```
.MODEL mosnpn NPN BF=85 IS=1E-10 RB=100 VAF=50
+ CJE=3PF CJC=2PF CJS=2PF TF=0.3NS TR=6NS

module srs_module(c1, b1, e, b2, c2);
electrical c1, b1, e, b2, c2;
mosnpn Q1 (c1, b1, e);
endmodule
```

Unlike SPICE, the first letter of the instance name, in this case `Q1` and `Q2`, is not constrained by the primitive type.

The ports passed by name is not supported in current SmartSpice.

8.5.2 SPICE Subcircuit Primitives

The following shows an example of how a subcircuit can be instantiated from Verilog-A.

Subcircuit definition in the SPICE netlist file:

```
.subckt inverter in out l=8u w=8u
mp out in vdd vdd pmos l='l' w='w'
mn out in gnd gnd nmos l='l' w='w'
.ends inv
Subcircuit instantiation in the Verilog-A source file:
module modva_inv(in, out);
inout in;
electrical out;
parameter real l=5u, w=5u;

// instantiates subcircuit 'inv' defined in the previous SPICE
netlist.

inverter #(l,w) s1(in,out); // l=5u, w=5u;
endmodule // modva_inv
```

8.5.3 R Element (Resistor)

Spice syntax

```
Rxxx n+ n- <mname> <R | RES=>resval > <TC1=val> <TC2= val>
```

Verilog-A syntax

```
resistor #(.r(resval),tc1(val), tc2(val)) xx (p,n);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
resistor	p, n	n+, n-	r, tc1, tc2	r | res, tc1, tc2

8.5.4 C Element (Capacitor)

Spice syntax

```
Cxxx n+ n- <mname> <C | CAP=>capval  <IC=val>
```

Verilog-A syntax

```
capacitor #(.c(lval),ic(val)) xx (p,n);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
capacitor	p, n	n+, n-	c, ic	c | cap, ic

8.5.5 L Element (Inductor)

Spice syntax

```
Lxxx n+ n- <mname> <L | IND=>lval  <IC=val>
```

Verilog-A syntax

```
inductor #(.l(lval),ic(val)) xx (p,n);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
inductor	p, n	n+, n-	l, ic	l | ind, ic

8.5.6 V/I Elements (Independent voltage/current sources)

Independent source with exponential type "EXP"

Spice syntax

```
Vxxx n+ n- <<DC> dcval> EXP (c1 c2 <td0 <tau0 <td1 <tau1>>>>)
+ <acmag <acphase>>>
Ixxx n+ n- <<DC> dcval> EXP (c1 c2 <td0 <tau0 <td1 <tau1>>>>)
+ <acmag <acphase>>>
```

Verilog-A syntax

```
vexp #(.dc(val), .mag(val),.phase(val),.val0(val),.val1(val),
.td0(val),.tau0(val),.td1(val),.tau1(val)) xx (p,n);
iexp #(.dc(val), .mag(val),.phase(val),.val0(val),.val1(val),
.td0(val),.tau0(val),.td1(val),.tau1(val)) xx (p,n);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
vexp / iexp	p, n	n+, n-	dc, mag, phase, val0, val1, td0, tau0, td1, tau1	dc, acmag, acphase, c1, c2, td0, tau0, td1, tau1,

Independent source with pulse type "PULSE"

Spice syntax

```
Vxxx n+ n- <<DC> dcval> PULSE (c1 c2 <td <tr <tf <pw <per>>>>)
+ <acmag <acphase>>>
Ixxx n+ n- <<DC> dcval> PULSE (c1 c2 <td <tr <tf <pw <per>>>>)
+ <acmag <acphase>>>
```

Verilog-A syntax

```
vpulse
#(.dc(val),.mag(val),.phase(val),.val0(val),.val1(val),.td(val),
.rise (val),.fall(val),.width(val) ,.period(val) ) xxx (p,n) ;
ipulse
#(.dc(val),.mag(val),.phase(val),.val0(val),.val1(val),.td(val),
.rise(val),.fall(val),.width(val),.period(val)) xxx (p,n) ;
```

Primitive	Verilog-A port name	Spice port name	Verilog-A parameter name	Spice parameter name
vpulse / ipulse	p, n	n+, n-	dc, mag, phase, val0, val1, td, rise, fall, width, period	dc, acmag, acphase, c1, c2, td, tr, tf, pw, per

Independent source with piecewise linear pairs type "PWL"

Spice syntax

```
Vxxx n+ n- <<DC> dcval> PWL (t1 v1 <t2 v2 t3 v3 ... tk vk>)
+ <acmag <acphase>>>
Ixxx n+ n- <<DC> dcval> PWL (t1 v1 <t2 v2 t3 v3 ... tk vk>)
+ <acmag <acphase>>>
```

Verilog-A syntax

```
vpwl #(.dc(val), .mag(val),.phase(val),.wave('{ t1 v1 <t2 v2 t3 v3
... tk vk>) })) xx (p,n);
ipwl #(.dc(val), .mag(val),.phase(val),.wave('{ t1 v1 <t2 v2 t3 v3
... tk vk> ) })) xx (p,n);
```

Primitive	Verilog-A port name	Spice port name	Verilog-A parameter name	Spice parameter name
vpwl / ipwl	p, n	n+, n-	dc, mag, phase, wave	dc, acmag, acphase, (t1 v1 <t2 v2 t3 v3 ... tk vk>)

Independent source amplitude modulation type "AM"

Spice syntax

```
Vxxx n+ n- <<DC> dcval> AM <(> <sa <offset <fm <fc <td>>>>> <)>
+ <acmag <acphase>>>
Ixxx n+ n- <<DC> dcval> AM <(> <sa <offset <fm <fc <td>>>>> <)>
+ <acmag <acphase>>>
```

Verilog-A syntax

```
vsine #(.dc(val),.mag(val),.phase(val),.ampl(val),.offset(val),
.ammodfreq(val),.freq (val),.td(val)) xxx (p,n) ;
isine #(.dc(val),.mag(val), .phase(val),.ampl(val),.offset(val),
.ammodfreq(val),.freq(val),.td(val)) xxx (p,n) ;
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
vsine / isine	p, n	n+, n-	dc, mag, phase, offset, ampl, freq, td, ammodfreq,	dc, acmag, acphase, offset, sa, fc, fd, fm

Independent source sinusoidal, type "SIN"

Spice syntax

```
Vxxx n+ n- <<DC> dcval> SIN (co ca <freq <td <theta <phase>>>>)
+ <acmag <acphase>>>
Ixxx n+ n- <<DC> dcval> SIN (co ca <freq <td <theta <phase>>>>)
+ <acmag <acphase>>>
```

Verilog-A syntax

```
vsine
#(.dc(val),.mag(val),.phase(val),.ampl(val),.offset(val),.td(val),
.freq (val),.damp(val),.sinephase(val)) xxx (p,n);
isine
#(.dc(val),.mag(val),.phase(val),.ampl(val),.offset(val),.td(val),
.freq(val),.damp(val),.sinephase(val)) xxx (p,n);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
vsine / isine	p, n	n+, n-	dc, mag, phase, offset, ampl, freq, td, damp, sinephase	dc, acmag, acphase, co, ca, freq, td, theta, phase

Independent source single-frequency, type "SFFM"

Spice syntax

```
Vxxx n+ n- <<DC> dcval> PWL (t1 v1 <t2 v2 t3 v3 ... tk vk>)
+ <acmag <acphase>>>
Ixxx n+ n- <<DC> dcval> PWL (t1 v1 <t2 v2 t3 v3 ... tk vk>)
+ <acmag <acphase>>>
```

Verilog-A syntax

```
vsine #(.dc(val), .mag(val),.phase(val),.ampl(val),.offset(val),
.td(val),.freq(val),.fmmodindex(val),.tmmodfreq(val)) xxx (p,n) ;
isine
#(.dc(val),.mag(val),.phase(val),.ampl(val),.offset(val),.td(val),
.freq(val),.fmmodindex(val),.tmmodfreq(val)) xxx (p,n) ;
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
vsine	p, n	n+, n-	dc, acmag, acphase, offset, ampl, freq, td, fmmodindex, tmmodfreq	dc, mag, phase, co, ca, fc, td, mdi, fs

8.5.7 T Element (Transmission Line, Lossless)

Spice syntax

```
Txxx n1 n2 n3 n4 Z0 | Zo=val TD=val
Txxx n1 n2 n3 n4 Z0 | Zo=val  F=val <NL=val>
```

Verilog-A syntax

```
tline #(.z0(val),td(val)) xx (t1,b1,t2,b2);
tline #(.z0(val),.f(val),. nl(val)) xx (t1,b1,t2,b2);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
tline	t1, b1, t2, b2	n1, n2, n3, n4	z0, td, f, nl	z0 | zo=, td, f, nl

8.5.8 G Element (Voltage-Controlled Current Source)

Spice syntax

```
Gxxx n+ n- <VCCS> nc1+ nc1- transconductance
```

Verilog-A syntax

```
vccs #(.gm(val)) xx (sink,src,ps,ns);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
vccs	sink, src, ps, ns	n+, n-, nc1+, nc1-	gm	transconductance

8.5.9 E Element (Voltage-Controlled Voltage Source)

Spice syntax

```
Exxx n+ n- <VCVS> nc1+ nc1- gain
```

Verilog-A syntax

```
vcvs #(.gain(val)) xx (p,n,ps,ns);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
vcvs	p, n, ps, ns	n+, n-, nc1+, nc1-	gain	gain

8.5.10 D Element (Diode)

Spice syntax

```
Dxxx n+ n- mname <area>
```

Verilog-A syntax

```
diode #(.area(val)) xx (a,c);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
diode	a, c	n+, n-	area	area

8.5.11 Q Element (Bipolar Junction Transistor - BJT)

Spice syntax

```
Qxxx nc nb ne <ns>  mname <area>
```

Verilog-A syntax

```
bjt #(.area(val)) xx (c,b,e,s);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
bjt	c, b, e, s	nc, nb, ne, ns	area	area

8.5.12 M Element (MOSFET)

Spice syntax

```
Mxxx nd ng ns <nb> mname <lval | L=val> <wval | W=val> <AD=val>
+ <AS=val> <PD=val> <PS=val> <NRD=val> <NRS=val>
```

Verilog-A syntax

```
mosfet #(.w(val), .l(val), .ad(val), .as(val), .pd(val), .ps(val),
.nrd(val), .nrs(val)) xx (d,g,s,b);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
mosfet	d, g, s, b	nd, ng, ns, nb	w, l, ad, as, pd, ps, nrd, nrs	wval | w, lval | l , ad, as, pd, ps, nrd, nrs

8.5.13 J Element (JFET/MESFET)

Spice syntax

```
Jxxx nd ng ns <nb> mname <area>
```

or

```
Bxxx nd ng ns <nb> mname <area>
```

or

```
Zxxx nd ng ns <nb> mname <area>
```

Verilog-A syntax

```
jfet #(.area(val)) xx (d,g,s);
mesfet #(.area(val)) xx (d,g,s);
```

Primitive	Verilog-A port	Spice port	Verilog-A parameter	Spice parameter
jfet/ mesfet	d, g, s	nd, ng, ns	area	area

See the SPICE Models Manual for a complete description of element models

8.6 Compact Modeling Extensions (SmartSpice Specific)

8.6.1 Introduction

This section will describe the SmartSpice Verilog-A specific implementation of compact modeling extensions.

A compact model (CM) is a mathematical description of the electrical behavior of a semiconductor device, suitable for simulation of electronic circuits.

Models, as implemented in SPICE simulators, are a set of implemented model equations which allow you to access and define the parameter values externally. The definition of the model equations, and the method for extracting the parameters, is called "`modeling`".

Many such different device models have been developed at universities, and through internal company research efforts, over the years for various technologies. Some of these models, when parameterized, match devices produced by manufacturers. It is these models that eventually become "industry standards".

It is these "industry standard" CM's that get implemented within the SPICE simulation program. Two issues arise from this process. The first is no documented agreements containing the technical specifications or other criteria used, such as guidelines, definitions, and application domain. The second issue is that you must wait for commercial SPICE vendors to implement these compact models within their respective codes.

Verilog-A compact models open many areas of simulation to SPICE users that were formally closed due to the above process.

Extensions of Verilog-AMS are at present being defined by a special sub-committee that is investigating how Verilog-AMS can better support compact modeling.

8.6.2 Parameters

Parameter Units and Description

Attributes for modules, ports, nodes, parameters and variables are available for generating module documentation. The available attributes are:

- Description and units for ports, nodes, parameters and variables.
- Description for modules.

The syntax of these attributes is:

```
attribute_instance::=
  (* attr_spec{, attr_spec} *)

attr_spec::=
```

```
  attr_name = constant_expression

attr_name::=
  desc
| units
```

These attributes can be specified at the beginning of modules, ports, nodes, parameters and variables declaration:

```
{attribute_instance}
 module module_name;

{attribute_instance}
 [in|out|inout] list_of_nets;

{attribute_instance}
 electrical list_of_nets;

{attribute_instance}
 parameter type parameter_identifier = constant_expression;

{attribute_instance}
 type variable_identifier = constant_expression;
```

If several ports, nodes, parameters or variables are declared using the same declaration statement, the same attributes are applied to all these elements.

Example

```
(*desc = “BSIM3v3.2.4 MOS model”*)
module mosfet(d, g, s, b);
inout d,g,s,b;

(*desc = “drain”*)
electrical d;
(*desc = “gate”*)
electrical g;
(*desc = “source”*)
electrical s;
(*desc = “bulk”*)
electrical b;

(*desc = “Gate oxide thickness”, units = “m”*)
parameter real TOX = 150.0e-10;

(*desc = “Threshold voltage”, units = “V”*)
real vth;
```

String Parameters

String parameter can be set using the `string` keyword as a type specifier:

```
string_parameter_declaration ::=
parameter string param_identifier = string
                                    [from    {string [, string]}]
                                    [exclude {string [, string]}];
```

Domain restriction can be applied to the value of a string parameter:

- If the `from` keyword is used, followed by a list of strings, the value of the string parameter is compared to this list. If the string parameter value doesn't appear in this list, an error message is returned.
- If the `exclude` keyword is used, followed by a list of strings, the value of the string parameter is compared to this list. If the string parameter value appears in this list, an error message is returned.
- If no domain restriction is specified, no verification is done.

The declared string parameter can be used with == and != operators in an analog block. It can be compared to literal strings and can be displayed using `$strobe` or `$debug` functions.

Example

```
parameter string TYPE   = “nmos” from {“nmos”, “pmos”};
parameter string FILE   = “output.dat” exclude {“”};

...
if (TYPE == “nmos”)
    u0 = 0.067;
else
    u0 = 0.025;
```

Parameter Aliases

The keyword `aliasparam` can be used in parameter declaration to create parameter aliases.

The syntax is as follows:

```
aliasparam alias_identifier = parameter_identifier;
```

The `parameter_identifier` must be a previously declared valid parameter. Once declared, the alias has the same data type as the parameter it references. The alias value can then be set in:

- The `.model` card in the SmartSpice netlist.
- The instance line in the SmartSpice netlist.
- The instantiation of the module in another Verilog-A module.

If the alias value is specified, the parameter it references takes this value. The alias cannot be used in an analog block, only the parameter can be used.

Example

In SmartSpice netlist:

```
.model nmos VLG module=mosfet vaf = 1.2 // In Verilog-A description
file
(va is equal to 1.2)

parameter  real va  = 1.0;
aliasparam      vaf = va;

analog
  begin
    temp = va; // only va can be used, not vaf
...
```

8.6.3 Environment Parameter Functions

Simulator specific parameters can be accessed in Verilog-A description with the `$simparam` system task. The syntax is as follows:

```
$simparam("simulator_option_name")
```

or

```
$simparam("simulator_option_name", value_to_return)
```

`$simparam` returns the value of the defined `simulator_option_name`. The difference between these two syntaxes is the error handling. The first syntax returns an error if the simulator parameter is unknown. The second syntax does not generate an error and returns the specified value `value_to_return`.

The possible parameters name and the corresponding SmartSpice options are:

Table 8-1 $simparam() options names and corresponding SmartSpice options

simulator_option_name	SmartSpice options	Description
gmin	GMIN/ DCGMIN	$simparam returns DCGMIN parameter value if an OP/DC analysis is running and GMIN if it is a TRAN analysis.
scale	SCALE	Scale factor for device parameters.
scalm	SCALM	Scale factor for model parameters.
tnom	TNOM	Nominal temperature of the circuit.
iteration	-	$simparam returns the current Newton-Raphson iteration number.
gdev	GNODE	Conductance value connected to every circuit node to improve convergence properties.
imax	EXPLI	Explosion current value.
simulatorVersion	-	SmartSpice simulator version.
simulatorSubversion	-	SmartSpice simulator sub-version.

Example

```
gmin = $simparam("gmin");
$debug ("iteration = %g", $simparam("iteration"));
```

8.6.4 Hierarchical Structure

Paramset Statement

The `paramset` hierarchical structure provides a convenient and powerful way to collect parameter values for a given module, like in a SmartSpice `MODEL` card. In its structure, the `paramset` looks like a module, but it contains no behavioral code, and is always associated with another `paramset` or module. Its syntax is:

```
paramset_declaration::=
paramset paramset_identifier module_or_paramset_identifier;
paramset_item_declaration {paramset_item_declaration}
```

```
paramset_statement        {paramset_statement}
endparamset

paramset_item_declaration::=
  parameter_declaration
| local_parameter_declaration
| string_parameter_declaration
| integer_declaration
| real_declaration

paramset_statement::=
.module_parameter_identifier = expression;
| statement
```

To use a `paramset` with a module, the name of the module must be precise in lieu of `module_or_paramset_identifier`. `paramset_identifier` must be replaced by the name of the `paramset`. In the following example, `nmos_proc` is the name of the `paramset` and `mosfet` the module name:

```
module mosfet (d,g,s,b);
...
endmodule

paramset nmos_proc mosfet;
...
endparamset
```

The `mosfet module` is then instantiated by giving the name of the `paramset` at instantiation statement.

- If the module is instantiated in SmartSpice indeck file:

  `YVLGmos1 d g s b nmos_proc`

- If the module is instantiated in another Verilog-A module:

  `nmos_proc mos1 (d,g,s,b);`

A `paramset` has parameters and statements. These statements are only assignment statements, not behavioral code in an analog block. The statement:

```
.module_parameter_identifier = expression;
```

is used to pass values to the parameters of the associated module. The right-hand side expression of the assignment can be composed of numbers, parameters, local parameters or variables declared in the paramset. References to parameters of any other module can also be done.

In the following example, if `mosfet` module looks like:

```
module mosfet (d,g,s,b);
  ...
  parameter real L    = 5.0e-6;
  parameter real W    = 5.0e-6;
  parameter real AS   = 0.0;
  parameter real AD   = 0.0;
  ...
  parameter real NSUB = 6.0e16;
  ...
endmodule
```

the `paramset` could be:

```
paramset nmos_proc mosfet;
  parameter real L=1u from [0.25u:inf);
  parameter real W=1u from [0.2u:inf);
  .L  = L;
  .W  = W;
  .AD = W * 0.5u;
  .AS = W * 0.5u;
  ...
  .NSUB = 1.3e17;
endparamset
```

and the instantiation line could be:

```
YVLGmos1 d g s b nmos_proc L=0.3u W=0.6u
```

From the SmartSpice simulator's point of view, the idea is to gather all the parameters common to a process in a single `paramset`. This `paramset` can be seen from outside as a black box, whose only inputs are the `instance` parameters (usually geometrical parameters as defined in SmartSpice compact models). These `instance` parameters will be implemented as paramset parameters. In the previous example, only the geometrical parameters L, W are accessible at instantiation.

The equivalent SmartSpice internal model of the previous example would be:

```
m1 d g 0 0 mosfet L=0.3u W=0.6u
.MODEL mosfet nmos level=81 nsub=1.3e17 ...
```

Restrictions on `paramset` statements and assignments are similar to the restrictions for analog functions.

Paramset Overloading

Multiple `paramsets` can be declared using the same paramset identifier, and they may refer to different modules. Typically, these `paramsets` may each contain process parameters specific to geometry ranges. During simulation, `paramset` parameter values are checked against their domain restriction specifications. The equivalent functionality in SmartSpice compact models is binning.

For example, if two `paramsets` are declared with the same name:

```
paramset nmos_proc mosfet;
  parameter real L    = 0.25u from [0.25u:1u);
  parameter real W    = 1u    from [0.2u:inf);
  .U0   = 640;
...
paramset nmos_proc mosfet;
  parameter real L  = 1u from [1u:inf);
  parameter real W  = 1u from [0.2u:inf);
  .U0   = 650;
...
```

and the instantiation line is:

```
YVLGm1 d g 0 0 nmos_proc L=0.28u W=0.4u
```

the values passed to `L` and `W` parameters at instantiation are checked against the domain ranges of each overload of the paramset `nmos_proc`. For the `L` parameter, that check is first `[0.25u:1u)` and `[1u:inf)`. Since `L=0.28u` satisfies the first domain range, the first overload will be chosen.

More precisely, the appropriate `paramset` is chosen according to the following rules:

- All parameters specified at instantiation are `paramset` parameters.
- All parameters specified at instantiation are within the allowed ranges specified in the `paramset` parameter declaration.
- All the local parameters of the `paramset` are within the allowed ranges specified at declaration (a local parameter is a special parameter which is not accessible from instantiation line).

If these rules are not sufficient for choosing the correct `paramset`, the following additional rules are used by the simulator to solve conflicts:

- The `paramset` with the fewest number of un-overridden parameters is selected.
- The `paramset` with the greatest number of range-specified local parameters is selected.

`paramset` functionality provides also a new kind of parameter: the `local` parameter. A `local` parameter is declared the same way as a parameter except that the `localparam` keyword is used in lieu of `parameter` keyword:

```
localparam real AREA = L*W from [0.25u:1u);
```

The difference with a parameter is that a local parameter value can not be overridden at instantiation; its value is checked against the given range, so it can be used for overload resolution.

Example

```
module mosfet (d,g,s,b);
  ...
  parameter  real L    = 5.0e-6;
  parameter  real W    = 5.0e-6;

  ...
  parameter real NSUB = 6.0e16;
  parameter real U0   = 670;
  ...
endmodule

paramset nmos_proc mosfet;
  parameter real L     = 0.25u from [0.25u:1u);
  parameter real W     = 1u    from [0.2u:inf);
  localparam real AREA = L*W   from [0.05p:1p);

  .L    = L;
  .W    = W;
  .NSUB = 1.3e17;
  .U0   = 640;
endparamset

paramset nmos_proc mosfet;
  parameter real L     = 1u  from [1u:inf);
  parameter real W     = 1u  from [0.2u:inf);
  localparam real AREA = L*W from [0.2p:1p);

  .L    = L;
  .W    = W;
  .NSUB = 1.3e17;
  .U0   = 650;
endparamset
```

At instantiation:

```
YVLGm1 d g 0 0 nmos_proc L=0.27u W=0.4u
```

L=0.28u: The first overload of nmos_proc paramset will be chosen because L parameter domain restriction in first overload is [0.25u:1u).

```
nmos_proc #(.L(1.2u), .W(0.4u)) m1 (.d(d), .g(g), .s(0), .b(0))
```

L=1.2u: The second overload of nmos_proc paramset will be chosen because L parameter domain restriction in second overload is [1u:inf).

```
nmos_proc #(.L(0.35u), .W(4u)) m1 (.d(d), .g(g), .s(0), .b(0))
```

L=0.35u and localparam AREA=1.4p: L parameter value fits the domain range of the first paramset overload but localparam AREA is not in the domain range. No overload is selected and an error message is returned.

8.6.5 Hierarchical System Functions

Hierarchical system parameters are implicitly declared for every module. They allow you to set hierarchical properties of an instance or a paramset. These parameters are:

parameter name	description	default value	allowed values
$mfactor	multiplicity factor (number of identical instances placed in parallel)	1	$mfactor > 0
$xposition	horizontal offset of the center of the instance in meters	0	any
$yposition	vertical offset of the center of the instance in meters	0	any
$angle	angle of the instance rotation in degrees and in counter-clockwise direction	0	0 < $angle < 360
$hflip	instance horizontally mirrored about its center (+1: not mirrored, -1: mirrored)	1	$hflip = +1 or -1
$vflip	instance vertically mirrored about its center (+1: not mirrored, -1: mirrored)	1	$vflip = +1 or -1

The value of these parameters can be overridden using a defparam statement, a module parameter value assignment by name, or in a paramset, like other parameters.

Examples

```
defparam x0.$mfactor = 10, x0.$angle=90;
...
mosfet #( .$mfactor(10), .$angle(90) ) mos1 (d,g,s,b);
...
paramset nmos_009_proc mosfet;
.$mfactor   = 10;
.$angle     = 90;
endparamset
```

Their value can be accessed within a module simply by using their names. The returned value is the specified value of the current instance combined with the hierarchical parameter of the parent instance, with the following rules:

parameter name	value returned
`$mfactor`	$mfactor$_{parent}$ * $mfactor$_{specified}$
`$xposition`	$xposition$_{parent}$ + $xposition$_{specified}$
`$yposition`	$yposition$_{parent}$ + $yposition$_{specified}$
`$angle`	($angle$_{parent}$ + $angle$_{specified}$) modulo 360
`$hflip`	$hflip$_{parent}$ * $hflip$_{specified}$
`$vflip`	$vflip$_{parent}$ * $vflip$_{specified}$

The contribution to a branch flow quantity in an instance is multiplied by the value of `$mfactor` calculated hierarchically from the top of the design to the current instance.

8.6.6 Hierarchical Detection Functions

$param_given

The `$param_given` system task purpose is to determine if you specified a parameter value in the simulation environment.

Syntax

```
$param_given(parameter_name)
```

This system task must be used in an analog block. It returns 1 if `parameter_name` has been given, and 0 if otherwise. `parameter_name` is detected if it is set in:

- The `.model` card in the SmartSpice netlist.
- The instance line in the SmartSpice netlist.
- The instantiation of the module in another Verilog-A module.

Example

Verilog-A description file:

```
module mosfet(d,g,s,b);
...
parameter real uo = 600.0;
...
analog begin
  if ($param_given(uo))
    begin
       kp = uo * tox;
    end
```

`$param_given` will return 1 if the parameter `uo` is set with:

```
.model nmos VLG module=mosfet uo=650.0
(.model card in the SmartSpice netlist)
```

or

```
YVLGm1 1 2 3 4 nmos uo=650.0
(instance line in the SmartSpice netlist)
```

or

```
mosfet #(.uo(650.0)) M1(d,g,s,b);
(instantiation in another Verilog-A module)
```

$port_connected

$port_connected function returns 1 if the port in argument has been given in the instance line of the SmartSpice netlist. It returns 0 if the port has been omitted.

Syntax

```
$port_connected(port_name)
```

Example

```
analog
  begin
    if (!$port_connected(bulk))
       V(bulk, source) <+ 0.0;
...
connects bulk and source ports together if bulk has been omitted.
```

8.6.7 Limiting Functions

The purpose of $limit system function is to limit the Newton-Raphson per-iteration value change of a potential or a flow. A built-in limiting function or a user-defined limiting function can be used. The syntax of $limit is:

```
limited_value    =    $limit(access_function_reference,    string,
arguments_list)
```

The returned value is the limited value to be used in analog block for further calculation. access_function_reference is the voltage or the current the limitation is applied to. It must be a potential or a flow quantity.

Example

```
vgs = TYPE * $limit (V(gate,sourcep), fetlim, von);
```

The second argument string determines the limiting method. If this argument is pnjlim or fetlim, the SPICE-like simulators limiting function pnjlim or fetlim is used.

Example

```
analog begin
  von   = TYPE * VT0;
  vcrit = ‘CONSTvt0 * ln(‘CONSTvt0 / (‘CONSTroot2*1.0e-14));

  vgs   = TYPE * $limit (V(gate,sourcep), fetlim, von);
  vbs   = TYPE * $limit (V(bulk,sourcep), pnjlim, ‘CONSTvt0, vcrit
);
...
```

- The pnjlim function is intended for limiting arguments to exponential. Its arguments are the step size vte and the critical voltage Vcrit. vte is usually the product of the thermal voltage $vt and the emission coefficient of the junction. Vcrit is generally obtained from the formula $V_{crit} = vte \cdot \ln(vte/(\sqrt{2} \cdot I_s))$ where I_s is the saturation current

of the junction. These arguments must be given right after the limiting method, in `arguments_list`.

- The `fetlim` function is intended for limiting the potential across the oxide of a MOS transistor. Its argument is generally the threshold voltage of the MOS transistor. It must be given in the `arguments_list`.

If `string` is not `pnjlim` or `fetlim`, then the simulator assumes that the limiting function is defined by you and declared in the current module. This limiting function must follow rules in its definition and in its content:

- In the definition of this function, the first two arguments must be the current value of branch quantity to limit and the value of branch quantity at previous iteration. When calling `$limit`, the first two arguments are passed automatically by the simulator.

Example

```
analog function real DEVlimvds;
    input VdsNew, VdsOld;
    real  VdsNew, VdsOld;

    if(VdsOld >= 3.5)
      begin
        if(VdsNew > vold)
          begin
    ...
endfunction

analog begin
    vds = TYPE * $limit(V(d, s), DEVlimvds);
...
```

- The user-defined function must return the limited value.

Example

```
analog function real DEVlimvds;
    input VdsNew, VdsOld;
    real  VdsNew, VdsOld;

    if(VdsOld >= 3.5)
      begin
        if(VdsNew > VdsOld)
          begin
            limitedValue = MIN(VdsNew,(3*VdsOld)+2);
    ...
    // DEVlimvds returns the result of the limitation
    DEVlimvds = limitedValue;
endfunction
```

- In the code of the function, the `$discontinuity(-1);` line must be called if the quantity has been effectively limited to indicate the simulator to perform more iterations.

Example

```
analog function real DEVlimvds;
    input VdsNew, VdsOld;
    real  VdsNew, VdsOld;

    if(VdsOld >= 3.5)
      begin
        if(VdsNew > VdsOld)
          begin
            limitedValue = MIN(VdsNew,(3*VdsOld)+2);
    ...
    if (limitedValue != VdsNew)
        $discontinuity(-1);

    DEVlimvds = limitedValue;
endfunction
```

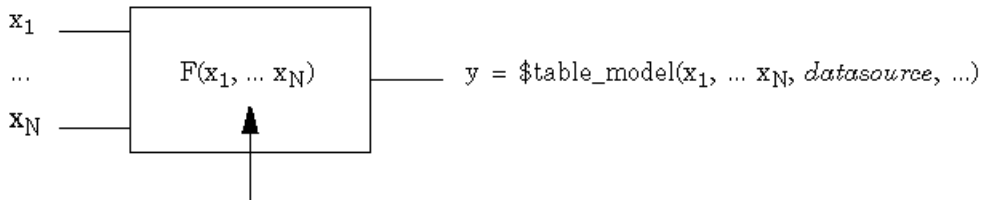
- Additional arguments can be used and must be placed after the first two ones in the function declaration. When calling `$limit`, these additional arguments are specified in `arguments_list`.

Example

```
analog function real DEVlimvds;
    input VdsNew, VdsOld, type;
    real  VdsNew, VdsOld, type;
    ...
endfunction

analog begin
    vds = $limit(V(d, s), DEVlimvds, TYPE);
```

8.6.8 Interpolation Function

`$table_model()` function performs a linear interpolation/extrapolation according to an array of sample points. A sample point i (x_{i1}, ... x_{iN}, y_i) verifies y = F(x_1, ... x_N), with F the function to interpolate:



The dimension of interpolation N can be 1, 2 or 3 (linear, bilinear or trilinear interpolation). `$table_model()` function returns the value interpolated at current input values x_1, ... x_N.

The syntax of `$table_model()` function is as follows:

```
$table_model(inputs, datasource, control_string);
```

`inputs` are the expressions used as interpolation independent variables x_1, ... x_N. The output `y=F(`x_1`, ... `x_N`)` of the `$table_model()` function is calculated according to these values. It will be a set of any legal variable or branch quantity. The number of table inputs determines the dimension of the interpolation.

`datasource` specifies the source of sample points. It can be a external text file or a set of one-dimensional arrays:

If the source is an external text file, each sample must be written in this file as a sequence of numbers (x_1 ... x_N y) on a single line, where x_k is the sample point in k'th dimension, and y the F(x_1, ... x_N) value at this sample point. Each number must be separated by one or more spaces or tabs, and must be a real or an integer number. The samples can be stored in any order. Lines beginning with # are comments and are ignored. The external text file source is selected if the name of the file containing the sample points is given.

An example of text file content (file name is `table_1D.tbl`):

```
# sample source file: table_1D.tbl
# x1      x2      y
  6       0       0.0
  6       1       0.216
  6       2       0.432
```

```
  8       0       0.0
  8       1       0.512
  8       2       1.024
  10      0       0.0
  10      1       1
  10      2       2
...
```

In this case, the call to `$table_model()` function in the Verilog-A module is:

```
Iab = $table_model(V(a), V(b), “table_1D.tbl”, “LE,E”);
```

In this example, `Iab=F(V(a),V(b))` is interpolated between the samples recorded in `table_1D.tbl` at the current potential values `V(a)` and `V(b)`. The file `table_1D.tbl` must be in the same directory as the Verilog-A module.

If the source is a set of one-dimensional arrays, the samples must be stored so that, for the i^{th} sample (x_{i1} ... x_{iN} y_i):

1st array	...	N’th array	(N+1)’th array
array1_name[i]=X_{i1}	...	arrayN_name[i]=X_{iN}	arrayY_name[i] = Y_i = F(X_{i1}, ... X_{iN})

The previous example translated in one-dimensional array format gives:

```
x1[0] = 6;      x2[0] = 0;      y[0] = 0.0;
x1[1] = 6;      x2[1] = 1;      y[1] = 0.216;
x1[2] = 6;      x2[2] = 2;      y[2] = 0.432;
x1[3] = 8;      x2[3] = 0;      y[3] = 0.0;
x1[4] = 8;      x2[4] = 1;      y[4] = 0.512;
x1[5] = 8;      x2[5] = 2;      y[5] = 1.024;
x1[6] = 10;     x2[6] = 0;      y[6] = 0.0;
x1[7] = 10;     x2[7] = 1;      y[7] = 1;
x1[8] = 10;     x2[8] = 2;      y[8] = 2;
```

The one-dimensional arrays are then passed to `$table_model()` function as a comma-separated list:

```
y = $table_model(V(a), V(b),
                 array1_name, ... arrayN_name, arrayY_name,
                 “”);
```

In the example:

```
Iab = $table_model(V(a), V(b), x1, x2, y, “”);
```

The array source is selected if the list of the one-dimensional arrays is given.

If the input values x_1, ... x_N are not in the domain bounded by sample points, F(x_1, ... x_N) has to be extrapolated. `table_control_string` is composed of substrings separated by commas with a maximum of 3 substrings. Each substring sets the extrapolation method to use for a particular dimension.

The available extrapolation methods are L (linear) or E (error). With the linear method, a linear extrapolation is performed, extending the nearest piecewise linear interpolation region to the requested point. With the error method, an error is reported if the requested point is beyond the interpolation region.

The substring can be composed by 2 following letters: L or E. In this case, requested extrapolation methods are applied respectively at lowest and highest coordinate values. If the

substring has only 1 letter L or E, the requested method is applied to both coordinate values. If no letter is given, the default method (linear) is applied to both coordinate values.

Examples

control_string	1st dimension lower coordinate extrapolation method	1st dimension higher coordinate extrapolation method	2nd dimension lower coordinate extrapolation method	2nd dimension higher coordinate extrapolation method
“”	Linear	Linear	Linear	Linear
“E,EL”	Error	Error	Error	Linear
“,LE”	Linear	Linear	Linear	Error

8.6.9 Compiler Directives

Predefined Macros for Simulator Specification

One text macro is defined internally in the SmartSpice Verilog-A interface:

```
‘define __VAMS_COMPACT_MODELING__ 1
```

Testing the `SILVACO_SMARTSPICE` macro with `‘ifdef` preprocessing directive means to test if the SmartSpice simulator is used. Testing the `__VAMS_COMPACT_MODELING__` macro means to test if the compact modeling extensions are implemented in the simulator.

Example

```
‘ifdef __VAMS_COMPACT_MODELING__
    gmin = $simparam(“gmin”,1);
‘endif
```

8.6.10 Interfacing Verilog-A Compact Model Code to the Input Deck

Verilog-A models are identified by the `YVLG` prefix followed by the device name (e.g., `YVLGmodel_name`). The `model_name.va` file must also be referenced within the SPICE deck.

Syntax

```
.VERILOG "model_name.va"
YVLGinst_name < ports> model_name params
```

For full description of the interface Verilog-A and the SPICE input deck, see Section 8.4“Interfacing Verilog-A Modules within the SPICE Input Deck”.

8.6.11 Interfacing Verilog-A Compact Model Code to a Schematic Symbol

Create a symbol as specified within the Gateway User's Manual, 5.7.1 Opening and Creating Symbols and Appendix C Symbol Files.

Using the **Symbol Editor**, press the SmartSpice button to access the SmartSpice String Editor. Within this window, type the following string:

```
YVLGmodel_name @Path %P1 %P2 %P3…..%Pn <ParamsID> param1=@param1
param2=@param2 paramn=@paramn
```

Where `%Pn` represents pin names, and `paramn=@paramn` allows values for `paramn` to be entered from the schematic.

The SPICE deck must contain the following:

```
.VERILOG "model_name.va"
```

and the:

```
.MODEL ParamsID VLG MODULE = model_name
*** Process Related Model Parameters
+ COX    = 3.45E-3
+ XJ     = 0.15E-6
.
.
.
```

8.7 Options for Verilog-A

8.7.1 How to Set Options for Verilog-A

Three different methods are available for specifying Verilog-A options when running SmartSpice. Another method is available when the Verilog-A translator is run manually.

When running SmartSpice, methods A, B and C may be used individually or in combination:

A. Environment variable:

```
SILVACO_VERILOGA_ARGS <option-string>
```

B. SmartSpice ".OPTION" card:

```
.OPTION VERILOGA-ARGS="<option-string>"
```

C. SmartSpice ".verilog" card:

```
.verilog "<file-name-string>" [<option-string>*]
```

which will be expanded to:

```
.verilog "<file-name-string>" $ENV $OPT.
```

<option-string> is a concatation of one or more options separated by spaces.

Note: Items inside [] are optional.

Note: $ENV expands to `SILVACO_VERILOGA_ARGS` value.

Note: $OPT expands to `.OPTION veriloga-args` value

When you make an option change which affects the translation/compilation of any verilog source file (e.g., adding "-no_opt") the system automatically re-translates and re-compiles the affected source files.

Example 1

```
setenv SILVACO_VERILOGA_ARGS "-no_opt"
```

The environment variable specifies only one option, "-no_opt".

Example 2

```
.OPTION VERILOGA-ARGS="-no_protected_math"
```

The ".option" card in the circuit specifies option "-no_protected_math".

Example 3

```
.verilog "MyNewModel.va2" -debug
```

The verilog source file "MyNewModel.va2" is to be translated and compiled with both the environment and option card plus debugging (e.g., "-no_opt -no_protected_math -debug").

8.7.2 Verilog-A Options

Performance Options

libVLGP engages all available optimizations which improve performance. The following options disable one or more optimizations to reduce translation/compilation time.

"-no_opt"	(No optimizations) Use this option for fastest Verilog-A translation and compiling. This option disables all optimizations.
"-no_recompile_for_collapsed_nodes"	(No nodes collapsing) Use this option to skip nodes collapsing in setup stage which may cause recompilation.
"-no_cse"	Do not use CSE algorithm to optimize the code.
"-no_cvr"	No constant variable reduction.
"-no_cp"	No constant propagation.
"-no_pe"	No parameter expression pre-calculation

Simulation Options

"-debug"	Enables the Verilog-A Debugger to debug the modules applied with this option
"-check_finite"	This option will enable checking results of expressions for NAN and infinite during simulation and report to the user.
"-no_protected_math"	Certain math functions are protected against invalid arguments (asin, acos, pow, sqrt). This can be switched off by using this option.

8.8 Usage of the .PRINT and .PLOT SmartSpice Commands

It is possible to refer to Verilog-A objects from the netlist with either `.PRINT` or `.PLOT` commands. Objects that are accessible from the netlist are as follows:

- Sources and probes of all potential or flow quantities.
- Real variables defined with attribute and all parameters.

Objects that are not yet accessible are:

- Potential or flow quantities not used as a source or a probe.
- Objects in a module instantiated from another Verilog-A module.

Example

In the module `I_level_shift` in the following listing, `I(out)` and `I(in)` are the only quantities to be available through the print command. The potential quantities `V(out)` and `V(in)` are not accessible.

Verilog-A module:

```
`include "discipline.h"

 module I_level_shift(in,out);
 input in;
 output out;
 electrical in, out;
 parameter real iout_offset = 0;
 // parameter can be accessed by .PRINT
 (* desc="demo" *)real demo ;
 // real variable "demo" can be accessed by .PRINT
 analog begin
 demo = I(in) + iout_offset;
 I(out) <+ demo; //I(out) and I(in) can be used in the
 //.PRINT and .PLOT commands
 // V(out) and V(in) are not accessible
 end
 endmodule
```

SmartSpice input deck:

```
* print VA parameter check
 .verilog "shift.va"
 v1 1 0 dc 1.0
 r1 2 0 10
 YVLGbig 1 2 I_level_shift
 + iout_offset=0.2
 .dc v1 0.0 1.0 0.2
 .print @YVLGbig[I(out)] @YVLGbig[demo] @YVLGbig[iout_offset]
 .end
```

8.9 Verilog-A Error and Warning Messages

The following section is a list of messages displayed when encountering errors. Only a subset of the error and warning messages are listed and explained. Examples are given to clarify the warning or error messages, but are not the only situation for which the message might occur. These messages are listed in alphabetical order.

Two types of error messages might occur:

- **Parsing Errors** (or warnings)**:** They are syntax or semantic errors related to the Verilog-A language, and have the following format:

```
<file name> at line <number>: Error:
<error message>
```

Example

```
"test1.va" at line 16: Error:
  Unknown system task '$last_cross'.
```

Once the error has been fixed (here in the previous example, `'$last_cross'` should have been changed to `'last_crossing'`), re-source the netlist so SmartSpice will re-compile the Verilog-A source file.

- **Simulation Errors** (or warnings)**:** Run-time errors found during the SmartSpice simulation. Some of these errors are domain or range checks that indicate that some Verilog-A equations are incomplete for some specific input values. It is your responsibility to create a model where all possible input values are processed correctly. These error messages have the following format:

```
Error: (VERILOGA): <error message>
```

Example

```
Error: (VERILOGA): Can not access verilog input file : 'test9.va'
```

8.9.1 Parsing Errors

- "analysis analog function requires string arguments."

Example

```
if ( analysis(AC) ) // ERROR: argument must be "AC"
```

- "Argument `<number>` of the concatenation operation has an unmatched type.": Concatenation element type does not match the other element types.
- "Assignment statement type mismatch.": The type of a right hand-side does not match the type of a left hand-side for a variable procedural assignment.

Example

```
integer val1;
val1 = "string val"; // ERROR: integer type expected in the right
hand-side.
```

- "Cannot assign value to genvar <name>.": Genvar variables can only be incremented in for-loops.
- "Can not evaluate size of vector <identifier> for port association.": Expression of a vector range definition must be a constant expression (numbers or parameters).
- "Can not re-declare access attribute in a derived nature.": Access attributes can only be declared in a base nature.

- “Domain error with constant argument for operator <operator name>.”: Argument is outside the domain of a mathematical operator.
- “Formal parameter <parameter name> not found in module instantiation.”: A parameter does not exist in the instantiated module.
- “Illegal type expression for parameter initialization.”: Real or integer type are only supported for parameter.
- “Implicit node <node name> cannot be a vector.”: Implicit nodes (nodes referenced in module instantiation statement that are not declared) can only be scalar.
- “Invalid access <function name> for branch <branch name>.”

Example

```
electrical node1;
F(node1) <+ R*I(node1);  // ERROR: access function F is not valid.
```

- “No Branch quantity expression found.”: Internal compiler error. See previous Verilog-A parsing errors.
- “No Code Generation Method for Expression <expression>.”: Internal compiler error. See previous Verilog-A parsing errors.
- “Node <node name> is a vector. Index expression required.”: Access to a vector element requires a bracket operator.
- “Nodes <node name> and <node name> are not compatible.”: Two nodes referenced in a branch do not have the same discipline.

Example

```
electrical node1;
thermal node2;
V(n1,n2) <+ cos(ph);     // ERROR: n1 and n2 have incompatible
disciplines.
```

- “Ordered parameter assignment in analog primitive instance <instance name> not allowed.”

Example

```
resistor #(200)r1(in,tmp); // ERROR:parameter assignment must be by
name
                           // ex:.p(200)
```

- “Parameter <name> not found in analog primitive <name> for instance <name>.”: Parameter not found in an analog primitive (SPICE primitive, SPICE subcircuit, SPICE model card).

Example

```
$ Smartspice netlist
.SUBCKT srlc2 in out w=10u
R1 in tmp 200
L1 tmp out 125m
C1 out 0 1u
.ENDS srlc
// Verilog-A source file
srlc1 #(.c(1u))  s1(out, grnd);

"wrongSubcircuit.va" at line 12: Error:
```

```
  Parameter 'c' not found in analog primitive 'srlc1' for instance
's1'.
```

- "Port <port name> can not be declared as ground node."
- "Preprocessor Error: Arguments mismatched for macro <macro name>": The number of actual arguments does not match the number of arguments specified during a macro function definition.

Example

```
'define MY_MAX(x,y,z) if(x > y) then x + z else y + z
val = 'MY_MAX(val0, val1); //ERROR: 3 arguments expected.
```

- "Preprocessor Error: Can not redefine <macro name>, already used as directive name.": The macro name specified in the `'define` directive is already used. Rename macro name.
- "Preprocessor Error: Could not find file <file name>.": The compiler could not add the file specified in the `'include` directive. Check file name.
- "Preprocessor Error: No arguments found during macro expansion.": The number of actual arguments does not match the number of arguments specified during a macro function definition.

Example

```
'define MY_MAX(x,y,z) if(x > y) then x + z else y + z
val = 'MY_MAX; //ERROR: 3 arguments expected.
```

- "Recursivity loop found in instantiation statement for module <module name>.": A module can not have a statement that instantiates itself.
- "Right operand of bitwise operator <operator name> should be integer."
- "Seed argument must be an integer variable.": The seed argument of random number generator functions must be an integer.
- "Syntax error near <character>.": This means that the compiler is unable to determine the exact cause of the error. To find the problem, look at the referenced line syntax. Look also at the preceding line to see if there is anything wrong with it, such as a missing semicolon. For example, the following module is missing a semicolon in line 9.

Example

```
'include "discipline.h"
module probe_v2(vout, vin_p, vin_n);
input vin_p, vin_n;
output vout;
electrical vout, vin_p, vin_n;
analog begin
$strobe("hi") // ERROR! Missing semicolon.
$strobe("lo");
V(vout) <+ V(vin_p,vin_n);
end
endmodule
```

- "<task name> system task requires a string as first argument.": Wrong type for the first argument of a task.

Example

```
$strobe(int_val); // ERROR: string argument expected instead of
integer
```

- "<task name> system task requires an integer variable as first argument (channel id).": Wrong type for the first argument of a task. Example:

```
  $fstrobe("=== first line ==="); // ERROR: missing channel id
argument
```

- "Range error with constant argument for operator <operator name>.
- "Redefinition of nature <nature name>.": Nature name must be unique.
- "Unexpected character <character>.": Lexical error. Non-valid character found.
- "Unknown discipline or type `'electrical'`. To use `'electrical'` as a discipline, please include the standard file `'discipline.h'`.": The line `'include "discipline.h"` has to be added at the beginning of the file so standard disciplines can be referenced.
- "Wrong keyword in domain specification."
- "Wrong number of arguments during function call.": In a user analog function call, the actual arguments number does not match the number of formal arguments.

Example

```
analog function real chopper;
      input sw, in;
V(if0) <+ gain * chopper(V(rf)); // ERROR: 2 arguments expected.
```

8.9.2 Parsing Warnings

- "Contribution statement inside analog event block. Results may be incorrect.": To avoid convergence problems, contribution statements should be executed at each iteration of the SmartSpice simulation. For this reason, it is preferable to set only variables in event blocks or if-then statements.

Example

```
 if ( V(Control) > 1 )
          I(a_T,b_T) <+ V(a_T,b_T)/Ron; // WARNING
```

- "Internal node <node name> not used. The design might not converge.": Every internal node must be referenced at least once in a contribution statement. This will avoid having a SPICE singular matrix. In an hierarchical design, the condition must be true at one hierarchy level only.
- "No instance name in hierarchical name <name>. Ignored.": No instance name was found in `defparam` statement.

Example

```
defparam top.p = 2; // WARNING: top is not an instance name.
```

- "`'$fmonitor'` system task has no string as second argument. Ignored."

8.9.3 Simulation Errors

- (VERILOGA): <file name> <line number>: Array index out of bound, index value is <number> for a range definition [<number>:<number>]

This Run-time check failed because of an out-of-bound access to a vector element. Make sure the array index is within the range definition.

- (VERILOGA) “Can not access verilog input file <file name>”: A file specified in a `.verilog` SmartSpice card could not be found.
- (VERILOGA) “Compilation of <file name> ... failed”: The Verilog-A compilation of a file failed. Check previous parsing error messages.
- (VERILOGA) “error in `absdelay` for <expression> delay amount argument has to be a positive number (was <number>)”: Delay argument in the `absdelay` analog function must be a positive number.
- (VERILOGA) “<file name>, line <number>: Argument to <function name> function outside domain range (<number>), returning sqrt(0.0)”: Run-time domain error, make sure that arguments are within the function’s domain.
- (VERILOGA) “<file name>, line <number>: division operand equals zero, returning 1.0: Run-time check for division operand.
- (VERILOGA) “Failed finding model <ident name>.”: See previous parsing error messages.
- (VERILOGA) “in instance <name>, parameter <name> outside its validity range.”: Run-time check for the parameter value inside its validity range.

Example

```
parameter real param1 = 5.0;
// ERROR : default value is outside validity range.
parameter  real  param2 = param1 exclude [-50.0:50);
```

- (VERILOGA) “Model <ident name> from file <file name> already exits as a model card.”: Duplicate model name. Look at all Verilog-A module names declared in the used Verilog-A files and at all VLG model cards. Suppress or rename duplicates.
- (VERILOGA) “Model <ident name> initialization FAILED (see previous messages).”: See previous parsing error messages.
- (VERILOGA) “model <model name> of analog primitive not found.”: See previous Verilog-A parsing error messages. See previous simulation error messages.
- (VERILOGA) “Not enough channel files to open file: <file name>. Limit is 31.”: Too many opened files.
- (VERILOGA) “when setting real parameter id <number> with value <value> for analog primitive <name>: <error message>”:
- (VERILOGA) “While parsing model <model name> declaration: No `MODULE` parameter specified.”: The keyword `MODULE` is missing during a VLG model card declaration.
- (VERILOGA) “Wrong number of ports in input deck for instance <ident name>. Model <ident name> requires <number> port(s). <number> found(s).”: The number of ports in the netlist for a `YVLG` device does not match the number of ports in the corresponding Verilog-A module definition.

Example

```
$ In the smartspice netlist
YVLGper
1 2  zero_div  $ ERROR: Only 1 port expected.
```

```
// In the verilog-A source file
module zero_div(in);
input in;
```

8.9.4 Simulation Warnings

• (VERILOGA): The previous warning(s) on line <number> occurred in a derivative expression automatically generated by the compiler. Results might be incorrect.

Since the SmartSpice Verilog-A Interface uses the derivative method to fill in the SPICE conductance matrix, only mathematical expressions that contain derived values within the function's domain should be used. This avoids convergence problems during simulation.

Example

```
p1B1 = W*cos(a1)*pow((fi_d-fi_sl),2.0)/
        (pow(d_dep,2.0)*(tan(a1)+tan(a2)))*miu_sa1_var*eps_ox;

Warning: (VERILOGA):  'B2_pondere_5_temp.2.va', line 247 : Division
operand equals zero, returning 1.0
Warning:  (VERILOGA):    the  previous  warning(s)  on  line  '247'
occurred in a derivative expression automatically generated by the
compiler.Results might be incorrect.
```

8.10 Verilog-A Keywords

Keywords are predefined, non-escape identifiers which are used to define the language constructs. Preceding a keyword with an escape character (\) causes it to be interrupted as an escaped identifier.

All keywords are defined in lowercase only. Keywords are reserved identifiers used to delimit the language constructs.

Verilog HDL (IEEE 1364) Keywords Supported

begin	case	default	else
end	endease	endfunction	endmodule
for	forever	function	if
initial	inout	input	module
output	repeat		

Verilog-A (AMS 2.3) Keywords Supported

abs	absdelay	abstol	access
acos	acosh	ac_stim	analog
analysis	asin	asinh	atan
atan2	atanh	bound_step	branch
ceil	continuous	connectrules	cos
cosh	cross	ddt	ddt_nature
delay	discrete	discipline	discontinuity
domain	driver_update	endconnectrules	enddiscipline
endnature	exclude	exp	final_step
flicker_noise	floor	flow	from
generate	genvar	ground	hypot
idt	idtmod	idt_nature	inf
initial_step	integer	laplace_nd	laplace_np
laplace_zd	laplace_zp	last_crossing	limexp
ln	log	max	min
nature	noise_table	parameter	potential
pow	real	sin	sinh
slew	sqrt	strobe	tan
tanh	timer	transition	units
white_noise	zi_nd	zi_np	zi_zd
zi_zp			

Verilog HDL (IEEE 1364) Keywords NOT Supported (but are still Reserved Identifiers)

always	and	assign	buf
bufif0	bufif1	casex	casez
cmos	deassign	defparam	disable
edge	endprimitive	endspecify	endtable
endtask	event	force	fork
highz0	highz1	ifnone	join
large	macromodule	medium	nand
negedge	nmos	nor	not
notif0	notif1	or	pmos
posedge	primitive	pull0	pull1
pulldown	pullup	rcmos	reg
release	realtime	rnmos	rpmos
rtran	rtranif0	rtranif1	scalared
small	specify	specparam	strong0
strong1	supply0	supply1	table
task	time	tran	tranif0
tranif1	tri	tri0	tri1
triand	trior	trireg	vectored
wait	wand	weak0	weak1
while	wire	wor	xnor
xor			

Chapter 9
Verilog-A Debugger

9.1 Environment Setup

9.1.1 Invoking Verilog-A Debugger

To debug a Verilog-A module, the following two main steps need to be followed:

1. Source the netlist with Verilog-A module:

   There are two alternative ways to do that:

A. In the SmartSpice window, check the pull-down menu **File→Source for Verilog-A Debugger**. Then source an input deck in regular way.

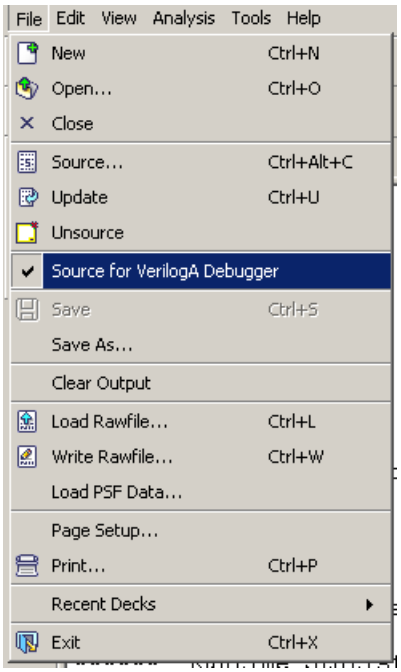


Figure 9-1 Source an Input Deck for Verilog-A Debugger

B. Source the netlist with **File→Source** after setting the Verilog-A debug mode:

- Set Verilog-A option `-debug` for the Verilog-A file. There are three ways to set it:
  - in the netlist with a `.options` command:

    ```
    .options veriloga-args="-debug"
    ```

  - in the netlist with a `.verilog` command:

    ```
    .verilog "<file-name-string>" -debug
    ```

  - Set environment variable `SILVACO_VERILOGA_ARGS` to `-debug`. Example under UNIX:

    ```
    %setenv  SILVACO_VERILOGA_ARGS  -debug
    ```

2. Launch Verilog-A debugger with loaded netlist:

Figure 9-2 Run Verilog-A Debugger Icon

After the input deck is loaded, if there are Verilog-A modules that can be debugged, the **Verilog-A Debugger** menu item under **Tools** will be activated.

Click this menu item or the **Run Verilog-A Debugger** icon (Figure 9-2) to run a simulation with the Verilog-A debugger.

9.1.2 Default Breakpoints

When running the debugger for the first time, breakpoints are set by default on the first statement of the first Verilog-A module.

9.2 Windows Layout

The layout of the main Verilog-A debugger window is displayed in Figure 9-3.

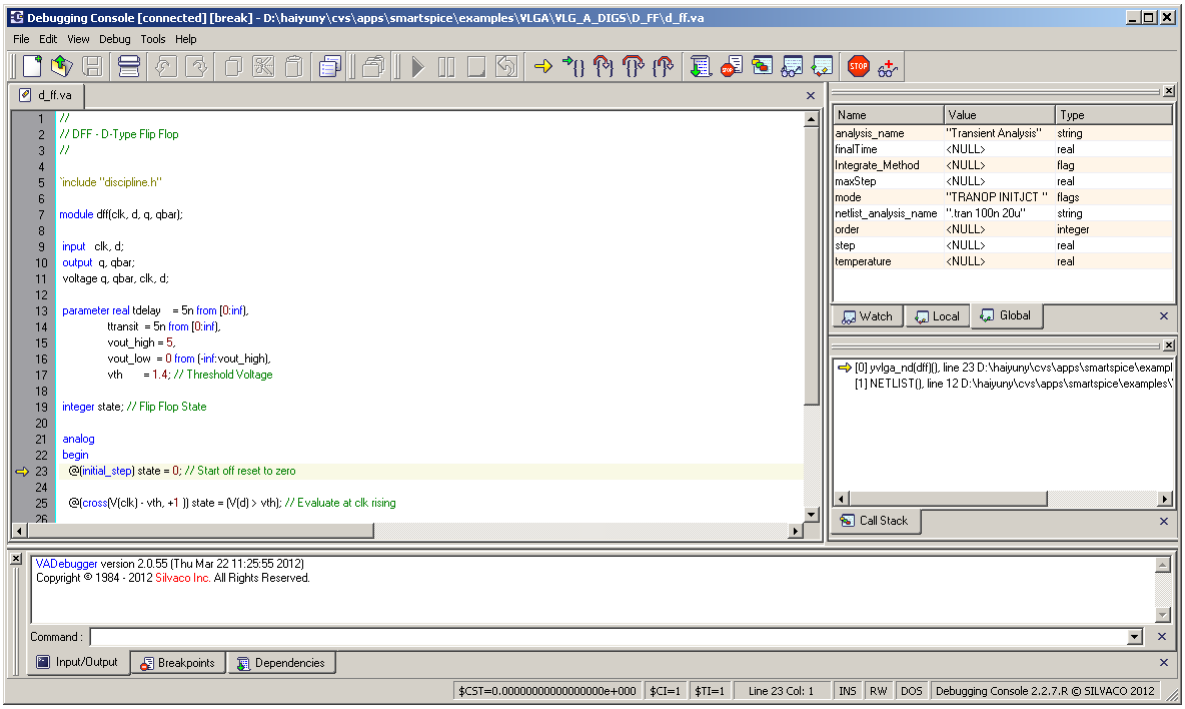


Figure 9-3 Verilog-A Debugger Main Windows Layout

This window will pop up automatically when steps in Section 9.1.1“Invoking Verilog-A Debugger” are followed. The window is divided into 7 sub-windows:

1. **The Pull-down Menu:** Provides a set of commands on files or debugging commands. See Section 9.3“The Pull-down Menu”.
2. **The Toolbar Panel:** Provides debugging commands. See Section 9.4“The Toolbar Panel”.
3. **The Text Editor Window:** This window shows the source code of the current Verilog-A module that is executed by SmartSpice. Features of this windows are discussed in Section 9.5“The Text Editor Window”.
4. **The Call Stack Window:** This display shows the instance hierarchy of the module executed and shown in the Text Editor window. See Section 9.7“The Call-Stack Window” for more details on this window.
5. **The Command Area Window:** This window acts as a console for every events occurring during the debugging. You can also send commands through a command line.
6. **The Simulation Status Bottom Panel:** This box displays the current status of the simulation and the location of the cursor in the Text Editor window. See Section 9.10“The Bottom Panel: Simulation Status and Version Number” for more information.
7. **The Version Number Bottom Panel:** The version number of the debugger is displayed in this window.
8. **The Variable Window:** It will show local, global and watch variables. These windows are described in more detail in Section 9.8“The Variables Window”.

9.2.1 Toggling Windows

Each sub-window with an **X** box (remove tab on Figure 9-5, which is located on a corner of the frame) can be removed from the main window. To restore a window, right click on the main frame, and a menu will appear (Figure 9-4). Check the box of the window you want to restore.

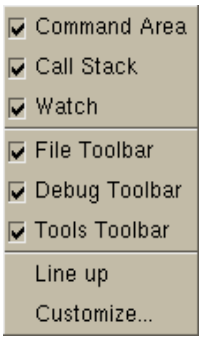


Figure 9-4 Check Box Windows Menu

9.2.2 Resizing Windows

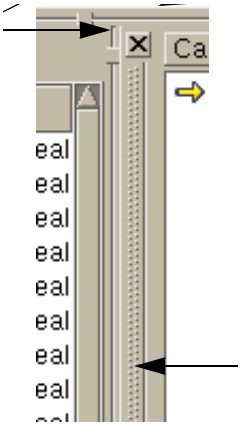


Figure 9-5 Resizing and Moving Windows

The resizing tabs shown in Figure 9-5 allows you to resize each window. Hold and drag the vertical or horizontal tabs to the desired new window size.

9.2.3 Moving Windows

The layout of the windows can be changed. To select a window to be moved, double-click on the moving tab displayed in Figure 9-5. The selected window can then be dragged to a new location.

9.3 The Pull-down Menu

9.3.1 File Menu

The **File** menu (Figure 9-6) allows you to manipulate the files displayed in the Text Editor. The file commands are the following:

- **New:** Create a new file, open an existing file, or close the current file.
- **Save:** Save the current file.

- **Page Setup:** Setup the page layout and print the current file.
- **Recent Files:** Switch to a file recently displayed in the Text Editor.
- **Exit:** Exit the debugger. The remaining simulation will be automatically executed in optimized mode.

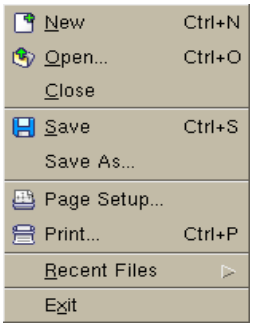


Figure 9-6 File Pull-down Menu

9.3.2 Edit Menu

The **Edit** menu (Figure 9-7) allows you to:

- **Undo** or **Redo** previous commands.
- **Cut**, **Copy**, **Paste** or **Delete** a selected text region.
- **Select All** the text.
- **Find** and **Replace** a string in the Text Editor.
- Go to a line location.
- Open the **Properties** dialog.

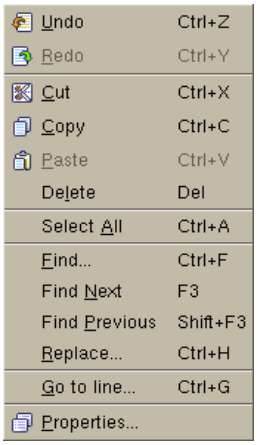


Figure 9-7 Edit Pull-down Menu

9.3.3 Debug Menu

The commands from the **Debug** menu (Figure 9-8) are:

- **Show Next Statement:** Displays the next statement to be executed by the debugger.
- **Continue:** Continues the execution until the next breakpoint or end of simulation.
- **Step Into:** The debugger executes only the next statement and will step inside a VERILOG-A user function if the statement includes function calls.

- **Step Over:** The debugger executes only the next statement. It will not step inside function calls.
- **Step Out:** The debugger finishes execution of a current function and stops at the next statement of the function call.
- **Run to Cursor:** Currently Not Available.
- **Toggle Breakpoint:** Add or remove breakpoint at the location of the cursor in the Text Editor (see Figure 9-10).
- **Add Watch:** Displays a dialog to enter a variable name to be added to the watch list.

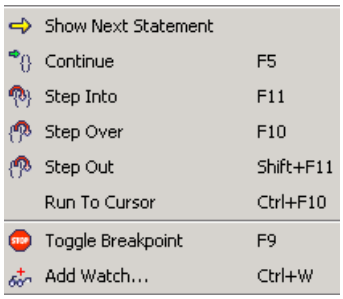


Figure 9-8 Debug Pull-down Menu

9.4 The Toolbar Panel

The Debugger command buttons are displayed in Figure 9-9 (Commands in the **Toolbar** panel are similar to the commands from the **Debug Pull-down** menu described in Section 9.3.3“Debug Menu”).

Figure 9-9 The Toolbar Panel

The button commands are the following:

- **New Document:** Open a new file in the Text Editor.
- **Open Document:** Open an existing file in the Text Editor.
- **Save Document:** Save the current file in the Text Editor.
- **Print:** Print current file in the Text Editor.
- **Undo:** Undo last command.
- **Redo:** Redo last command.
- **Copy:** Copy a highlighted text region into the clipboard.
- **Cut:** Cut a highlighted text region into the clipboard.
- **Paste:** Insert the clipboard at the current location in the Text Editor.
- **Compile Verilog-A file:** Currently Invalid.
- **Start Simulation:** Start the SmartSpice simulation.
- **Pause Simulation:** Pause the SmartSpice simulation.
- **Stop Simulation:** Currently Invalid
- **Restart Simulation:** Currently invalid
- **Show Statement:** Displays the next statement to be executed by the debugger.
- **Continue:** Continues the execution until the next breakpoint or end of simulation.
- **Step Into:** The debugger executes only the next statement, and will step inside a Verilog-A user function if the statement includes function calls.
- **Step Over:** The debugger executes only the next statement. It will not step inside function calls.
- **Step Out:** The debugger finishes execution of current function and stop at next statement of the function call.
- **Dependencies:** Verilog-A files and functions in current circuit.
- **View Breakpoint:** Displays the Breakpoint dialog (see Figure 9-11).
- **Show Call Stack:** Displays current call stack.
- **Show Watches:** Display the variables in watch list.
- **Show Local Variables:** Display the local variables.
- **Toggle Breakpoint:** Add or remove a breakpoint at the location of the cursor in the Text Editor (see Figure 9-10).
- **Add Watch:** Displays a dialog to enter a variable name to be added to the watch list.

9.5 The Text Editor Window

The Text Editor shows the current Verilog-A module being executed. An example of the Text Editor is shown in Figure 9-10.

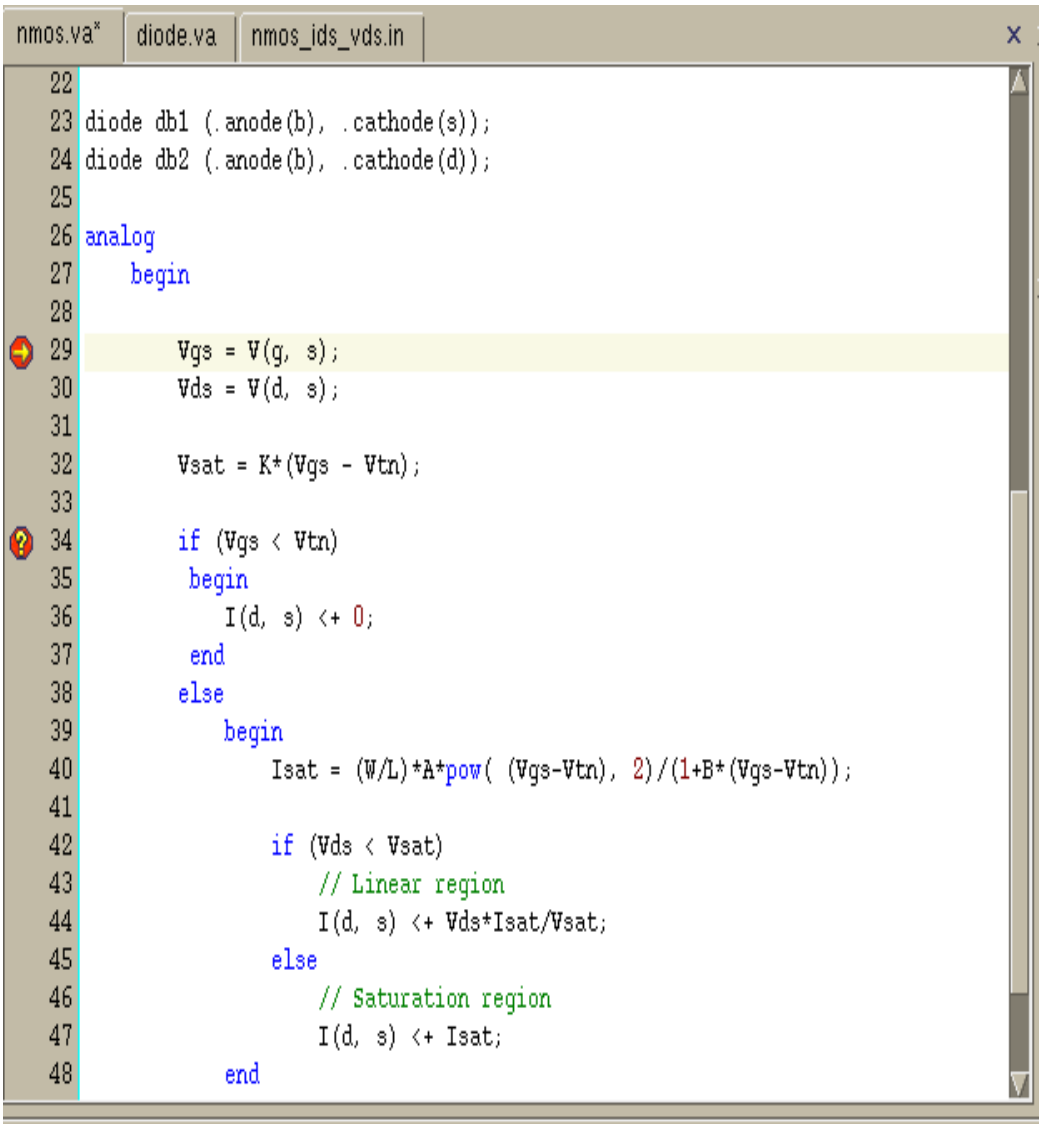


Figure 9-10 Text Editor Window

9.5.1 Text Editor Properties

Multi Files Capability

All files recently loaded in the Text Editor are quickly accessible through a tabulation system. For each new file open in the Text Editor, a new tab is automatically added. A '*' character (star) is append to the tab file name when a save is needed.

Line Numbering

To toggle the line numbering on or off, bring up the **Properties Dialog** window (**Edit→Properties...**). In the Properties dialog tree on the left side of the window, select **Source**, **General**, and then check the **Show Line Number** box. Hit the **Apply** button in the right corner to validate your change.

Color Highlight Syntax

The color highlight syntax can be customized in the **Properties Dialog** (**Edit→Properties...**). In the Properties dialog tree on the left side of the window, select **Source**, **General**, and then check the **Show Line Number** box. Hit the **Apply** button in the right corner to validate your change.

9.5.2 Automatic Value Display

In the Text Editor, when you pass the cursor over a variable or a parameter, the value of the object automatically appears in a small yellow box. If you move the cursor away, the yellow box automatically disappears.

9.6 Breakpoints

This section describes how to set or delete breakpoints. Breakpoints are represented by a red dot on the left margin of the Text Editor. A yellow question mark '?' is displayed over the red dot if a condition is attached to the breakpoint.

9.6.1 Default Breakpoints

When running the simulator for the first time, a default breakpoint is set on the first statement of the first Verilog-A module of the circuit.

9.6.2 Toggling Breakpoints

To add or delete a breakpoint, left click in front of the line or on the button Toggling breakpoint of the Toolbar panel (see Figure 9-9). When a breakpoint is added, a red dot will appear at the line where the breakpoint is attached. If a breakpoint was already present, it will be removed.

9.6.3 The View Breakpoint Dialog

The **View Breakpoint Dialog** allows you to manage or customize breakpoints. To bring up the **View Breakpoint Dialog**, left click the button **View breakpoint...** in the toolbar panel (see Figure 9-11).

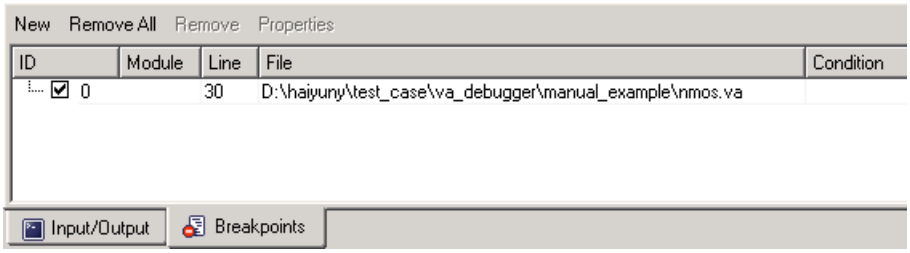


Figure 9-11 View Breakpoint Dialog Window

9.6.4 Setting Breakpoints

By clicking the **NEW** button on the breakpoint list window, a breakpoint setup dialog will pop up.

The **Breakpoint** dialog offers different ways to set breakpoints:

- **At Line Number:** Enter a valid line number (an execution statement) in the **At line** entry field. Choose a file from the next entry field (if multiple Verilog-A source files present).
- **In function:** Currently unavailable.
- **Condition:** After entering a line number (and selecting a file), you can add a condition to a breakpoint by checking the **Condition** box and entering a valid Verilog-A expression. The syntax of a valid Verilog-A expression is discussed in the next section.

Register a breakpoint by clicking the **OK** button.

9.6.5 Breakpoint Condition Expressions

A breakpoint condition is a Verilog-A expression that returns a boolean value.

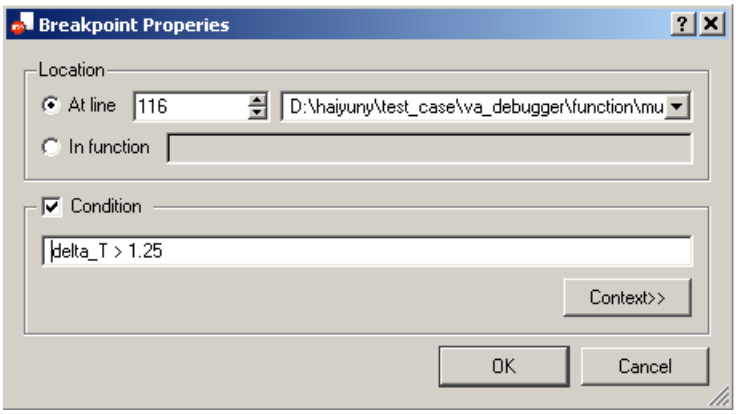


Figure 9-12 View Breakpoint Dialog With a Condition Expression

A **Context** window can be expand/hide by clicking the **Context** button in the breakpoint dialog window.

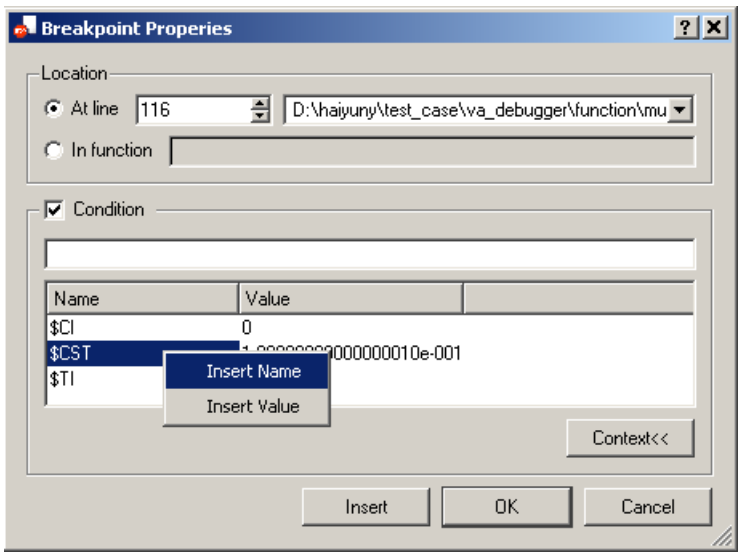


Figure 9-13 Context Window

The context window displays a list of variables and their values. The variable name or value can be inserted into the condition expression field by right-clicking on a variable and choosing **Insert Name** or **Insert Value** from the menu.

Current available variables and meanings:

`$CI`	Iterations in current simulation time
`$TI`	Total iterations
`$CST`	Current Simulation time
`$INST`	Instance name

Examples to set a conditional breakpoint:

1. Set a breakpoint so that the debugger only stops at a specific instance (e.g., `yvlgcircuit.x1.x2.nmos`):

   ```
   $INST == "yvlgcircuit.x1.x2.nmos"
   ```

2. Set a breakpoint so that the debugger stops after a set simulation point (e.g., 1.0e-6 seconds for transient or 1.0e-6V for DC sweep):

   ```
   $CST >= 1.0e-6
   ```

3. Set a breakpoint so that the debugger stops after a set number of iterations (e.g., 126 iterations):

   ```
   $TI == 126
   ```

These simple conditional expressions can be mixed to form more complex conditional functionality requirements.

The syntax for a valid condition expression must follow the BNF rules defined below:

```
expression:==
logical_operation   |
     relation_operation  |
     bitwise_operation   |
shift_operation     |
arith__operation    |
term_expression

term_expression :==
(  expression )             |
constant_number             |
variable_reference          |
parameter_reference         |
branch_quantity_reference   |
function_call

variable_reference :==
variable_identifier |
variable_identifier [ integer_expression ]

parameter_reference :==
parameter_identifier |
parameter_identifer [ integer_expression ]

branch_quantity_reference :==
access_function_identifier ( node_identifier )

constant number :==
REAL_NUMBER     |
INTEGER_NUMBER

function_call :==
builtin_function_call

analog_function_call
system_function_call
```

The logical, relational, bitwise, shift and arithmetic operators that are accepted in a condition expression are listed in the following table. The built-in functions are listed in:

Logical, Relational, Bitwise, Shift And Arithmetic Operators

Function name	Description	Accepted argument type
+, -, *, /	arithmetic	integer/real
%	modulus	integer/real
>, >=, <, <=	relational	integer/real
!	logical negation	integer/real
&&	logical and	integer/real
||	logical or	integer/real
==	logical equality	integer/real
!=	logical inequality	integer/real
~	bitwise negation	integer
&	bitwise and	integer
|	bitwise or	integer
^	bitwise exclusive or	integer
^~ or ~^	bitwise equivalence	integer
<<	left shift	integer
>>	right shift	integer

Built-in Functions

Function name	Description	Argument domain
sin(x)	Sine	All
cos(x)	Cosine	All
tan(x)	Tangent	x not odd
asin(x)	Arc-sine	[-1:1]
acos(x)	Arc-cosine	[-1,1]
atan(x)	Arc-tangent	All
sinh(x)	Hyperbolic sine	x < 80
cosh(x)	Hyperbolic tangent	x < 80
tanh(x)	Hyperbolic tangent	All
asinh(x)	Arc-hyperbolic sine	All
acosh(x)	Arc-hyperbolic cosine	x >= 1
atanh(x)	Arc-hyperbolic tangent	[-1:1]

Function name	Description	Argument domain
ln(x)	Natural logarithm	x > 0
log(x)	Decimal logarithm	x > 0
exp(x)	Exponential	x < 100
sqrt(x)	Square root	x >= 0
min(x,y)	Minimum	All
max(x,y)	Maximum	All
abs(x)	Absolute	All
pow(x,y)	Power: x^y	if x >= 0 all y;if x < 0, int(y);

9.7 The Call-Stack Window

The **Call Stack** window displays the location of the debugger inside the instance hierarchy. The call stack is updated each time the execution stops in a new instance or inside a user function. An example of a **Call Stack** window is shown in Figure 9-14. Like in the Text Editor, the yellow arrow indicates the location of the next statement executed by the debugger.

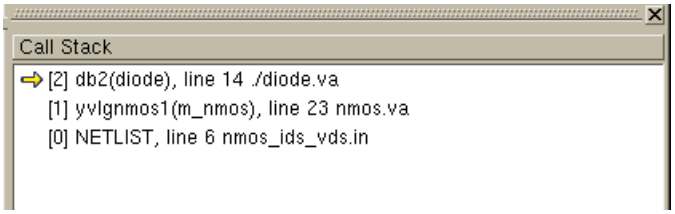


Figure 9-14 Call Stack Window

The name format for each level is the following:

```
[level] instance_name(module_name), line_number file_name
```

The level `[0]` is the top level located in the SmartSpice netlist. The line number and file name indicate where the next level instance is instantiated.

For example (see Figure 9-14), the instance `yvlgnmos1` from module `m_nmos` is instantiated in the netlist `nmos_ids_vds.in` at line 6. Double-clicking on a level will bring in the Text Editor the cursor at the instantiation location. The result of double-clicking on level `[0]` is shown in Figure 9-15.

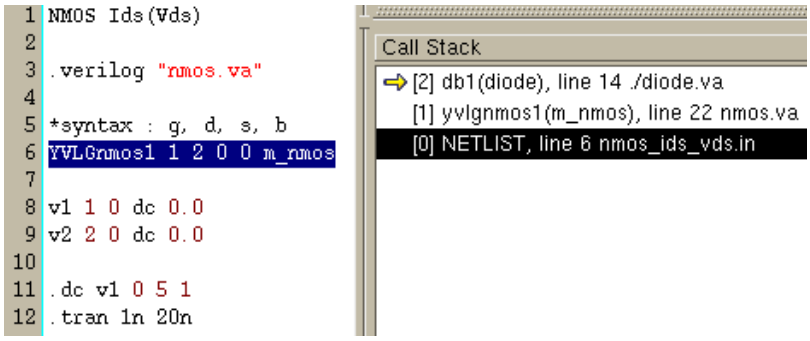


Figure 9-15 Double-clicking on Level 0 of the Call Stack

The location information at the higher level indicates the next statement executed by the debugger.

9.8 The Variables Window

9.8.1 Local Variables

The **Local Variables** panel shows the value and the type of all objects related to the Verilog-A module currently loaded in the Text Editor. Verilog-A objects are parameters, variables and branch quantities. Their values in the panel are automatically updated if needed after a stepping operation.

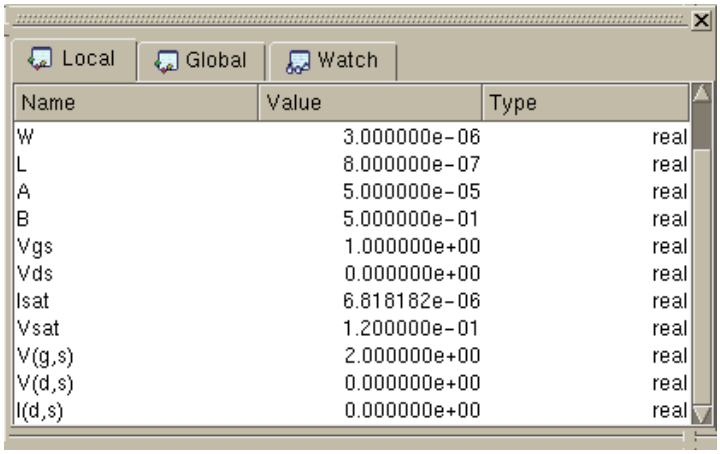


Figure 9-16 Local Variables Panel

9.8.2 Global Variables

The **Global Variables** panel shows values related to the current analysis, or values of internal parameters of the simulator.

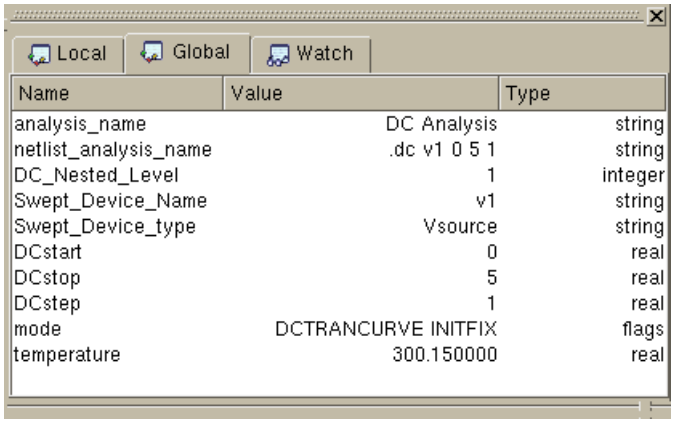


Figure 9-17 Global Variables Panel

9.8.3 Watch Variables

This feature is useful when the Verilog-A module contains a large number of parameters or variables, but only a few of them are relevant during the stepping of the code. These variables can be selected in a separate window to monitor their values more easily.

9.9 The Command Area Window

There is three panels in the command area window:

- **The Input/Output panel:** This panel acts as a console where a trace of the executed command is output. A command line prompt allows you to enter commands in a text format. The **Input/Output** panel is displayed in Figure 9-18. The list of available commands can be obtained by typing ‘help’ in the command line.

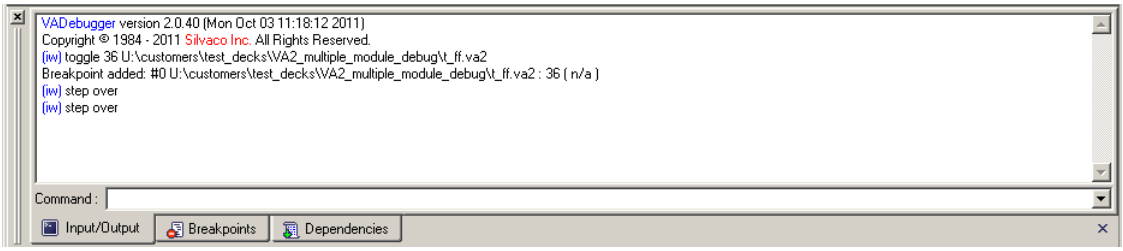


Figure 9-18 Input/Output Panel in the Console Window

- **The Dependencies panel:** This panel lists all the Verilog-A source files debuggable in this circuit. Also, double-click a listed file will display the file in text editor window, and list all user defined functions in that file.
- **The Breakpoints panel:** This panel lists all breakpoints. (Figure 9-11)

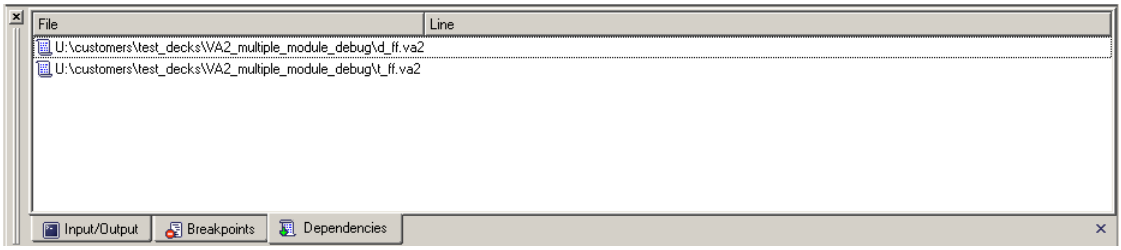


Figure 9-19 Dependencies Panel in the Console Window

9.10 The Bottom Panel: Simulation Status and Version Number

Figure 9-20 Bottom Panel Area

The bottom panel area is divided into 8 parts:

1. $CST: current simulation time.
2. $CI: iterations at current simulation time.
3. $TI: total iterations.
4. The current location of the cursor in the Text Editor.
5. File modification mode INS (insertion) or OVR (overwrite).
6. File read/write property.
7. File format (DOS or Unix).
8. The version number of the Verilog-A debugger.

9.11 Debug Multiple Verilog-A Source Files

If a circuit includes multiple Verilog-A source files, when VADebugger is started, the file contains the first module will be displayed in the text editor window. Breakpoint can be set for this file by just click in front of a line. To set breakpoint for files not displayed, VADebugger provides two ways:

1. Using new breakpoint dialog window, specify directly the file name and line number.
2. Click Dependencies panel (Figure 9-19), and double-click the file you like to set breakpoint. This file will be displayed in the text editor window, and you can set breakpoint by just clicking on the line.

Chapter 10
Verilog-A Examples

This chapter contains Verilog-A examples ranging from simple digital gates to complex behavioral models.

There is a set of Verilog-A examples shipped with the SmartSpice software under the installation area:

```
<install_directory>/examples/smartspice/<version_number>/VLGA/
VERILOGA_MAN_EX/
```

10.1 Basic Digital Electrical Models

This section describes the following basic digital components:

- Inverter
- Buffer
- Nand
- Nor
- Xor
- D-type Flip Flop
- Shift register
- Counter

10.1.1 Inverter

Figure 10-1 shows the symbol for an inverter.

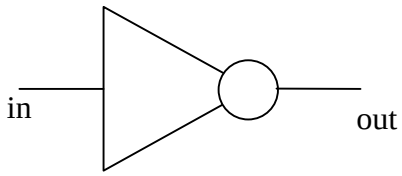


Figure 10-1 Inverter

Truth Table

IN	OUT
1	0
0	1

Verilog-A Source File (inv.va)

```
//
// Inverter
//

‘include "discipline.h"

module inv(in,out);

 input  in;
 output out;
 voltage in,out;

//Other parameters that define the performance of the inverter gate
 parameter real          vout_high = 5,
                         vout_low  = 0,
                         vth       = 1.4,
                         tdelay    = 5n from [0:inf),
                         trise     = 8.5n from [0:inf),
                         tfall     = 10n from [0:inf);
 real val;

 analog
   begin
       @(initial_step)
       begin
             if (V(in) > vth) val = vout_low;
             else             val = vout_high;
       end

       @(cross(V(in) - vth, +1)) val = vout_low;
       @(cross(V(in) - vth, -1)) val = vout_high;

       V(out) <+ transition(val, tdelay, trise, tfall);
   end
endmodule
```

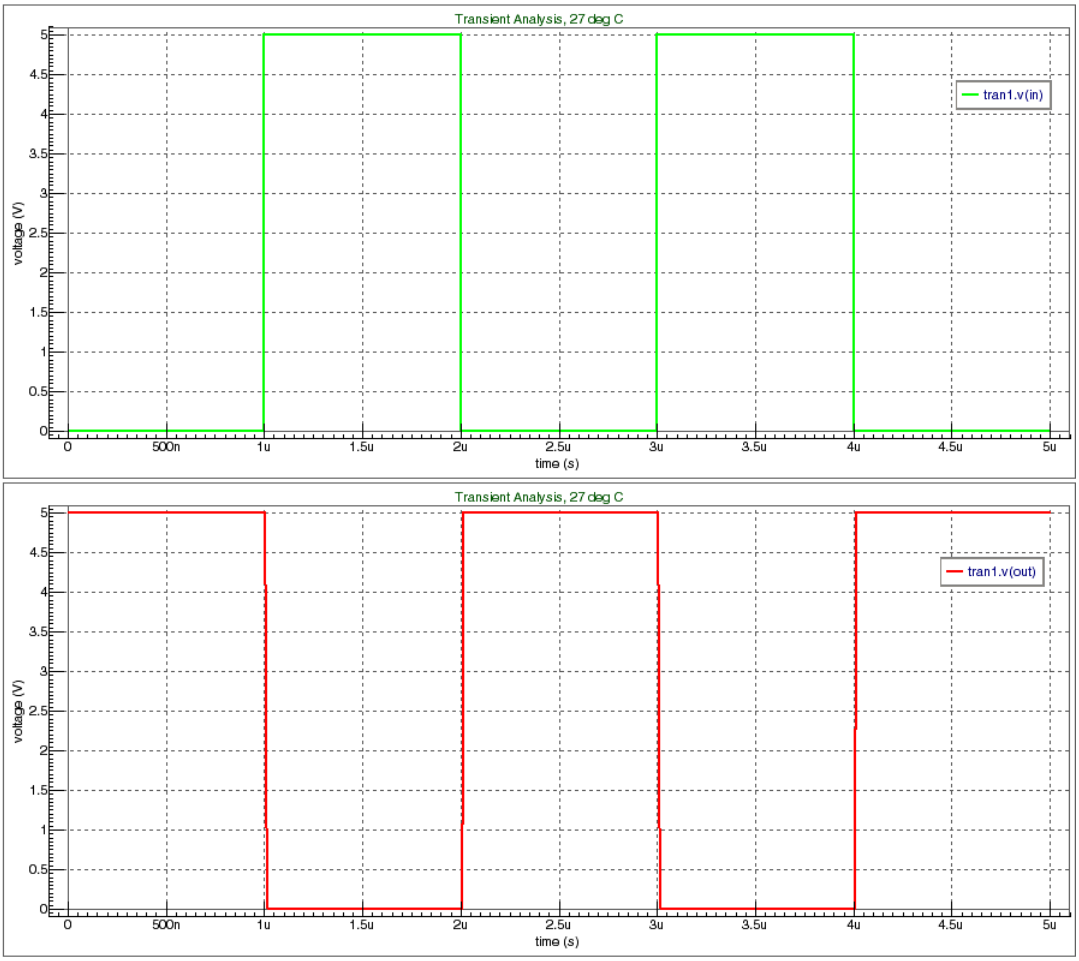


Figure 10-2 Inverter Simulation Result

Spice Input Deck (inv.in)

```
* Inverter

.verilog "inv.va"

* Input Waveform
vin in 0 pulse(0 5 1u 1n 1n 1u 2u)

* Inverter
YVLG_inv in out inv

* Output Load
RLoad out 0 1MEG

* Analysis
.tran 10n 5u

* Plot Waveforms
.iplot v(in) v(out)

.end
```

10.1.2 Buffer

Figure 10-3 shows the symbol for a buffer.

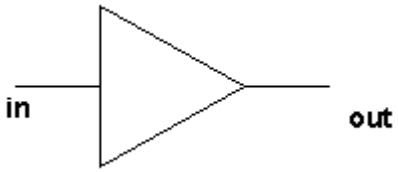


Figure 10-3 Buffer

Truth Table

IN	OUT
0	0
1	1

Verilog-A Source File (buff.va)

```
//
// Buffer
//
‘include "discipline.h"

module buffer(in,out);

 input      in;
 output     out;
 voltage    in, out;

 parameter real vout_high = 5;
 parameter real vout_low  = 0;
 parameter real vth       = 1.4;
 parameter real tdelay    = 5n;
 parameter real trise     = 8.5n from [0:inf);
 parameter real tfall     = 10n from [0:inf);

 real val;

   analog begin
     @( initial_step ) begin
        if ( V(in) > vth ) val = vout_high;
        else               val = vout_low;
     end

     @( cross( V(in) - vth, +1) ) val = vout_high;
     @( cross( V(in) - vth, -1) ) val = vout_low;

     V(out) <+ transition( val, tdelay, trise, tfall );
   end

endmodule
```

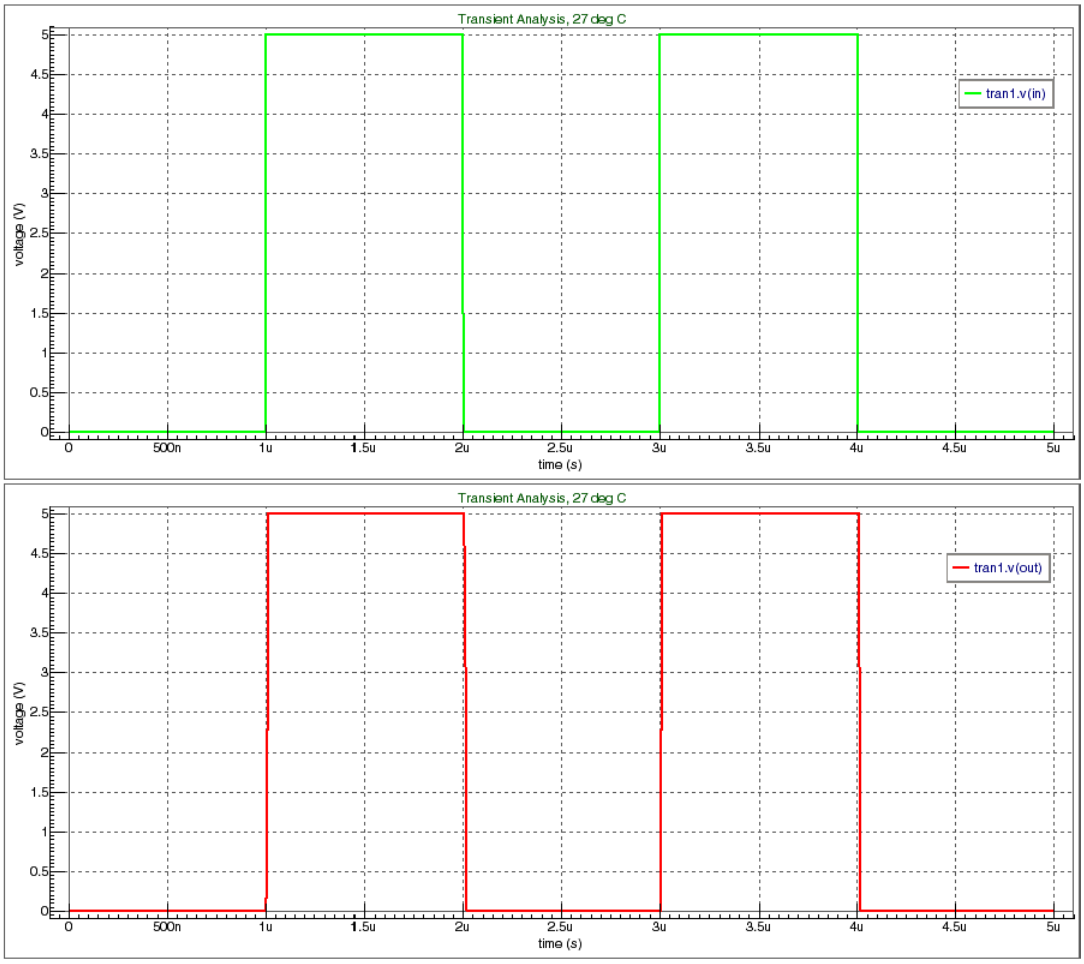


Figure 10-4 Buffer Simulation Result

Spice Input Deck (buf.in)

```
* Buffer

.verilog "buff.va"


* Input Waveform
vin in 0 pulse(0 5 1u 1n 1n 1u 2u)

* Buffer
YVLG_buf in  out buffer

* Output Loading
RLoad out 0 1MEG

* Analysis
.tran 10n 5u

* Plot Wavwforms
.iplot v(in) v(out)

.end
```

10.1.3 Nand

Figure 10-5 shows the symbol for a Nand gate.

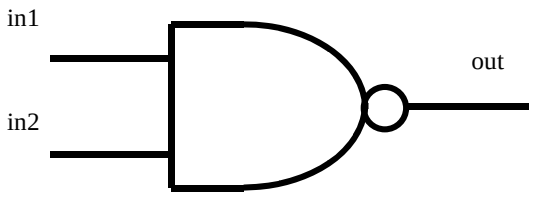


Figure 10-5 2-Input Nand Circuit

Truth Table

IN1	IN2	Out
0	0	1
0	1	1
1	0	1
1	1	0

Verilog-A Source File (nand.va)

```
//
// NAND gate
//

`include "discipline.h"

module nandg (in,out);

 parameter integer size      = 2 from [2:inf);

parameter real  vout_high = 5,
                vout_low  = 0 from (-inf:vout_high),
                vth       = 1.4,
                tdelay    = 5n from [0:inf),
                trise     = 10n from [0:inf),
                tfall     = 12n from [0:inf);

 input [0:size-1] in;
 output           out;
 voltage          in, out;

 integer in_state[0:size-1];
 integer out_state;
 genvar i, j;
 real    vout;

 analog
 begin
   @(initial_step)
     begin
      out_state   = 1;
      for(i=0; i<size; i=i+1) in_state[i] = V(in[i]) > vth;
```

```
      for (i=0; i<size; i=i+1)
         if (!(out_state && in_state[i])) out_state = 0;
       if (out_state) vout = vout_low;          // Inversion
       else           vout = vout_high;
    end


  generate i (0, size-1)
  begin
    @(cross(V(in[i]) - vth))
    begin
      in_state[i] = V(in[i]) > vth;
      out_state   = 1;
      for (j=0; j<size; j=j+1)
        if (!(out_state && in_state[j])) out_state = 0;
      if (out_state) vout = vout_low;          // Inversion
      else           vout = vout_high;
    end
  end

  V(out) <+ transition(vout,tdelay,trise,tfall);
 end
endmodule
```

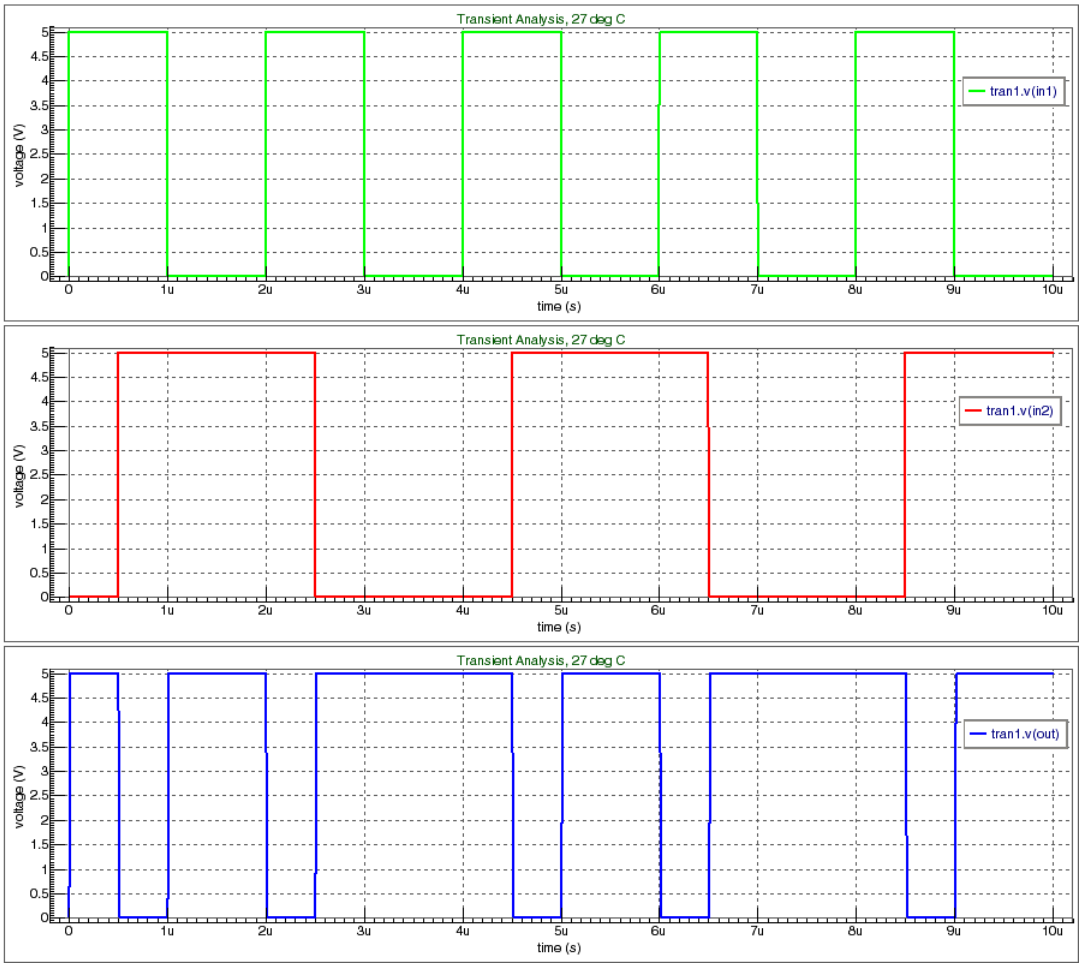


Figure 10-6 2-Input Nand Simulation Result

Spice Input Deck (nand.in)

```
* Two Input Nand Gate

.verilog "nand.va"

* Input Waveforms
vin1 in1 0 pulse(0 5 0u 1n 1n 1u 2u)
vin2 in2 0 pulse(0 5 0.5u 1n 1n 2u 4u)

* Instance Nand
YVLG_nand in2 in1 out nandg

* Output Loading
RLoad out 0 1MEG

* Analysis
.tran 10n 10u

* Plot WaveForms
.print v(in1) v(in2) v(out)

.end
```

Example of a 4 Input Nand Gate

The number of inputs is limited by the parameter `size`, so if you want more inputs, set the parameter `size = number` required.

If you want a 4 input nand gate, change the parameter `size` to 4.

The SPICE input deck is described (`4nand.in`):

```
* Four Input Nand Gate

.verilog "nand.va"

* Input Waveforms
vin1 in1 0 pulse(0 5 0u 1n 1n 1u 2u)
vin2 in2 0 pulse(0 5 0.5u 1n 1n 2u 4u)
vin3 in3 0 pulse(0 5 0u 1n 1n 4u 8u)
vin4 in4 0 pulse(0 5 0u 1n 1n 8u 16u)

* Instance Nand with size parameter
YVLG_nand in4 in3 in2 in1 out nandg size=4

* Output Loading
RLoad out 0 1MEG

* Analysis
.tran 10n 20u

* Output WaveForms
.iplot   v(in1) v(in2) v(in3) v(in4) v(out)

.end
```

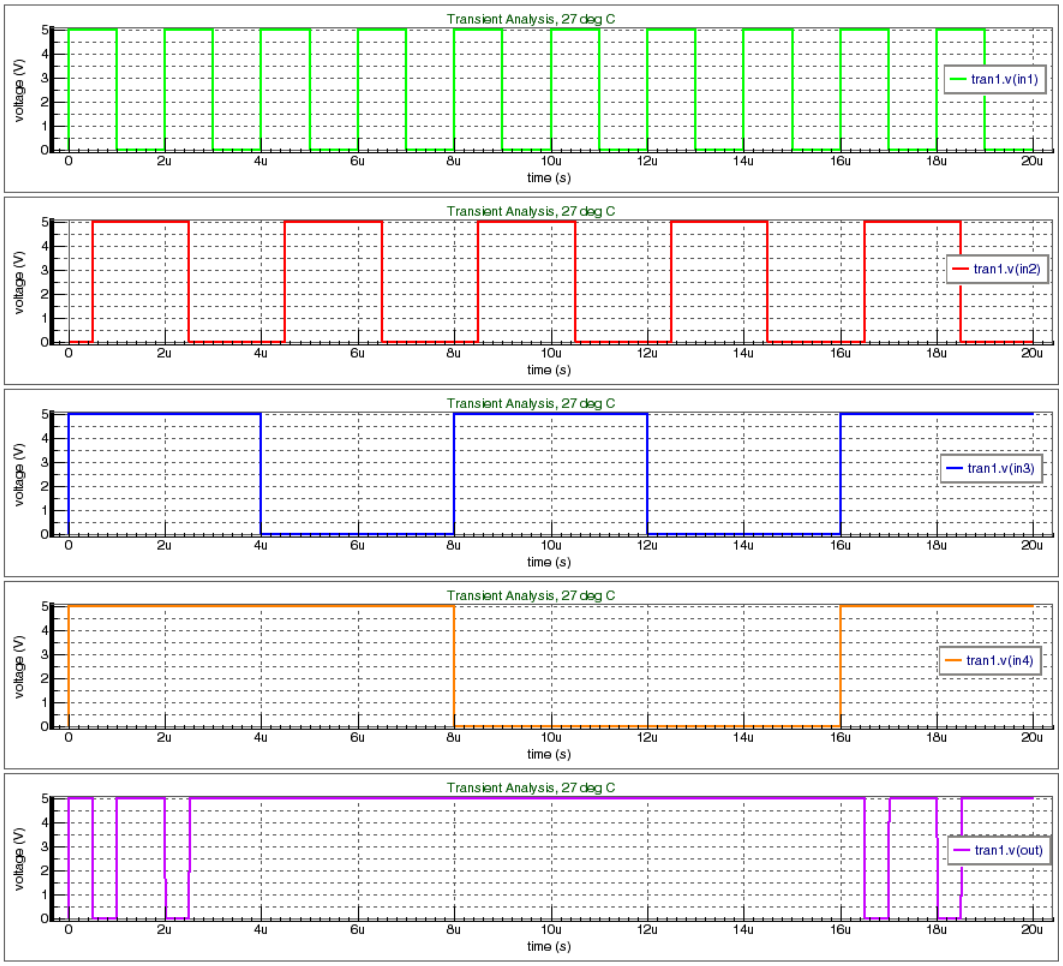


Figure 10-7 4-Input Nand Simulation Result

10.1.4 Nor

Figure 10-8 is a symbol for 2 input NOR gate.

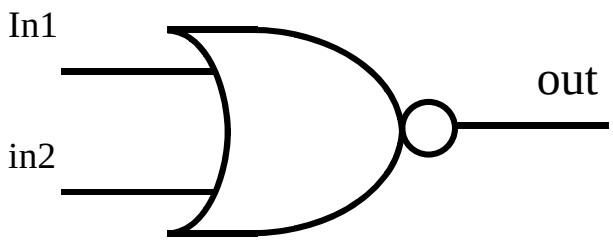


Figure 10-8 2-Input NOR

Truth Table

IN1	IN2	Out
0	0	1
0	1	0
1	0	0
1	1	0

Verilog-A Source File (nor.va)

```
//
// NOR Gate
//

`include "discipline.h"

module norg(in,out);

 parameter integer size      = 2 from [2:inf);

 parameter real     vout_high = 5,
                vout_low  = 0 from (-inf:vout_high),
                vth       = 1.4,
                tdelay    = 5n from [0:inf),
                trise     = 10n from [0:inf),
                tfall     = 10n from [0:inf);

 input [0:size-1] in;
 output           out;
 voltage in, out;

 integer in_state [0:size-1];
 genvar i, j;
 integer out_state;
 real    vout;

 analog
 begin
   @(initial_step)
     begin
```

```
      out_state   = 0;
      for(i=0; i<size; i=i+1) in_state[i] = V(in[i]) > vth;
      for (i=0; i<size; i=i+1)
         if (in_state[i]) out_state = 1;
       if (out_state) vout = vout_low;          // Inversion
       else           vout = vout_high;
    end

  generate i (0,size-1)
  begin
    @(cross(V(in[i]) - vth))
    begin
      in_state[i] = V(in[i]) > vth;
      out_state   = 0;
      for (j=0; j<size; j=j+1)
        if (in_state[j]) out_state = 1;
      if (out_state) vout = vout_low;    // Inversion
      else           vout = vout_high;
    end
  end

  V(out) <+ transition(vout, tdelay, trise, tfall);
end

endmodule
```

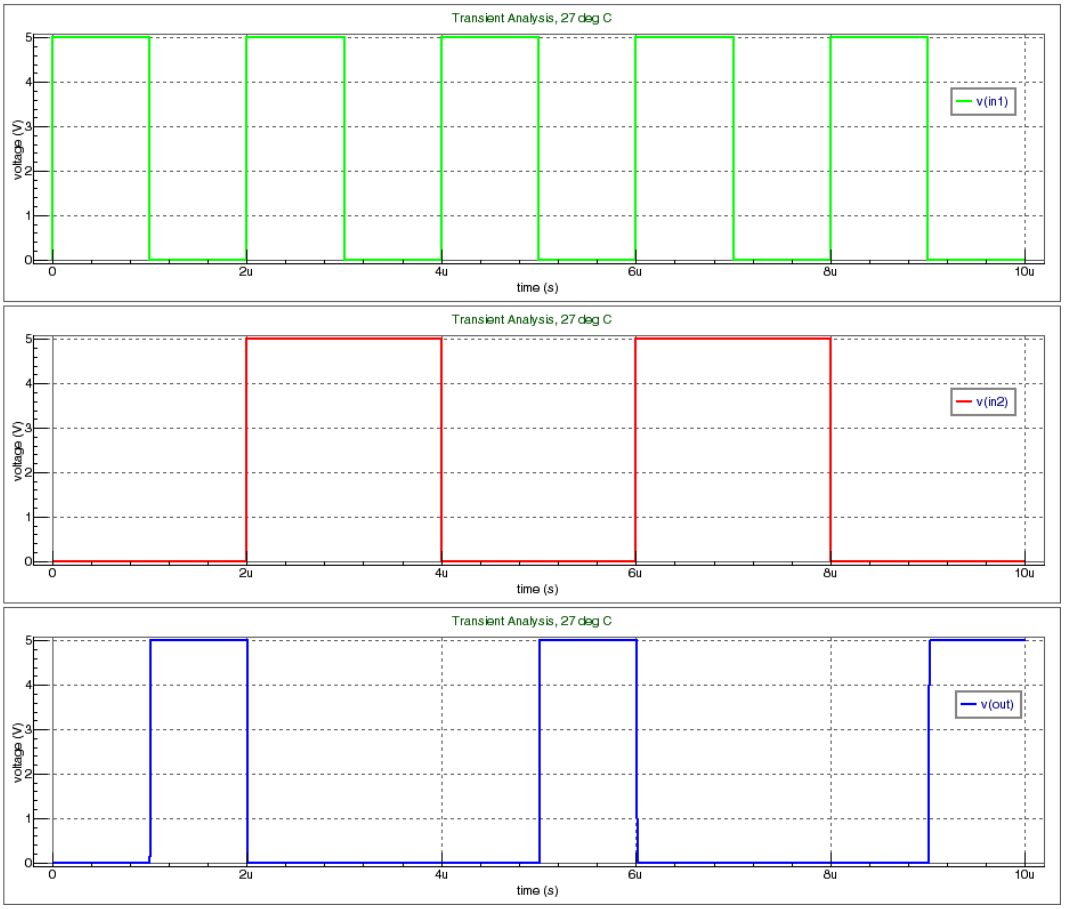


Figure 10-9 2-Input NOR Simulation Result

Spice Input Deck (nor.in)

```
* Two Input NOR Gate

.verilog "nor.va"

* Input Waveforms
vin1 in1 0 pulse(5 0 0u 1n 1n 1u 2u)
vin2 in2 0 pulse(5 0 2u 1n 1n 2u 4u)

* Instance Nor
YVLG_NOR in1 in2 out norg

* Output LoadLoading
RLoad out 0 1MEG

* Analysis
.tran 10n 10u

* Plot WaveForms
.print v(in1) v(in2) v(out)

.end
```

Example of a 3 Input Nor Gate

This example can also increase the number of inputs. If you want more (e.g., 3 inputs), change the SPICE input deck as follows.

Spice Input Deck (3nor.in):

```
* Three Input NOR Gate

.verilog "dnor.va"

* Input Waveforms
vin1 in1 0 pulse(0 5 0u 1n 1n 1u 2u)
vin2 in2 0 pulse(0 5 1u 1n 1n 2u 4u)
vin3 in3 0 pulse(0 5 1u 1n 1n 4u 8u)

* Instance NOR Gate with size parameter
YVLG_NOR in1 in2 in3 out norg size=3

* Output Loading
RLoad out 0 1MEG

* Analysis
.tran 10n 20u

* Output WaveForms
.iplot   v(in1) v(in2) v(in3) v(out)

.end
```

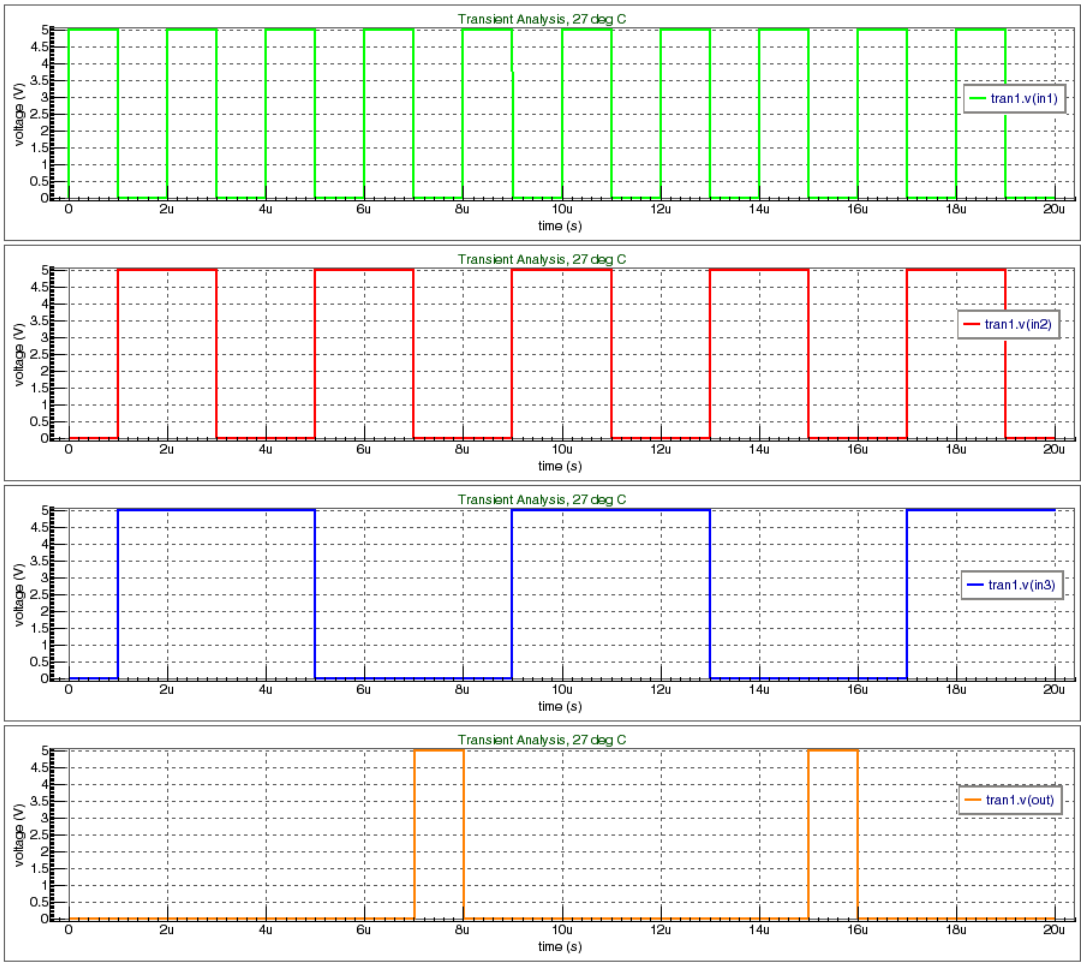


Figure 10-10 3-Input NOR Simulation Result

10.1.5 Xor

Figure 10-11 is a symbol for a 2 input XOR gate.

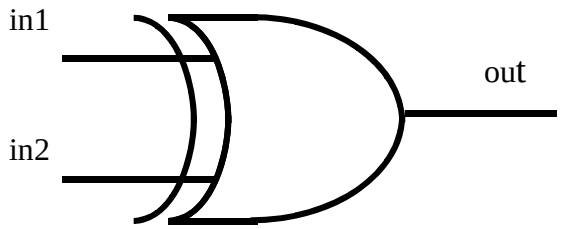


Figure 10-11 2-Input XOR

Truth Table

IN1	IN2	Out
0	0	0
0	1	1
1	0	1
1	1	0

Verilog-A Source File (xor.va)

```
//
// XOR Gate
//

`include "discipline.h"

module xorg(in,out);

 parameter integer size    = 2 from [2:inf);
 parameter real    vhigh   = 5,
                   vlow    = 0 from (-inf:vhigh),
                   vth     = 1.4,
                   tdelay  = 5n from [0:inf),
                   trise   = 10n from [0:inf),
                   tfall   = 10n from [0:inf);

 input [0:size-1] in;
 output      out;
 voltage in, out;

 integer in_state [0:size-1];
 integer out_state;
 genvar i, j;
 real    vout;

 analog
 begin
   @(initial_step)
     begin
      out_state   = 0;
      for(i=0; i<size; i=i+1) in_state[i] = V(in[i]) > vth;
```

```
      for (i=0; i<size; i=i+1)
         if (in_state[i]) out_state = out_state + 1;
       if (out_state == 1) vout = vhigh;
       else                vout = vlow;
    end

  generate i (0, size-1)
  begin
    @(cross(V(in[i]) - vth))
    begin
      in_state[i] = V(in[i]) > vth;
      out_state   = 0;
      for (j=0; j<size; j=j+1)
       if (in_state[j]) out_state = out_state + 1;

      if (out_state == 1) vout = vhigh;
      else                vout = vlow;
    end
  end

  V(out) <+ transition(vout, tdelay, trise, tfall);
end

endmodule
```

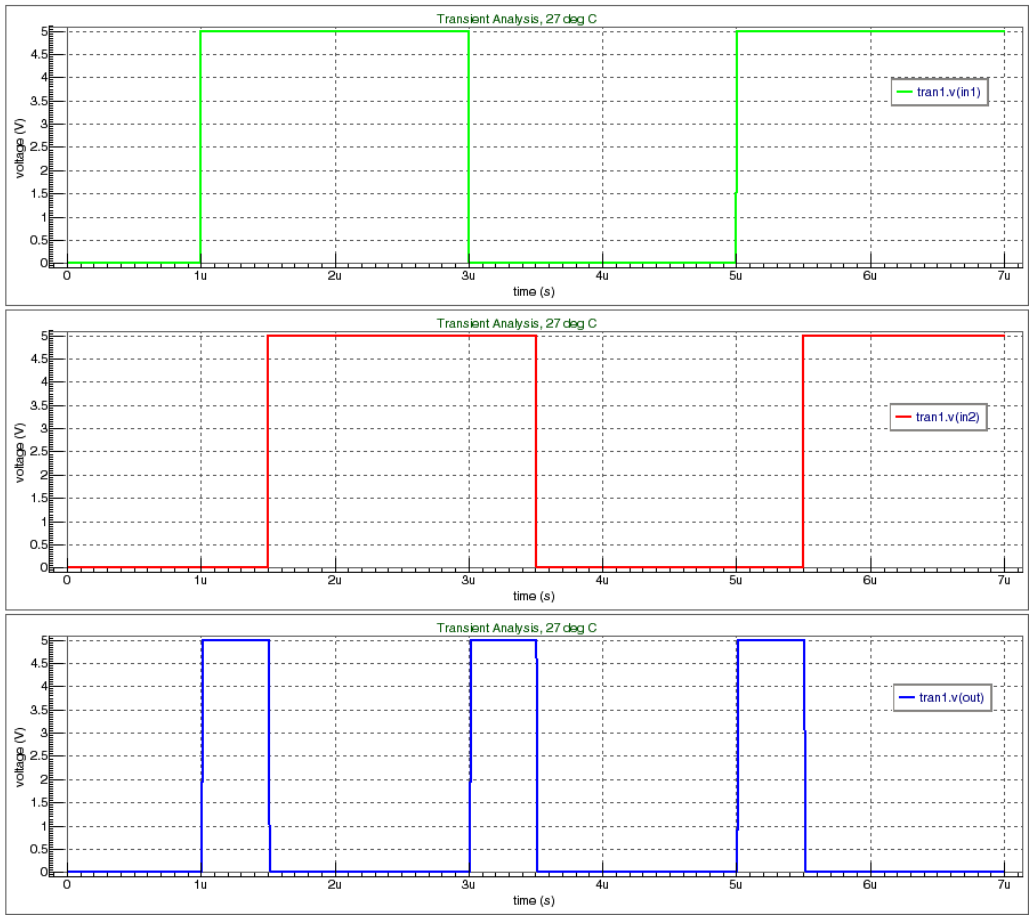


Figure 10-12 XOR Simulation Result

Spice Input Deck (xor.in)

```
* Two Input XOR Gate

.verilog "xor.va"

* Input Waveforms
vin1 in1 0 pulse(0 5 1u 1n 1n 2u 4u)
vin2 in2 0 pulse(0 5 1.5u 1n 1n 2u 4u)

* Instance Xor
YVLG_XOR in1 in2 out xorg

* Output Loading
RLoad out 0 1MEG

* Analysis
.tran 10n 7u

* Plot Waveforms
.print v(in1) v(in2) v(out)

.end
```

10.1.6 DFF (D-type Flip Flop)

Figure 10-13 is a symbol for D type Flip Flop.

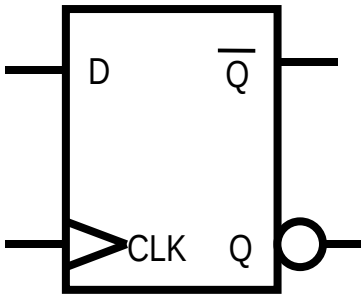


Figure 10-13 D-Type Flip Flop

Truth Table

CLK	D	Q	Q_bar		
0	0	0	1		
0	1	0	1		
1	1	1		0	(CLK:L - >H)
1	0	0		1	(CLK:L - >H)

Verilog-A Source File (dff.va)

```
//
// DFF - D-Type Flip Flop
//

‘include "discipline.h"

module dff(clk, d, q, qbar);

 input   clk, d;
 output  q, qbar;
 voltage q, qbar, clk, d;

 parameter real tdelay    = 5n from [0:inf),
                ttransit  = 5n from [0:inf),
                vout_high = 5,
                vout_low  = 0 from (-inf:vout_high),
                vth       = 1.4; // Threshold Voltage

 integer state; // Flip Flop State

 analog
 begin
   @(initial_step) state = 0; // Start off reset to zero

   @(cross(V(clk) - vth, +1 )) state = (V(d) > vth); // Evaluate at
clk rising

    V(q)     <+ transition( vout_high*state   + vout_low*!state,
tdelay, ttransit );
```

```
      V(qbar)  <+  transition(  vout_high*!state  +  vout_low*state,
tdelay, ttransit );
 end

endmodule
```

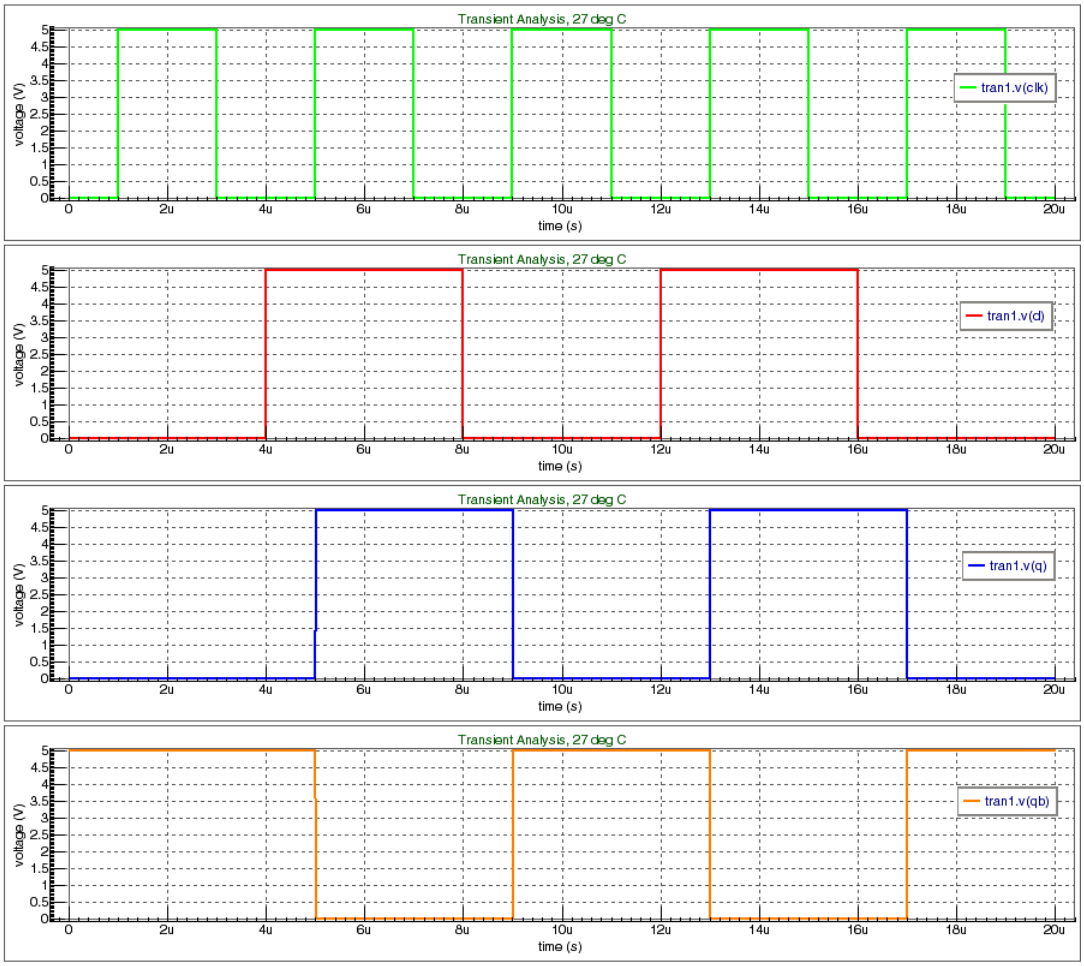


Figure 10-14 DFF Simulation Result

Spice Input Deck (dff.in)

```
* D-Flip Flop

.verilog "dff.va"

* Data
vd   d   0 pulse ( 0 5 4u 1n 1n 4u 8u)

* Clock
vclk clk 0 pulse (0 5 1u 1n 1n 2u 4u)

* Instance D-FlopFlop
YVLGa_nd clk d q qb dff

* Output Loading
CLoadQ  q  0 1n
CLoadQB qb 0 1n

* Analysis
.tran 100n 20u

* Plot Waveforms
.iplot v(clk) v(d) v(q) v(qb)

.end
```

10.1.7 4 Bit Shift Register

Figure 10-15 shows a 4 bit `shift_register` example that uses a DFF block.

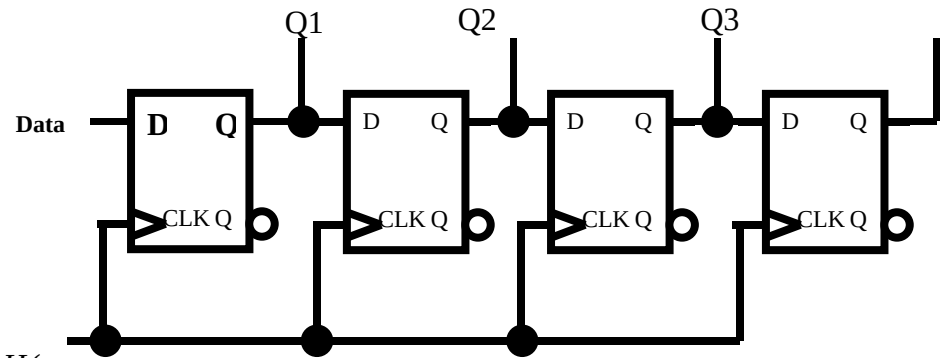


Figure 10-15 4bitShift Register

Spice Input Deck (4bit_shift_reg.in)

```
* 4 Bit Shift Register

.verilog "dff.va"

* Input Waveforms
*    Data
vdata din  0 pwl(0 0 0.999u 0 1u 5 1.999u 5 2u 0 20u 0 R 0n)
*    Clock
vclk  clk  0 pulse(0 5 1u 1n 1n 2u 4u)

* Instance Shift Register
YVLG_dff1 clk din q1 q1b dff
YVLG_dff2 clk q1  q2 q2b dff
YVLG_dff3 clk q2  q3 q3b dff
YVLG_dff4 clk q3  q4 q4b dff

* Output Loading
RLoadQ1b q1b 0 1MEG
RLoadQ2b q2b 0 1MEG
RLoadQ3b q3b 0 1MEG
RLoadQ4  q4  0 1MEG
RLoadQ4B q4b 0 1MEG

* Analysis
.tran 200n 20u

* Plot Waveforms
.iplot v(clk) v(din) v(q1) v(q2) v(q3) v(q4)

.end
```

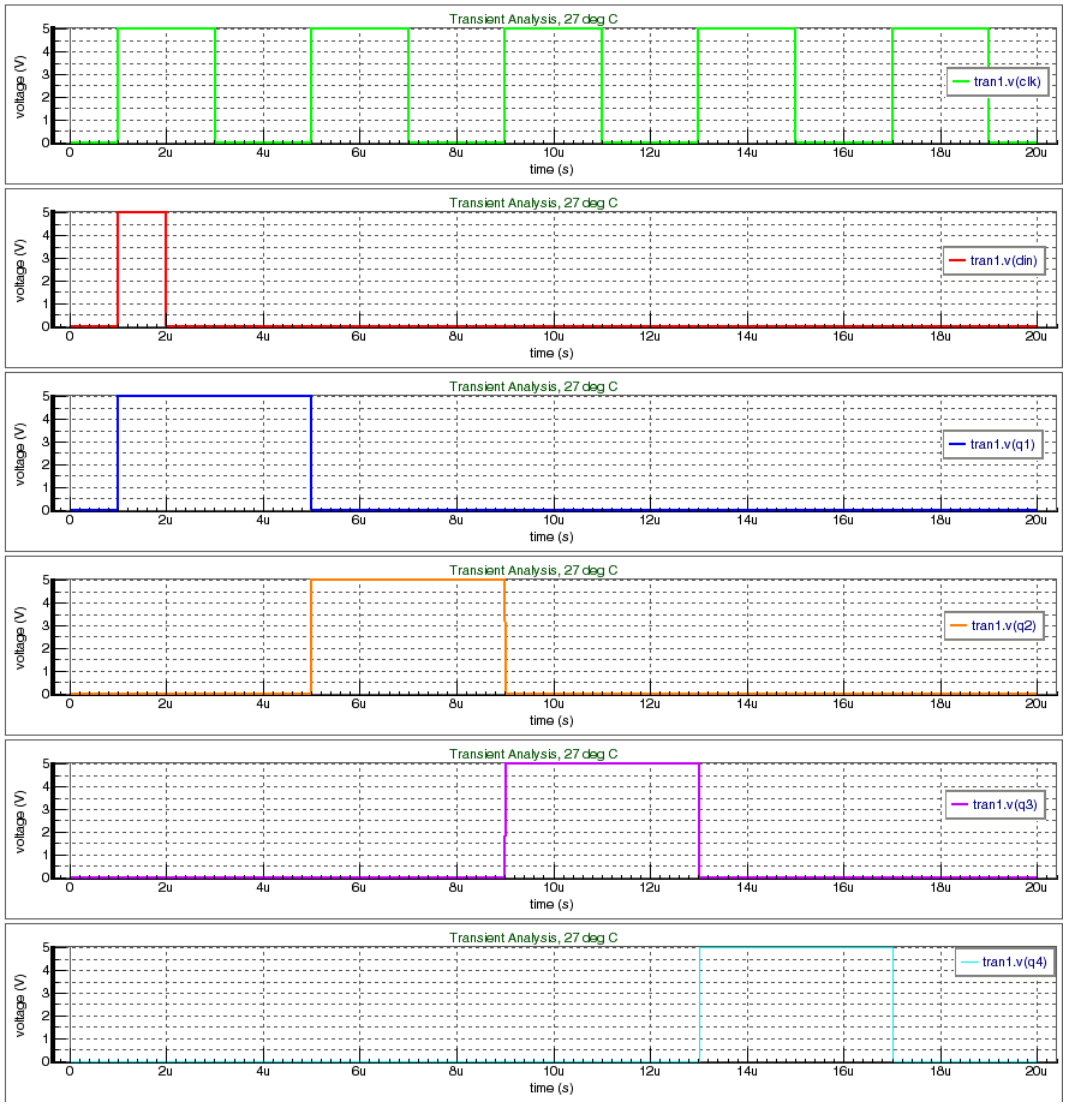


Figure 10-16 4 Bit Shift Register Simulation Result

10.2 Basic Analog Electrical Models

This section describes the following basic analog components:

- Low pass filter
- High pass filter
- Band pass filter
- OPAMP (Operational Amplifier)
- Sample and Hold model
- Ideal Analog to Digital Converter (ADC)
- Ideal Digital to Analog Converter (DAC)
- Delta-Sigma Modulator

10.2.1 Low Pass Filter (LPF)

Figure 10-17 is a symbol for a low pass filter.

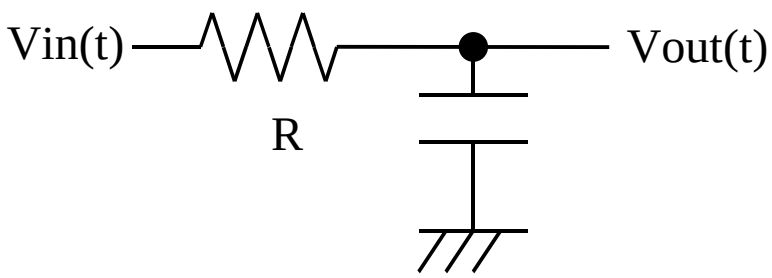


Figure 10-17 RC Filter

The transfer function for the first order LPF is given by the following equation:

$$H(s) = \frac{1}{1 + \left(\frac{1}{2\pi f_0}\right)S},$$

where f_0 is cutoff frequency.

This equation can be written in Verilog-A using the Laplace function.

```
laplace_nd(V(in), {1}, {1, 1/(`M_TWO_PI*freq_p1)}),
```

where `freq_p1` - cutoff frequency.

Figure 10-18 shows the LPF frequency response.

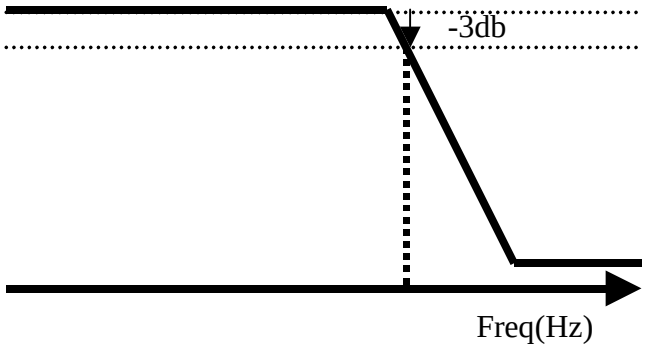


Figure 10-18 LPF Characteristics

Verilog-A Source File (lpf.va)

```
//
// LPF - Low Pass Filter
//

‘include "discipline.h"
‘include "constants.h"

module lpf(in,out);

 parameter real freq_p1 = 1M from (0:inf);

 input  in;
 output out;

 voltage in, out;

 analog
   V(out) <+ laplace_nd(V(in), '{1}, '{1, 1/(‘M_TWO_PI*freq_p1)});

endmodule
```

Spice Input Deck (lpf.in)

```
* Low pass filter

.verilog "lpf.va"

* Input Waveform
vin in 0 ac 1 sin(0,1, 250)

* Instance LPF
YVLG_lpf in out lpf

* Output Loading
Rload out 0 1MEG

* Analyses
.ac oct 10 1 1g
.tran 0.1m 4m

* Plot Waveforms
.iplot ac vdb(out)
.iplot tran v(in)

.end
```

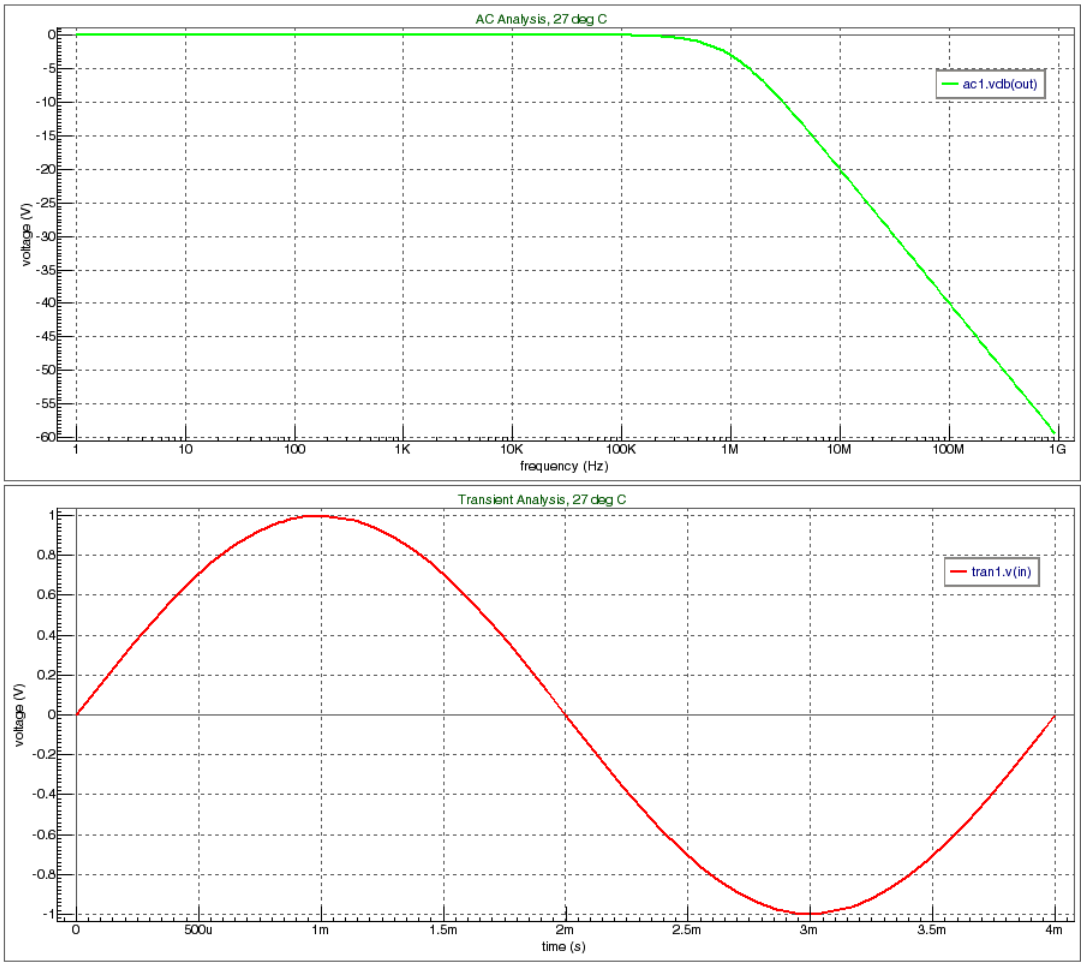


Figure 10-19 LPF Simulation Result

10.2.2 High Pass Filter (HPF)

Figure 10-20 is the symbol for a high pass filter.

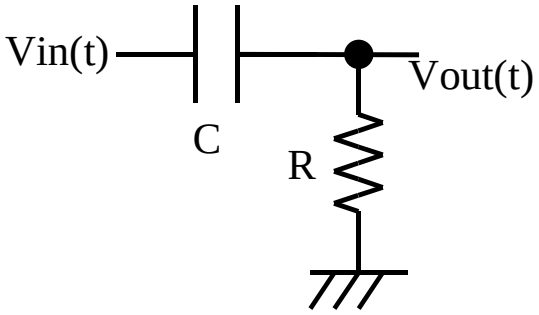


Figure 10-20 HPF filter

The transfer function for the first order HPF is described by the following equation.

$$H(s) = \frac{\left(\frac{1}{2\pi f_0}\right)S}{1 + \left(\frac{1}{2\pi f_0}\right)S}$$

We can write the equation to Verilog-A using the Laplace function:

```
laplace_nd(V(in),      {0,1/(`M_TWO_PI*freq_p1)}      ,      {1,1/
(`M_TWO_PI*freq_p1)} )
```

Verilog-A Source (hpf.va)

```
//
// HPF - High Pass Filter
//

‘include "discipline.h"
‘include "constants.h"

module hpf(in,out);

 input   in;
 output  out;

 voltage in, out;

 parameter real freq_p1 = 1M from (0:inf); // HPF Cutoff Frequency

 analog
   V(out) <+ laplace_nd(V(in), {0,1/(‘M_TWO_PI*freq_p1)} ,
              {1,1/(‘M_TWO_PI*freq_p1)} );

endmodule
```

Spice Input Deck (hpf.in)

```
* High Pass Filter

.verilog "hpf.va"

* Input Waveform
vin in 0 1 ac 1 sin(0,1, 250)

* * Instance HPF
YVLG_hpf in out hpf

* Output Loading
Rload out 0 1MEG

* Analyses
.ac oct 10 1 1g
.tran 0.1m 4m

* Plot Waveforms
.iplot ac vdb(out)
.iplot tran v(in)

.end
```

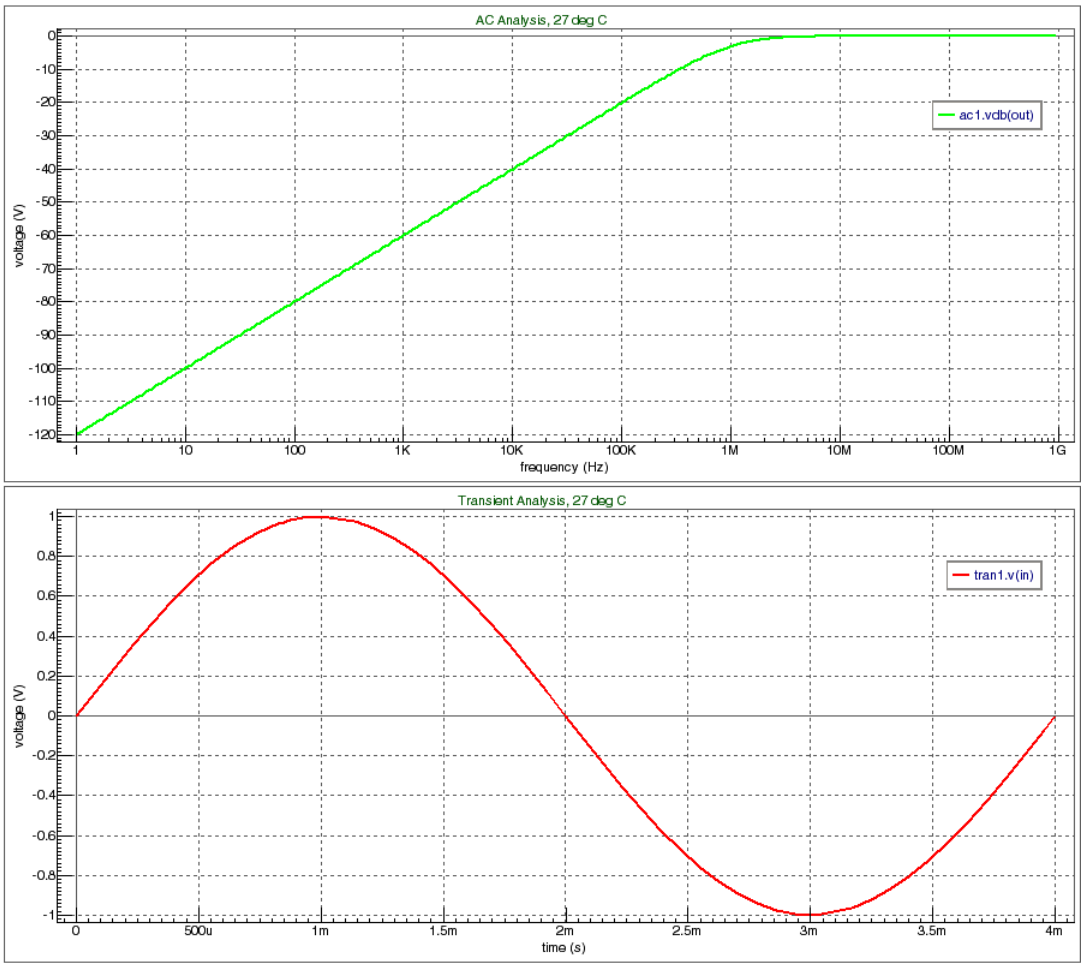


Figure 10-21 HPF Simulation Result

10.2.3 Band Pass Filter (BPF)

Figure 10-22 shows the frequency response for a band pass filter.

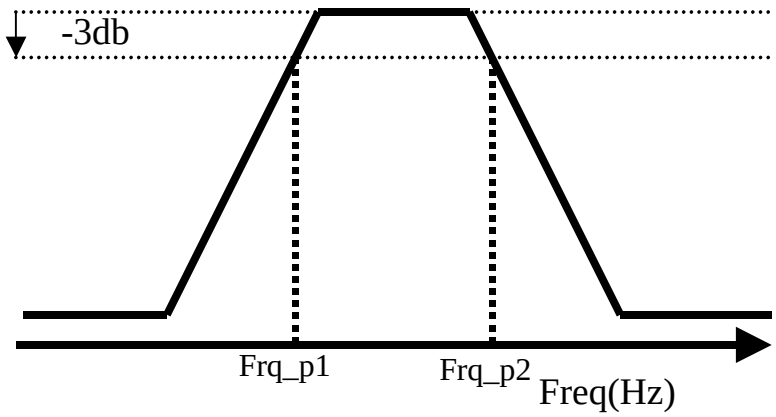


Figure 10-22 BPF Characteristics

If the input signal Frequency is "f", then BPF behavior is shown as:

- If f <Frq_p1: The input signal is attenuated.
- If Frq_p1 < f <Frq_p2: The input signal is not attenuated.
- If Frq_p2 < f: The input signal is attenuated.

The BPF transfer function is given, for example, by the following equation:

$$s) = \frac{\left(\frac{\quad}{2\pi f_0^2}\right)S}{1 + \left(\frac{\Delta f}{2\pi f^2}\right)S + \left(\frac{1}{(2\pi f \quad)^2}\right)},$$

where Δf is bandwidth, f_0 is peak frequency.

We can describe transfer function for bandpass filter using the next Verilog-A laplace function:

```
laplace_nd(V(in), {0,fdelta/(`M_TWO_PI * fc * fc)},
{1, fdelta/(`M_TWO_PI * fc * fc),
1/pow((`M_TWO_PI * fc),2)}).
```

Verilog-A Source (bpf.va)

```
//
// Bandpass filter 20-120kHz
//

‘include "discipline.h"
‘include "constants.h"

module bpf(in,out);

 input in;
 output out;
 voltage in,out;

 parameter real     fc = 50k from (0:inf),
   fdelta = 100k from [0:inf);


 analog

   V(out) <+ laplace_nd( V(in), {0, fdelta/( ‘M_TWO_PI * fc * fc )},
        {1, fdelta/( ‘M_TWO_PI * fc * fc ), 1/( pow( (‘M_TWO_PI *
fc), 2) )} );

endmodule
```

Spice Input Deck (bpf.in)

```
* Bandpass filter 20-120kHz

.verilog "bpf.va"

* Input Waveform
vin in 0 ac 1 sin(0,1, 250)

* Instance BPF
YVLG_lpf in out bpf

* Output Loading
Rload out 0 1MEG

* Analyses
.ac DEC 100 10k 240k
.tran 0.1m 4m

* Plot Waveforms
.iplot ac vdb(out)
.iplot tran v(in)

.end
```

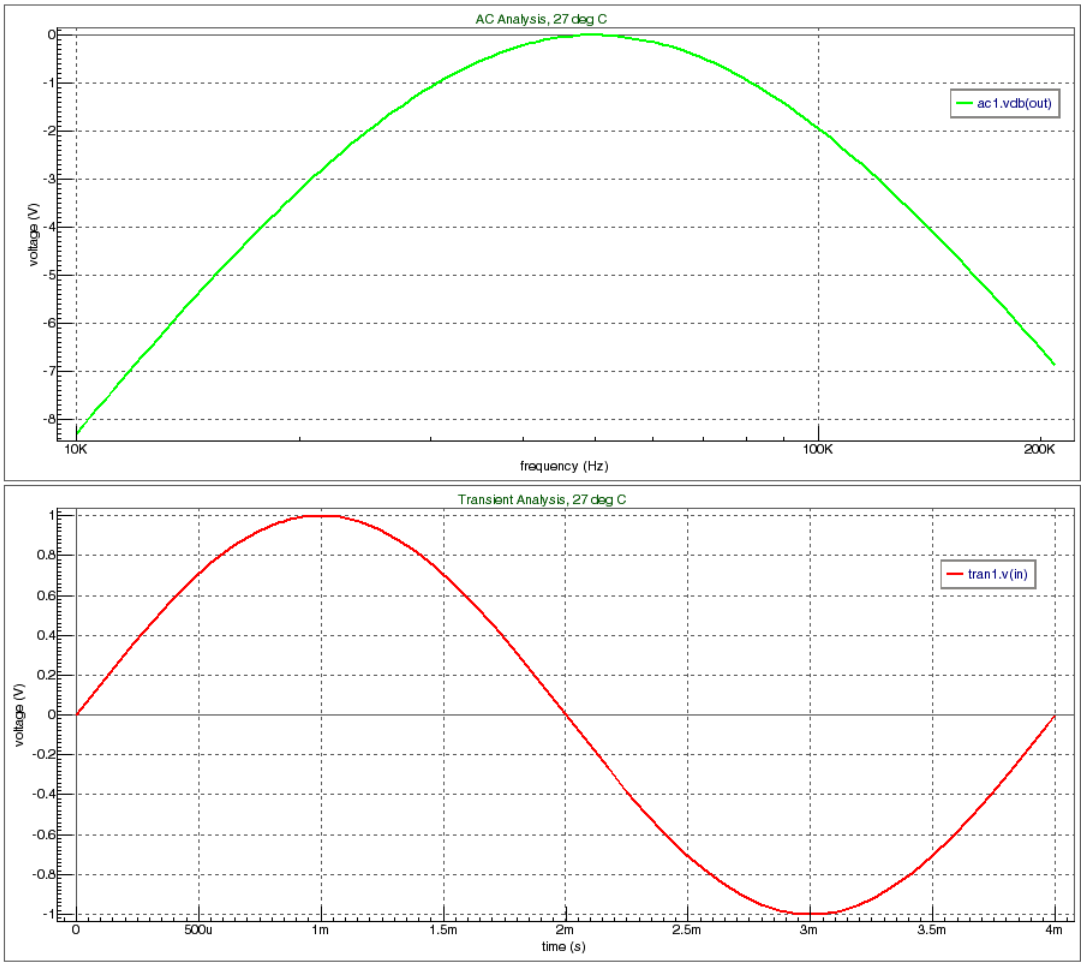


Figure 10-23 BPF Characteristics

10.2.4 OPAMP (Operational Amplifier)

Figure 10-24 is an example of an inverting operational amplifier (opamp). The input is Vin, the output is Vout, and the negative feedback resistances are R1 and R2.

The opamp gain is given by the following equation:

$$\text{Gain} = \frac{R2}{R1}$$

The opamp behavior is described by the following equation:

$$\text{Vout} - \text{Vref} = -\frac{R2}{R1} \times (\text{Vin} - \text{Vref})$$

However, note that Vout is also clamped by the supply voltage.

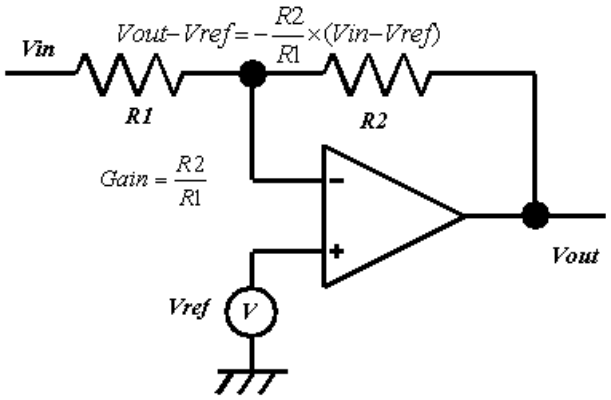


Figure 10-24 Inverted Amplifier

Verilog-A Source (amp.va)

```
//
// Models an inverting amplifier built from an opamp.
// The gain is the ratio of R2 and R1.
//

‘include "discipline.h"

module amp(in,out);

 input      in;
 output     out;
 voltage    in, out;

 parameter real  avcc = 5;    //analog supply voltage source
 parameter real  avss = 0;    //analog ground reference
 parameter real  vref = 2.5;  //reference input
 parameter real  R1   = 1k;   //inital resitance R1
 parameter real  R2   = 10k;  //inital resitance R2

 real gain, vout;

  analog
  begin
```

```
        @(initial_step)
    begin
      gain = -R2/R1;  // Inverting Amplifier
    end

        vout   = gain*(V(in) - vref) + vref; // Amplify the input
signal

       if( vout > avcc ) // Clamp output signal to PSU plus
            vout = avcc;
       if( vout < avss ) // Clamp output signal to PSU minus
            vout = avss;

       V(out) <+ vout;
  end

endmodule
```

Spice Input Deck (amp.in)

```
* Opamp (Operational amplifier)

.verilog "amp.va"

* Input Waveform
vin      in  0   sin(2.5 0.1 1meg)

* Opamp
YVLG_AMP in  out amp

* Output Loading
RLoad out 0 1MEG

* Analysis
.tran 1n 2.5u

* Plot Waveforms
.iplot v(in) v(out)

.end
```

In the above SPICE input deck, the amplitude of the input signal is `0.1V` and `R2` is set to `10k`, so the `Gain=R2/R1=10k/1k=10`. We expect the amplitude of `Vout` to be 10 times of the input signal voltage.

Figure 10-25 shows the result of the SmartSpice simulation.

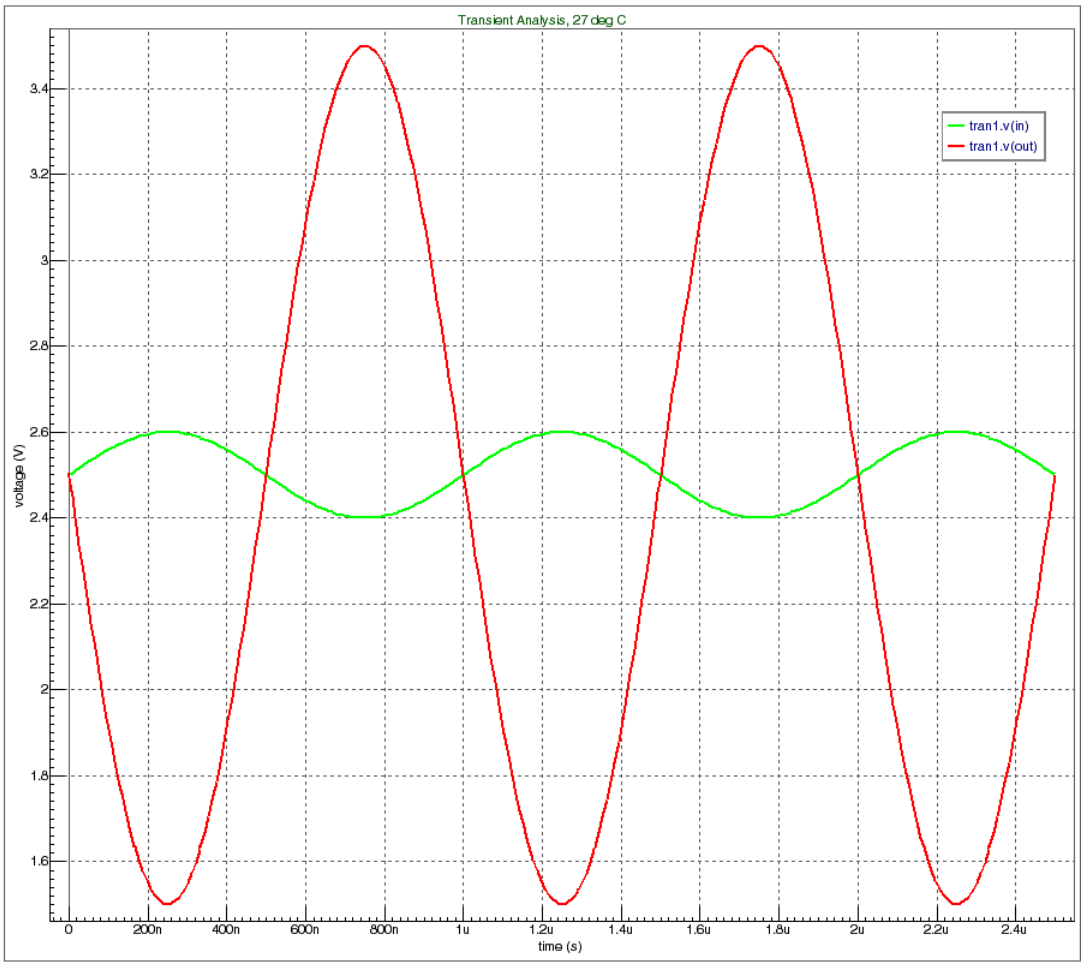


Figure 10-25 OPAMP Simulation Result

10.2.5 Sample and Hold model

A Sample and Hold circuit is shown in Figure 10-26.

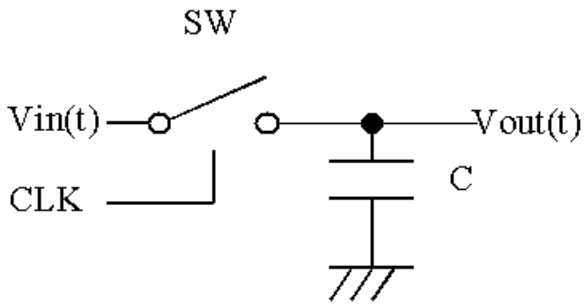


Figure 10-26 Sample and Hold Circuit

The Sample and Hold circuit performs the following function:

CLK: sampling clock signal

```
CLK=High, SW=CLOSED
CLK=LOW,  SW=OPEN
```

In other words, when CLK is high at t, Vout(t) is Vin(t). When CLK changes from high to low at t+dt, Vout(t+dt) is Vout(t).

Verilog-A Source (sample_hold.va)

```
//
// Sample and hold model
//

‘include "discipline.h"

module sample_hold(in,out,clk);

 input in,clk;
 output out;
 voltage in,out,clk;

 parameter real clk_vth  = 2.5;
 real v;

 analog
 begin
   @(initial_step)
     v = V(in);

   if (analysis("static") || (V(clk) > clk_vth))
     v = V(in);                          // passing phase

   @(cross(V(clk)-clk_vth,0))
     v = V(in);                          // sampling phase

   V(out)<+v;

 end
```

```
endmodule
```

Spice Input Deck (shold.in)

```
* Sample and Hold

.verilog "sample_hold.va"

* Input Waveform
v01 in 0  sin(0 2.5 100k)
v02 clk 0 pulse(0 5 0.2u 1n 1n 0.3u 0.6u)

* Instance Sample and Hold
YVLG_sample_amp in out clk sample_hold

* Output Loading
RLoad out 0 1MEG

* Analysis
.tran 0.1u 10u

* Plot Waveforms
.iplot v(in) v(clk) v(out)

.end
```

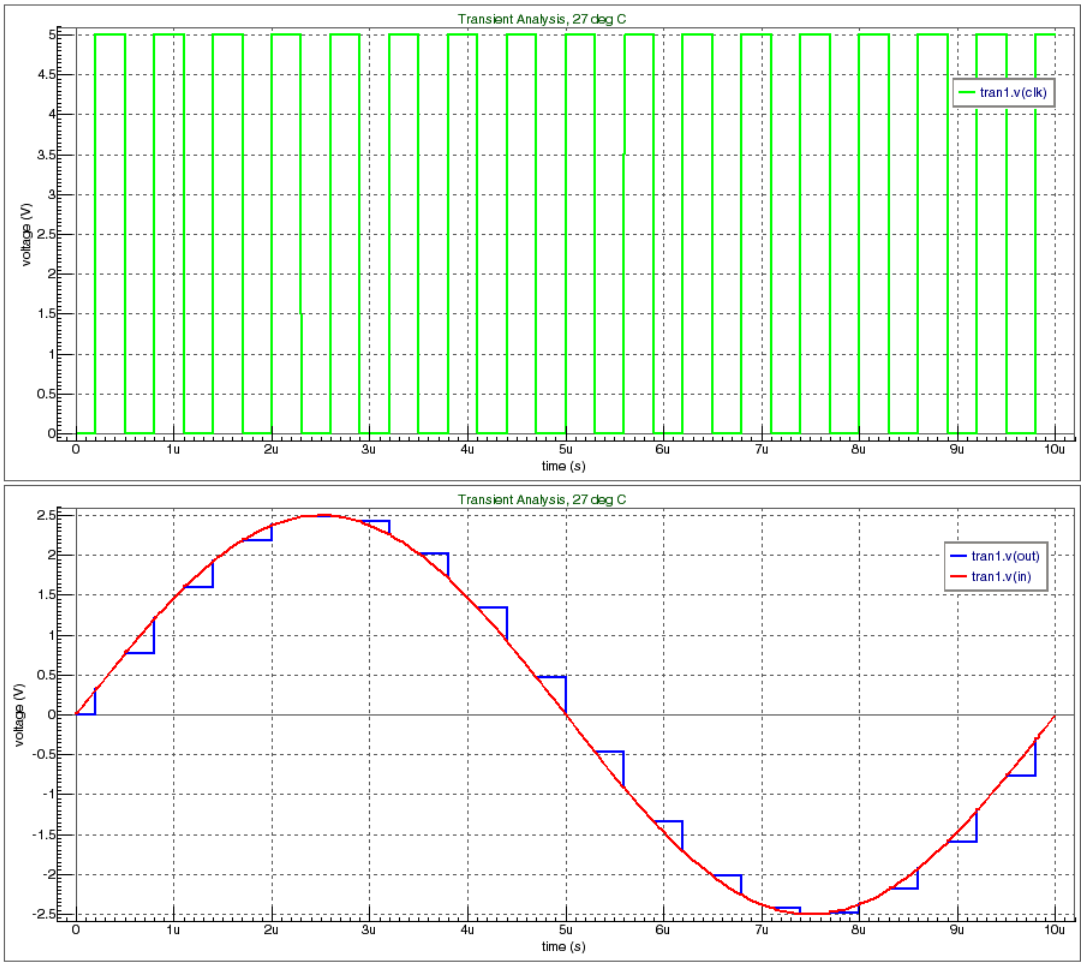


Figure 10-27 Sample and Hold Simulation Result

10.2.6 Ideal Analog to Digital Converter (ADC)

An Analog to Digital Converter changes an analog signal to a digital code with a particular resolution.

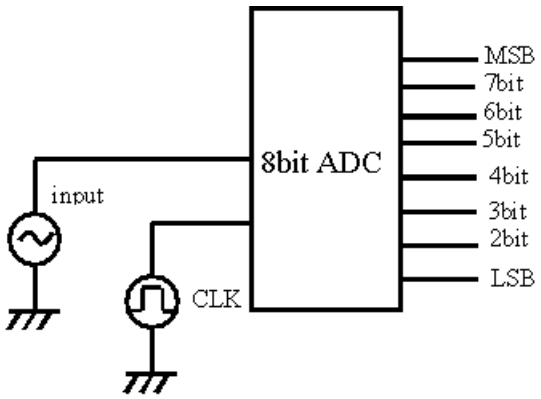


Figure 10-28 8 bit ADC Block

Verilog-A Source (adc.va)

```
//
// Pipelined ADC
//

‘include "discipline.h"

module adc(in,clk,out);

  parameter integer bit       = 8 from [1:inf);   // ADC resolution
  parameter real    fullscale = 5.0;             // supply voltage
  parameter real    vth       = 2.5;              // threshold
  parameter real    dly       = 1n;             // transition delay
  parameter real    ttime     = 1n;                  // transition
rising time
  parameter real   outhigh   = 5.0,
                   outlow    = 0.0 from (-inf:outhigh); //high and
low levels

   input                in;        // input analog voltage
   input                clk;       // input clock

   output [bit-1:0]     out;        // digital vector output

   electrical           in, clk;
   electrical [bit-1:0] out;

   real                 sample;
   real                 result[bit-1:0];   // integer array

   genvar              i;                       // index loop for
analog_for

   // Perform the conversion at each rising clock edge.
   analog
   begin
      @( cross(V(clk)-vth, +1) )
      begin
         sample = V(in);
         for( i = bit-1; i >= 0; i = i-1 )
         begin
            if( sample > vth )
              begin
                 result[i] = outhigh;
                 sample = sample - vth;
              end
            else
              begin
                 result[i] = outlow;
              end
            sample = 2.0 * sample;
         end
      end

// Update the digital output with the result of the conversion,
// using a transition delay and a rise time.
```

```
// Analog_for  - using genvar j index loop for transition()

      for (i = 0; i < bit; i = i+1 )
         V(out[i]) <+ transition(result[i], dly, ttime);

   end

endmodule
```

Spice Input Deck (adc.in)

```
* Pipelined ADC : 8 Bits.

.verilog "adc.va"

* Input Waveform
vin       in  0 sin   (2.5 2.5 500k 0 0 )

* Clock Signal
vclk      clk 0 pulse (0 5 10n 1n 1n 5n 10n)

* Instance ADC
YVLG_ADC  in clk bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0 adc bit=8
fullScale=5

* Output Loading
RLoadMSB  bit7 0 1MEG
RLoadbit7 bit6 0 1MEG
RLoadbit6 bit5 0 1MEG
RLoadbit5 bit4 0 1MEG
RLoadbit4 bit3 0 1MEG
RLoadbit3 bit2 0 1MEG
RLoadbit2 bit1 0 1MEG
RLoadlsb  bit0 0 1MEG

* Analysis
.tran 1n 2u

* For output
.let7bit='v(bit7)/5+20'
.let6bit='v(bit6)/5+18'
.let5bit='v(bit5)/5+16'
.let4bit='v(bit4)/5+14'
.let3bit='v(bit3)/5+12'
.let2bit='v(bit2)/5+10'
.let1bit ='v(bit1)/5+8'
.let0bit ='v(bit0)/5+6'

* Plot Waveforms
.iplot v(in) v(clk) 7bit 6bit 5bit 4bit 3bit 2bit 1bit 0bit

.end
```

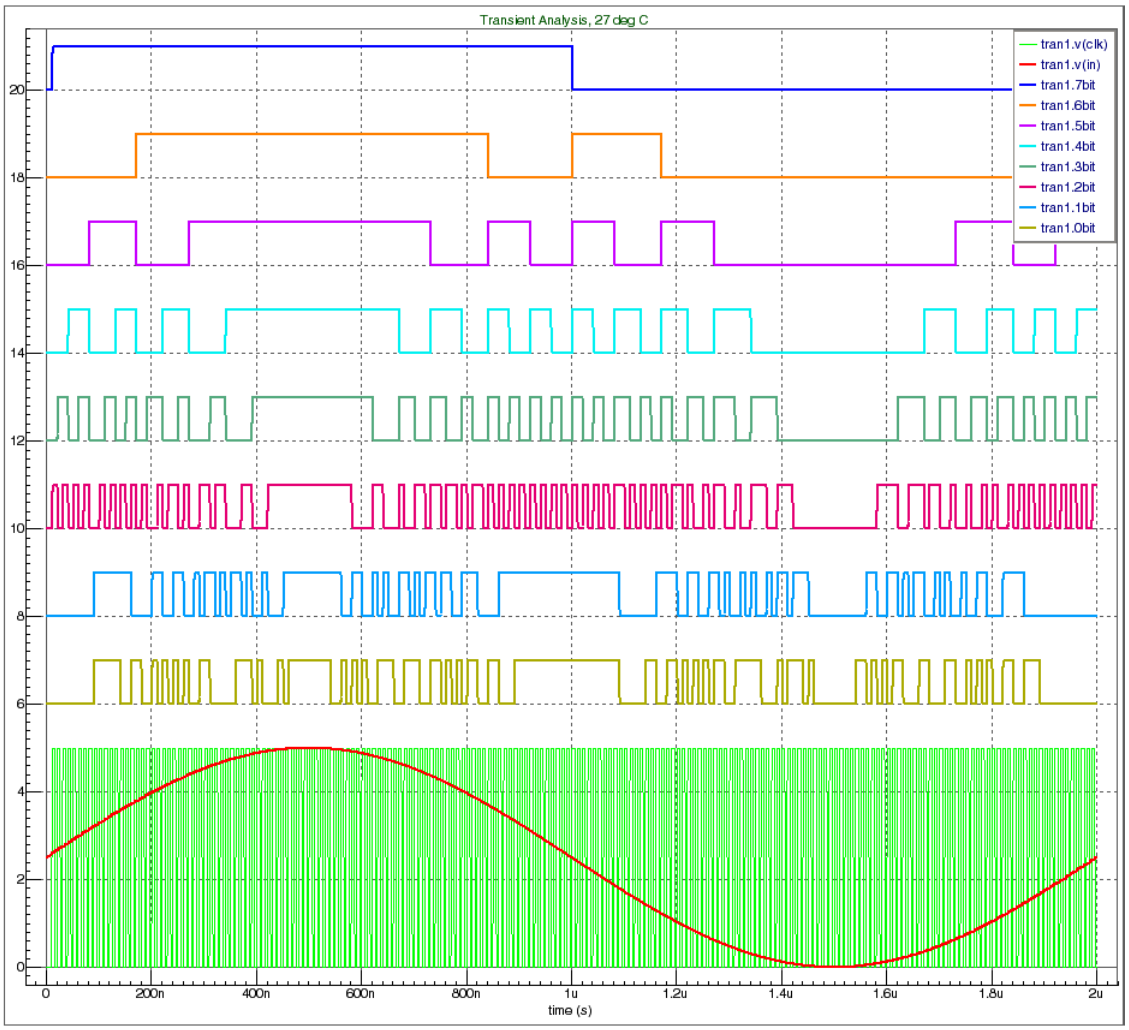


Figure 10-29 ADC Simulation Result

10.2.7 Delta-Sigma Modulator

A Delta-Sigma consists of number blocks arranged in a feedback loop, as shown in Figure 10-30. The first block is the summing block, the second block is the integrator, the third block is a quantizer, and the final block is 1 bit DAC.

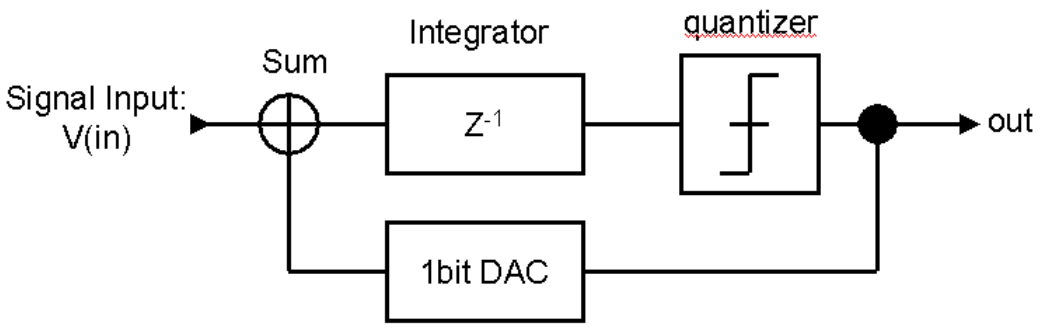


Figure 10-30 Delta-Sigma Modulator

All blocks are driven by a sampling clock signal (CLK).

Verilog-A Source (delta_sigma_mod.va)

```
//
// First Order Delta Sigma Modulator
//

‘include "discipline.h"

module delta_sigma(in,clk,out);

 parameter real quantizer_vth = 2.5;
 parameter real clk_vth       = 0;
 parameter real d2a_gain      = 1.0;

 input   in, clk;
 output  out;

 voltage in, clk, out;

 real    vsum, vd, vint, vout, hi, lo;

 analog
 begin
   @(initial_step)
   begin
     vd   = 0;
     vint = 0;
     vout = 0;
     hi   = 5;
     lo   = 0;
   end

   @(cross( V(clk) - clk_vth, 1 ))
     begin
// summing junction
```

```
        vsum = V(in) - vd;
    // integrator
        vint = vint + vsum;
    // quantizer
        if (vint > quantizer_vth) vout = hi;
        else                      vout = lo;
    // D to A
        vd = d2a_gain * vout;
      end

    V(out) <+ vd;
  end

endmodule
```

Spice Input Deck (sdelat.in)

```
* Analog First Order Delta Sigma Modulator

.verilog "delta_sigma_mod.va"

* Input Waveform
vin  in  0 sffm (2.5 2.5 44.1k 1000 10 )

* Clock Waveform
vclk clk 0 pulse(0 5 10n 0.1n 0.1n 10n 20n)

* Instance Delta-Sigma
YVLG_DeltaSigma in clk out delta_sigma

* Output Loading
RLoad out 0 1MEG

* Analysis
.tran 0.1 18.5u

* For Output
.let in='v(in)+6'
.let out='v(out)+6'

* Plot Waveforms
.iplot  v(clk) in out

.end
```

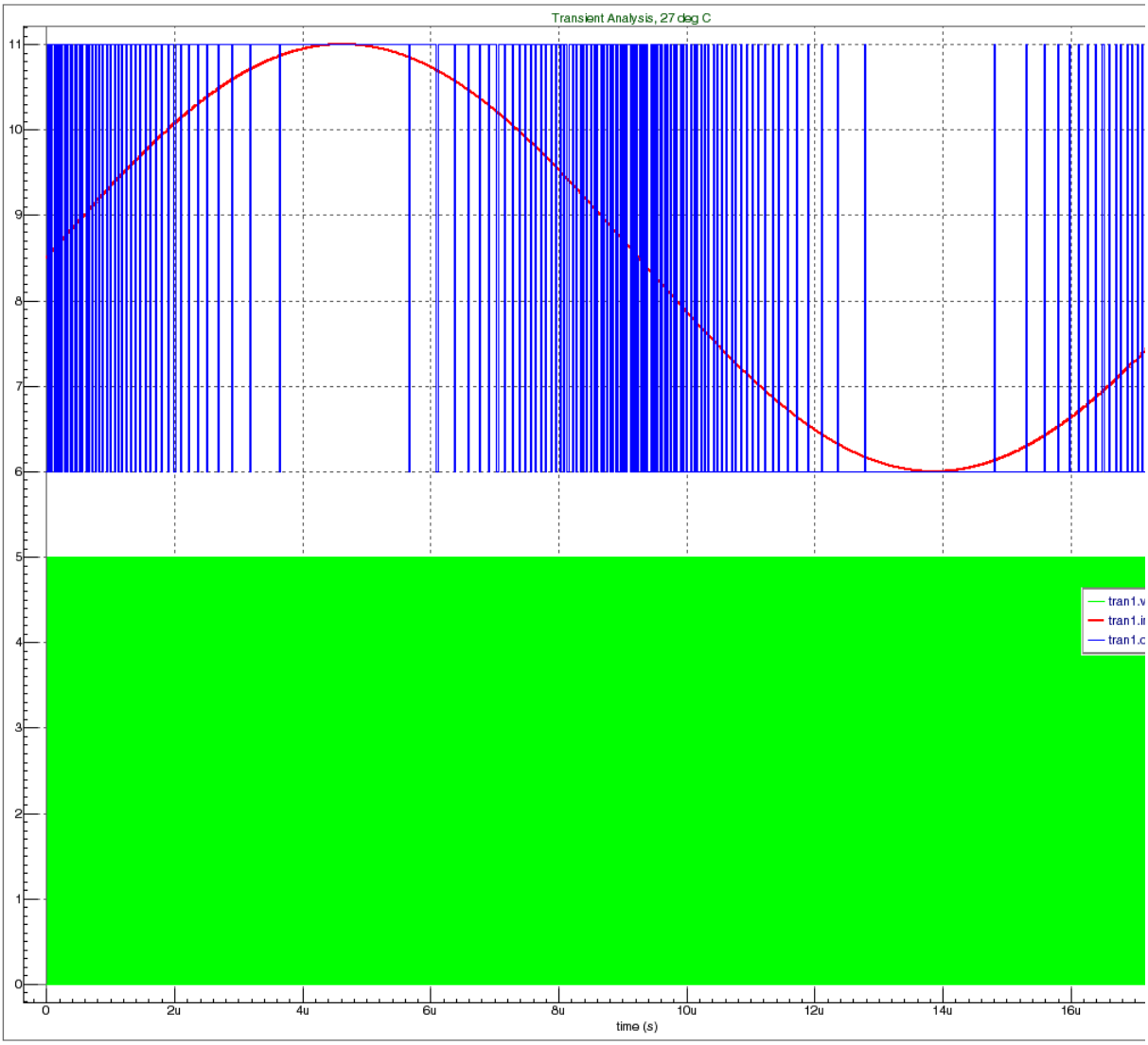


Figure 10-31 Delta-Sigma Simulation Result

10.3 Advance Electrical Models

This section describes the advance electrical models and technique for creating and simulating such components with SmartSpice.

10.3.1 D-type Flip-Flop with CLEAR button (Accurate Cell Model)

This example shows some techniques for creating an accurate Verilog-A module of Data Flip-Flop cell. An accurate module can be realized by introducing complex delay and rise/fall time models, which are a function of the output load and ramp-time of the clock pulse. Another illustrated technique is the use of SPICE primitives such as capacitor. In this example, capacitors are instantiated inside the module representing the load capacitance of module terminals.

A practical example of DFF module is shown below.

Example: DFF Module Description (dff_cl.va)

```
//
// D-FF with CLEAR
//   Tpd, Tr and Tf are extracted from the result of device
simulation
//  using BSIM3v3.2 default parameter set and nominal condition.
//

‘include  "discipline.h"

module dff_cl(d, clk, clr, q);

    input         d, clk, clr;
    output        q;
    electrical    d, clk, clr, q;

    parameter  real  VDD        = 3.3,      THRESH     = 0.5,
                     LO_THRESH  = 0.2,      HI_THRESH  = 0.8,
                     FO         = 2.0,      SLOPE_IN   = 0.09n;
    parameter  real  TR_CLK_0   = 38.7647p, TR_CLK_1 = 244.058p,
                     TF_CLK_0   = 259.585p, TF_CLK_1 = 128.525p,
                     TF_CLR_0   = 276.347p, TF_CLR_1 = 122.161p,
                     TD_L2H_0   = 937.706p, TD_L2H_1 =-163.297p,
                     TD_H2L_0   = 831.739p, TD_H2L_1 =-76.1828p,
                     TD_CLR_0   = 184.611p, TD_CLR_1 =-67.1826p,
                     SLOPE_COEF = 0.33033,   SLOPE_NOM = 0.09n;
    parameter  real  C_DATA     = 9.77591e-15,
                     C_CLK      = 9.98186e-15,
                     C_CLR      = 12.4037e-15;

   real    vth, hl_thresh, tr_clk, tf_clk, tf_clr, td_l2h, td_h2l,
td_clr;
    real     td, tr, tf;

    integer  state;

    capacitor #(.c(C_DATA))  cdata(d);
    capacitor #(.c(C_CLK))   cclk(clk);
    capacitor #(.c(C_CLR))   cclr(clr);

    analog
```

```
    begin
        @(initial_step)
  begin
            hl_thresh = HI_THRESH-LO_THRESH ;
            vth    = VDD*THRESH ;
            state  = 0 ;
            tr_clk = (TR_CLK_0+TR_CLK_1*FO)/hl_thresh ;         //
Rise time of Q triggered by CLK
            tf_clk = (TF_CLK_0+TF_CLK_1*FO)/hl_thresh ;         //
Fall time of Q triggered by CLK
            tf_clr = (TF_CLR_0+TF_CLR_1*FO)/hl_thresh ;         //
Fall time of Q triggered by CLR
              td_l2h = TD_L2H_0+TD_L2H_1*FO+SLOPE_COEF*(SLOPE_IN-
SLOPE_NOM) ;    // Delay of high data
              td_h2l = TD_H2L_0+TD_H2L_1*FO+SLOPE_COEF*(SLOPE_IN-
SLOPE_NOM) ;    // Delay of low data
              td_clr = TD_CLR_0+TD_CLR_1*FO+SLOPE_COEF*(SLOPE_IN-
SLOPE_NOM) ;    // Delay of CLR
            tr     = tr_clk;
            tf     = tf_clk;
            td     = td_l2h;
          end

        @(cross(V(clk)-vth,+1))
          begin
            if(V(clr) < vth)
              state = 0;
             else
              state = (V(d) > vth);

            if(state)
              td = td_l2h;
              else
        begin
                  td  = td_h2l;
                  tf  = tf_clk;
                end
          end

        @(cross(V(clr)-vth,-1))
          begin
            state = 0;
            td    = td_clr;
            tf    = tf_clr;
          end

        V(q) <+ transition(VDD*state, td, tr, tf);
    end

endmodule
```

In the module description, three capacitors are introduced to represent the input capacitance of module ports; `d`, `clk` and `clr`. The capacitance values are derived from transistor level simulation, and should match to the equivalent gate capacitance of MOSFET at each module

port. For a rough estimation, the following capacitance equation can be used to calculate gate capacitance.

```
Cgate = EPSox*L*W/TOX
```

where:

- EPXox is gate oxide permittivity = 3.453143e-11 [F/m].
- L and W are channel length and width in meters.
- TOX is gate oxide thickness in meters.

Another technique for achieving an accurate model is the introduction of a complex formula to model delay time and rise/fall transition time of a flip-flop. Generally, the delay time is a function of fanout and ramp-time of the clock signal, and can be modelled as follows;

```
Tdelay = T0 + T1*FANOUT + T2*SLOPE
```

where:

- FANOUT is the number of fanout drived by DFF output.
- SLOPE is ramp-time of the clk pulse.
- T0 is the delay time at FANOUT=0 and SLOPE=0.
- T1 is coefficient of fanout dependency.
- T2 is coefficient of ramp-time dependency.

In this example, the parameter SLOPE is replaced by (SLOPE - SLOPEnom), where SLOPEnom is the nominal slope at which T0 and T1 are extracted.

The transition time is formulated in the same manner as delay time, except the fanout dependency term is excluded, and is defined as following formula;

```
Ttrans = TT0 + TT1*FANOUT
```

where:

- FANOUT is the number of fanout driven by DFF output.
- TT0 is transition time at FANOUT=0.
- TT1 is coefficient of fanout dependency.

Note that since the delay time of high and low data transitions are not identical, the different delay parameters are introduced to calculate high and low data transitions. In addition, different delay parameters are introduced to calculate the clear time.

10.3.2 Shift Register Circuit based on DFF. Simulation

An 8 bit shift register circuit (used DFF Verilog-A instances described above) is shown in Figure 10-32.

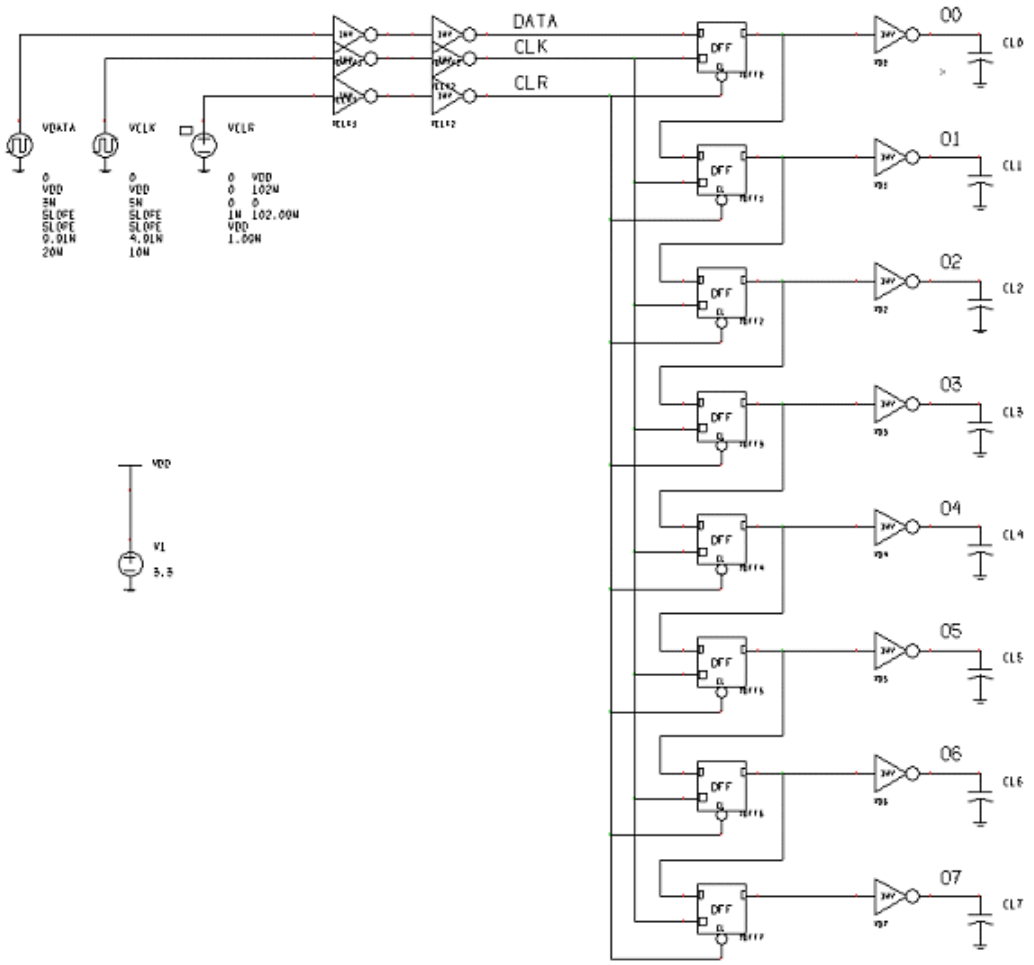


Figure 10-32 8 Bit Shift Register

Example: Input Deck of 8-bit Shift Register with DFF Verilog-A instances (DFF_example.in)

```
* DFF with Clear SPICE primitive

.verilog "dff_cl.va"

* Options
.OPTIONS  ACCT  NOMOD

*{ Parameters begin ------------------------------------

* PSU Voltage
.param VDD=3.3

* Waveform Rise and Fall Times
.param SLOPE=0.09N

* DFF Shift Register Rise and Fall Times
```

```
.param SLOPE_VAL=308p

*} Parameters end ----------------------------------------

* Globals
.global VDD  GND

*{ Subcircuit begin --------------------------------

* Inverter Subcircuit
.subckt INVERTER IN OUT
 MP1 OUT IN GND GND NM L=LN W=WN
 MP2 OUT IN VDD VDD PM L=LP W=WP
.ends INVERTER

*} Subcircuit end ----------------------------------

*{ Voltage sources begin ----------------------------------

* Power Supply
V1 VDD 0 DC 3.3

* Clear Signal
VCLR CLR 0 PWL(0 0 1N 0 1.09N VDD 102N VDD 102.09N 0 )

* Clock Signal
VCLK CLK 0 PULSE(0 VDD 5N SLOPE SLOPE 4.91N 10N)

* Data Signal
VDATA DATA 0 PULSE(0 VDD 3N SLOPE SLOPE 9.91N 20N)

*} Voltage sources end ----------------------------------------

*{ X-calls begin ------------------------------------------------
--

* Data Driver
XDATA1 DATA NET1    INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
XDATA2 NET1 DATA_VA INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U

* Clock Driver
XCLK1  CLK  NET2    INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
XCLK2  NET2 CLK_VA  INVERTER  PARAMS: WP=28U LP=0.4U WN=20U LN=0.4U

* Clear Driver
XCLR1  CLR  NET3    INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
XCLR2  NET3 CLR_VA  INVERTER  PARAMS: WP=28U LP=0.4U WN=20U LN=0.4U

* Output Inverters to create Qbar
XQ0    Q0   O0_VA   INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
XQ1    Q1   O1_VA   INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
XQ2    Q2   O2_VA   INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
XQ3    Q3   O3_VA   INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
XQ4    Q4   O4_VA   INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
XQ5    Q5   O5_VA   INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
XQ6    Q6   O6_VA   INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U
```

```
XQ7    Q7   O7_VA   INVERTER  PARAMS: WP=7U  LP=0.4U WN=5U  LN=0.4U

*} X-calls end -------------------------------------------------
--

*{ Capacitive loading begin ------

CL0 O0_VA 0  '0.02PF'
CL1 O1_VA 0  '0.02PF'
CL2 O2_VA 0  '0.02PF'
CL3 O3_VA 0  '0.02PF'
CL4 O4_VA 0  '0.02PF'
CL5 O5_VA 0  '0.02PF'
CL6 O6_VA 0  '0.02PF'
CL7 O7_VA 0  '0.02PF'

*} Capacitive loading end --------

*{ Verilog-A instances DFF begin ---------------------------------
--

YVLG0    DATA_VA    CLK_VA    CLR_VA    Q0    DFF_CL       FO=2.03085
SLOPE_IN=SLOPE_VAL
YVLG1   Q0              CLK_VA   CLR_VA   Q1   DFF_CL     FO=2.03085
SLOPE_IN=SLOPE_VAL
YVLG2   Q1              CLK_VA   CLR_VA   Q2   DFF_CL     FO=2.03085
SLOPE_IN=SLOPE_VAL
YVLG3   Q2              CLK_VA   CLR_VA   Q3   DFF_CL     FO=2.03085
SLOPE_IN=SLOPE_VAL
YVLG4   Q3              CLK_VA   CLR_VA   Q4   DFF_CL     FO=2.03085
SLOPE_IN=SLOPE_VAL
YVLG5   Q4              CLK_VA   CLR_VA   Q5   DFF_CL     FO=2.03085
SLOPE_IN=SLOPE_VAL
YVLG6   Q5              CLK_VA   CLR_VA   Q6   DFF_CL     FO=2.03085
SLOPE_IN=SLOPE_VAL
YVLG7   Q6              CLK_VA   CLR_VA   Q7   DFF_CL     FO=1.34995
SLOPE_IN=SLOPE_VAL

*} Verilog-A instances DFF end -----------------------------------
--

* Transistor Model Parameters (Default Model)
.model  PM  PMOS  LEVEL=49
.model  NM  NMOS  LEVEL=49

* Analysis
.tran  1P  120N

* Plotting

.save  V(O1_VA)  V(O0_VA)  V(CLR_VA)  V(CLK_VA)  V(O7_VA)  V(O5_VA)
V(O6_VA)     +V(O3_VA) V(O4_VA) V(O2_VA)

.end
```

Simulation Results

Simulation results are shown in Figure 10-33 through Figure 10-35. These figures show that the waveforms obtained from Verilog-A simulations (solid lines) matched to those obtained from transistor level simulations (dotted lines).

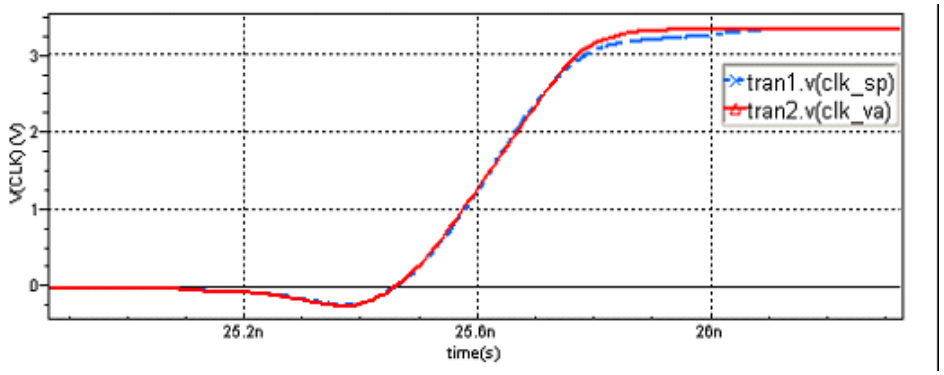


Figure 10-33 CLK Signal Simulation Results

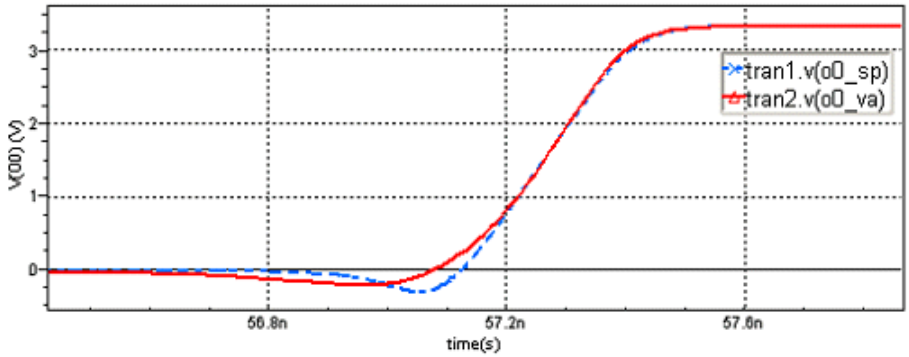


Figure 10-34 Output Signal (Rise) Simulation Results

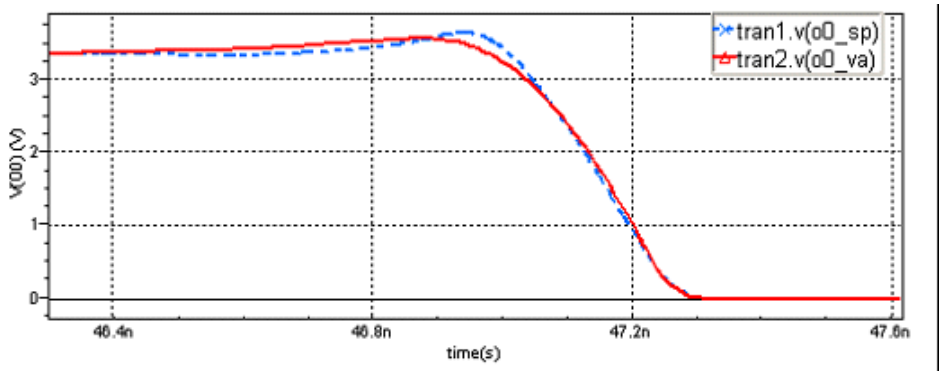


Figure 10-35 Output Signal (Fall) Simulation Results

Chapter 11
Continuous Model Approach

11.1 Introduction

Normally a single active device model is extracted to cover a range of device geometries and temperature. Sometimes this single scalable model is not sufficient to describe all the changes in output characteristics over the range of geometry and temperature required. The total range of geometry and temperature is then broken up into regions, and a model produced for each of these sub-sets of devices. This is the basis of a binned model and can lead to discontinuities at the bin boundaries as the model card is changed. To get around this problem SmartSpice has introduced a new functionality to allow you to go back towards a single scalable model card by using a function rather than a single parameter value. This new powerful algorithm allows you to specify a formula linking in other device model card parameters, giving a continuous multi-dimensional function. This then gets around the problems of binning and gives the potential for a more accurate model fit to the device output characteristics.

11.1.1 Setting Up the Model's Parameter

SmartSpice offers a new algorithm for setting up a model for SOI, TFT, MOSFET, JFET, MESFET, bipolar and diode active devices. This new feature is intended to be a substitute for the binning scheme, and gives a more powerful and accurate device modeling mechanism. This algorithm allows a user to specify a formula for any model card parameter. The formula describes a wide operating range of data, and the target model parameter is represented by a continuous multidimensional function.

Syntax

```
m0 B0_x10 adr7 B0_x11 vss nenh l=5.1e-06 w=2.11e-06
.model nenh NMOS
...
+  VERSION=3.1
+  Vth0='0.9302 + 0.125E-012/(w*l) +0.001*l-0.012*w'
...
```

In the example above, the model card parameter `Vth0` is driven by width and length dependency, which comes from the instance's set of parameters. For device `m0` SmartSpice will use a model with the parameter `Vth0` calculated using *l=5.1e-06* and *w=2.11e-06*.

Syntax

```
mp y a vdd vdd TP w='wpp' l='lpp' custompar='sj'
.MODEL          TP          PMOS          (LEVEL=3          VTO='-0.8/
sqrt(W*L*3.33333E10*sqrt(9*custompar)/3)'
+ TOX=0.41E-7 UO=150
```

In the example above, the model card parameter `VTO` is driven by `custompar` which is not an instance parameter. `custompar` might be also any native instance parameter. If above syntax is used, SmartSpice will not complain about `custompar` used in the instance line if `custopar` is used in the expression of model parameter and if the model is used by corresponding instance.

11.1.2 SmartSpice Shell Command for Checking Up the Model's Parameter Table

All devices are grouped with respect to the arguments of the formula used for model parameter. To see how many different groups are created and the target model parameter, SmartSpice offers the shell command `cmcstat`. Output after the use of `cmcstat` is shown below:

```
****internal information about CMC format model use
****
Total Overhead During Simulation =   0.000000
Model nenh(model in use)
       vth0=0.9302 + 0.125e-012/(w*l) +0.001*l-0.012*w
group 0  W=1.050000e-05 L=1.440000e-06      9.384671e-01
group 1  W=1.050000e-05 L=1.450000e-06      9.384101e-01
group 2  W=1.050000e-05 L=1.460000e-06      9.383538e-01
group 3  W=2.100000e-06 L=1.100000e-06      9.843125e-01
group 4  W=2.110000e-06 L=2.100000e-06      9.584103e-01
group 5  W=2.120000e-06 L=1.100000e-06      9.838020e-01
group 6  W=2.130000e-06 L=1.100000e-06      9.835504e-01
group 7  W=2.140000e-06 L=1.100000e-06      9.833011e-01
group 8  W=2.150000e-06 L=1.100000e-06      9.830541e-01
group 9   W=2.160000e-06 L=1.100000e-06         9.828094e-01
```

11.1.3 User Oriented Error and Warning Messages

Conflict of names in the .PARAM and Formula Arguments

The continuous model approach algorithm reports errors. In the case where the same names for a parameter in a `.PARAM` statement and arguments in model card parameter formula model's parameter (Berkeley approach) have been used, SmartSpice will issue the following warning.

Example

```
.param w=1
.model nenh NMOS
+  Vth0='0.9302 + 0.125E-012/(w*l) +0.001*l-0.012*w'
```

Warning

```
New model format in use: member [ w ]  in the expression [ 0.9302 +
0.125e-012/(w*l)  +0.001*l-0.012*w  ]  of  model  [nenh](file  ./
bsim3v3.mod/home/user/MYINDECK/CMC/bsim3v3.mod)  is  treated  as
device geometry and overrides the value of parameter [w]  in the
.PARAM
```

You must check either the formula in the `.model` card, or the parameter in the `.PARAM`.

Arguments in the formula for model's parameter do not correspond to the set of device's parameters

Example

```
q1 cq bq gnd1 gnd QNLREF m=1  areac=45
.model qnlref npn (
...
+  BF='0.5+area+1'
```

Fatal Error:

```
CMC model: Model qnlref can not find instance parameter 'area' in
the specified device's line
  q1 cq bq gnd1 gnd qnlref m=1 areac=45
  used in the expression '0.5+area+1'
Error on lines:
  29 : .model  nlref  npn (
  30 : *+   BF=100
  31 : +    BF='0.5+area+1'
...
cannot evaluate expression '0.5+area+1'
```

SmartSpice's System Internal Messages

SmartSpice prints the error message "Device type "CAP32" is not supported in the enhanced modeling algorithm" if the parser found inconsistencies during the parser phase. In that case, check the `.model` card for device CAP32.

System message "CMC model: module found fatal error, parameters table and correspondent values table are not synchronized" is issued by SmartSpice if the internal engine fails to find a corresponding entry in the internal tables. Report this message to the SmartSpice support team.

11.1.4 Multithreading Support in the Continuous Model Approach

`MODELALG=0` and `MODELALG=1` support multithreading in SmartSpice. You do not need to specify any extra input for SmartSpice, only `-P n`, where `n` is the number of CPUs as normal.

11.1.5 Integration with Parametric Analysis

Continuous model algorithm is integrated with .MODIF analysis. The command `cmcstat` prints parametric information. Internal tables will be populated with a new entry; old ones are kept in the table.

Example

```
.modif loop=10 m1(l)+=(2.0u)0.2u
****internal information about CMC format model use
****
Total Overhead During Simulation =   0.000070

Model n.85
               u0=0.0600648+0.02*(l-2u)/4u

group   0     W=1.800000e-06   L=2.200000e-06          6.106480e-02
0.000000e+00(step 0) 6.106480e-02(step 1)

group   1     W=1.800000e-06   L=2.400000e-06          6.206480e-02
0.000000e+00(step 0)

group 2  W=1.800000e-06 L=4.000000e-06    7.006480e-02  7.006480e-
02(step 0) 7.006480e-02(step 1)  7.006480e-02(step 2)

Model p.85
               u0=0.0219015+w+l
group 0  W=3.600000e-06 L=1.200000e-06   2.190630e-02   2.190630e-
02(step 0) 2.190630e-02(step 1)  2.190630e-02(step 2)

****Parametric data on each step:****
step 0,  modified parameter:  m1[l] =2.200000e-06
step 1,  modified parameter:  m1[l] =2.400000e-06
```

11.1.6 AKO Type of Model in the Continuous Model Approach

A new model approach is integrated with the AKO type of model. A correct set of expressions is used by SmartSpice in the AKO derivative model.

Example

Regular Model

```
.MODEL  NE  NMOS  (VTO = '5.0 + 0.7 / (w * l)' LAMBDA = '0.1+w*l'
KP = '20E-6 + w*l'   GAMMA=0.37   NSUB=5E14 TOX=0.1U LD=1.0U CJ=70U
CJSW=220P CGSO=345P  CGDO =345P )
```

AKO Model

```
.model NE2 AKO:NE NMOS (VTO = '1.0 + 5E-012/(w*l)')    KP = '30E-6
+ w*l'
.model NE3 AKO:NE NMOS (VTO = '1.0 + 5E-012/(w*l)')    LAMBDA =
'0.77+w*l'
```

Table information is generated by the SmartSpice shell command cmcstat:

```
--> cmcstat

****internal information about CMC format model use
****
Total Overhead During Simulation =   0.000000

Model ne (model not in use)

                         vto=5.0 +  0.7 / (w * l)    lambda=0.1+w*l
kp=20e-6 + w*l
group  0  W=3.000000e-05  L=2.000000e-05  1.166667e+09  1.000000e-01
2.000060e-05
group  1  W=4.000000e-05  L=2.000000e-05  8.750000e+08  1.000000e-01
2.000080e-05

Model ne2 (model in use)
                           vto=1.0 + 5e-012/(w*l)    lambda=0.1+w*l
kp=30e-6 + w*l
group  0  W=4.000000e-05  L=2.000000e-05  1.006250e+00  1.000000e-01
3.000080e-05

Model ne3 (model in use)
                          vto=1.0 + 5e-012/(w*l)    lambda=0.77+w*l
kp=20e-6 + w*l
group  0  W=3.000000e-05  L=2.000000e-05  1.008333e+00  7.700000e-01
2.000060e-05
```

11.1.7 Enhancements of the Continuous Modeling Approach

The set of allowed instance parameters which can be used in the formula for model card parameters is the subject for change. They are displayed in the table for each model.

11.1.8 Integration with Statistical Functions

A new model approach is integrated with the built-in SmartSpice statistical functions where parameters are a function of the instance's parameters.

After sourcing, the shell command cmcstat displays the value and expression of the model parameter, which is a function of instance's parameter.

```
.param sigma =1 multiplier=1
.param first ='0.298213+dvthn'
.param second = '3*avtn/sqrt(1e12*w*l)'
.param VTH4=AGAUSS(first,second,sigma,multiplier)
.MODEL nch.4 nmos ( LEVEL=49  VERSION = 3.2
+VTH0=VTH4
****internal information about CMC format model use
****
Total Overhead During Simulation =   0.000000

Model nch.4 (model in use)

vth0=first second sigma multiplier
group        0               W=1.000000e-05        L=1.000000e-05
-3.579578e-01
group        1               W=1.000000e-05        L=2.000000e-05
```

```
1.659430e+00
```

11.1.9 Integration with Binning Algorithm

A new model approach is integrated with the SmartSpice binning scheme.

Example

```
.MODEL nch.4 nmos ( LEVEL=49  VERSION = 3.2
+VTH0=VTH4
```

It demonstrates the use of `VHT0` as a function of `W/L`. The binning scheme works in a regular SmartSpice method where the model's parameters can be a function of allowed instance's parameters.

11.1.10 Scoping Rules for CMA

CMA model parameters, which are functions of parameters or instance's parameters, resolve the scoping of names with the following rules:

1. Look for instance's parameter
2. Look for local `.param`
3. Look for global `.param`

Example

```
.param m=1
.model model R vlx='1.5*m'
r 1 2 model L=7u W=6u m=4

.param m=1
.model model R vlx='1.5*m'
r 1 2 model L=6u W=4u
```

In the first example, `m=4` will be used to calculate model parameter `vlx`.

In the second example, `m=1` will be used to calculate model parameter `vlx`.

11.2 User Specified Runtime Model Parameters

Runtime expressions (expressions which depend on integration time (TIME), integration timestep (TSTEP), frequency (HERTZ), node voltages, voltages across two nodes and branch currents) can be used to change the model parameter values during the simulation.

Example

```
.OPTIONS nomod nodeck post=2 probe

.PARAM vto_val = "v(nm1)*3.5"

VVDD VDD 0 DC  5V
VVSS VSS 0 DC -5V
VINP INP 0 DC -5 PULSE ( -5 5 120e-9 500e-9 500e-9 700e-9 2e-6 )
VMOD NM1 0 PWL ( 0 0 0.5us 0 1us 1 1.5us 0 )
RMOD NM1 0 1

Xpas1 VDD INP OUT1 VSS INVER

.SUBCKT INVER VDD INPUT OUTPUT VSS
M3 VDD INPUT OUTPUT PP W=100E-6 L=10E-6
M4 OUTPUT INPUT VSS NN W=50E-6 L=10E-6
.ENDS INVER

.MODEL NN nmos (level=36 version=1 vto="1.5+vto_val")
.MODEL PP pmos (level=36 version=1 vto="1.5+vto_val")

.tran 0.001us 2us

.probe v(out1)

.END
```

In this example, the model parameter VTO is changed during the simulation run.

11.3 Advanced Runtime Model Parameters and Probing Mechanism

Usually runtime expressions depend on integration time (TIME), integration timestep (TSTEP), frequency (HERTZ), node voltages and branch currents. In advanced algorithms mode, parameters can be depended on other model/instance parameters or variables, that can be changed during simulation. Additionally some event mechanism can be requested. To support such functionality, the algorithms statement .REALSAVE and a probing mechanism can be used.

11.3.1 .REALSAVE (Save Specific Runtime Vectors)

Syntax

```
.REALSAVE <outvar1 <outvar2>>
```

This statement causes SmartSpice to process variables outvar1, outvar2 at runtime. Model parameters depended on outvar1, outvar2 are also runtime dependant and will be reevaluated during simulation.

Example

```
.realsave @m1[vth]   @m1[vto]
.model …
+vto = 'func(@m1[vth]   , @m1[vto])'
```

Probing mechanism contains a set of measure statements usually defined through ternary operators. During transient analysis at each time point SmartSpice calculates measures and updates their properties. The following measure probe properties are available:

@<measure name>.val	value of measure
@<measure name>.trig1	last time point, where ternary operator condition was switched from false to true
@<measure name>.trig0	last time point, where ternary operator condition was switched from true to false

Note: The option autostop must be set to -1 to activate measure probing mechanism.

Example

```
* Probing mechanism
.tran 0.001us 2us
.OPTIONS nomod nodeck probe  autostop=-1

VVDD VDD 0 DC  5V
VVSS VSS 0 DC -5V
Vinp   inp  0   pwl (0    -5      500n    -5    501n   5       1500n
5     1501n   -5    r = 0  )
Vprobe   pro  0   pwl (0    0      1000n    0    1001n    1   2000n
1      r = 0  )

.meas tran probe expr val='(v(pro) >= 1  )  ? 1 : 0'
.realsave   probe  probe.trig1   probe.trig0
.PARAM vto_val = "3.5"
```

```
.define mode1 '0.1-0.1*exp(-(TIME - 2* @probe.trig1)/0.5u)+vto_val'
.define mode2 '0.1-0.1*exp((TIME + 2*@probe.trig0)/0.5u)+vto_val'
.define VTOmode    '(@probe.val)?  mode1 :   mode2 '

X1 VDD INP OUT1 VSS INVER
.SUBCKT INVER VDD INPUT OUTPUT VSS
M1 VDD INPUT TEMP1 PP W=100E-6 L="10E-6"
M2 TEMP1 INPUT VSS NN W=50E-6 L="10E-6"
M3 VDD TEMP1 TEMP2 PP W=100E-6 L="10E-6"
M4 TEMP2 TEMP1 VSS NN W=50E-6 L="10E-6"
M5 VDD TEMP2 OUTPUT PP W=100E-6 L="10E-6"
M6 OUTPUT TEMP2 VSS NN W=50E-6 L="10E-6"
.MODEL NN nmos (level=36 version=1
+ vto =  '((time == 0) ? vto_val : VTOmode ) '
+ capmod=0 asat=1 at=3e-8 blk=1e-8 bt=0 dasat=0 dd=1e-7
+ delta=6 dg=1e-8 dmu1=0 dvt=1.5 dvto=0 eb=0.56 eta=15 io=50
+ ioo=5000 lasat=0 lkink=2.2e-5 mkink=2 mmu=3.2 mu1=8e-5
+ muo=32 mus=1.5 rd=0 rs=0 tox=1e-7 vfb=-3.5 vkink=9.1 von=3
+ )
.MODEL PP pmos (level=36  version=1
+ vto =  '((time == 0) ? vto_val : VTOmode ) '
+ capmod=0 asat=1 at=3e-8 blk=1e-8 bt=0 dasat=0 dd=1e-7
+ delta=6 dg=1e-8 dmu1=0 dvt=1.5 dvto=0 eb=0.56 eta=15 io=50
+ ioo=5000 lasat=0 lkink=2.2e-5 mkink=2 mmu=3.2 mu1=8e-5
+ muo=32 mus=1.5 rd=0 rs=0 tox=1e-7 vfb=3 vkink=9.1 von=-3
+  )
.ENDS INVER
.probe v(out1)  v(inp) v(pro)  @m.x1.m2[vto]  @m.x1.m2[vth]
.END
```

The above-mentioned netlist is a simple example how to use probing mechanism.

The statements:

```
.meas tran probe expr val='(v(pro) >= 1  )  ? 1 : 0'
.define VTOmode    '(@probe.val)?  mode1 :   mode2 '
```

define measure statements and control behavior of the model parameter VTO:

```
vto =  '((time == 0) ? vto_val : VTOmode )'
```

There are two modes for VTO parameter - mode1 and mode2:

```
.define mode1 '0.1-0.1*exp(-(TIME - 2* @probe.trig1)/0.5u)+vto_val'
.define mode2 '0.1-0.1*exp((TIME + 2*@probe.trig0 )/0.5u)+vto_val'
```

During simulation at the time point 1ns the model parameter VTO changes its behavior from mode2 to mode1.

Figure 11-1 shows waveforms during transient analysis:

tran1.probe	measure vector;
tran1.@m.x1.m2[vto]	model parameter vto;
tran1.v(out1)	output vector;
tran2.v(out1)	output vector, when model parameter VTO is constant.

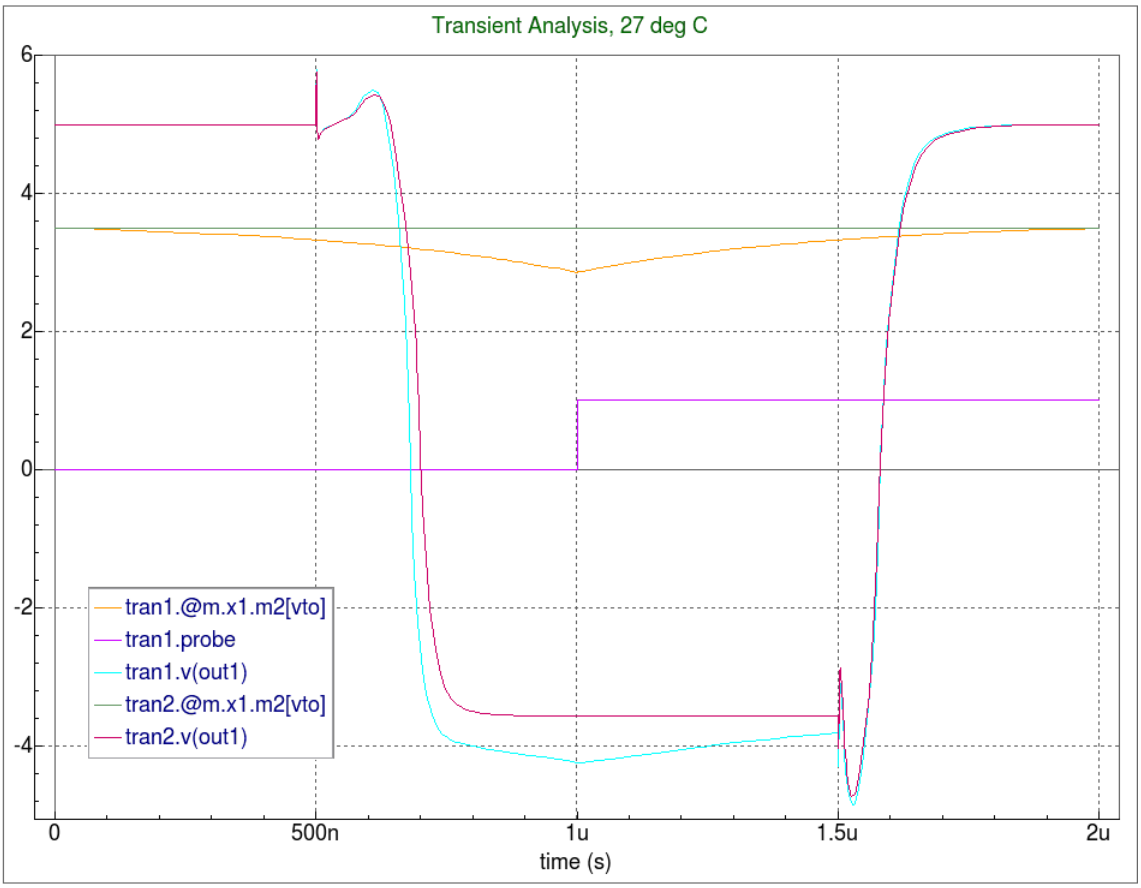


Figure 11-1 Simulation results for constant and controlled by measure probe model parameter VTO

Some advanced algorithms can request model parameter values at a specific time point. For such purpose the syntax below can be used.

Syntax

```
@<instance>[parameter]@time
```

`instance`	instance name
`parameter`	model parameter name
`time`	specific time point

Example

```
.option nomod nodeck
M1 D G S ptft W=5u L=20u
M2 D G S ntft W=5u L=20u

.param _diff='v(g) - v(s)'
.param vthz= '@m1[vth]'
.param low='-2.3-0.1*exp(time/10u)'
.param high='-2.3-0.1*exp(time/10u)+0.18*exp(time/10u)'
.param vht_threshold=0.4
Vgs G 0   -5
Vds D 0 2
Vss S 0 0
.param t1=20u  t2=25u  t3=30u

.MODEL PTFT PTFT (
+vto='(((  (abs(@m1[vto]@t1  -  @m1[vto]@t2))  >=vht_threshold)   &&
((abs(@m1[vto]@t2  -  @m1[vto]@t3))  >=vht_threshold)  )  ?      (
@m1[vto]@t3 ?  @m1[vto]@t3  :   -2.3)   :    low )'
```

Note: This expression is one continuous line, folded here for easy viewing.

In this example behavior of the parameter VTO depends on its values at 20us, 25us, and 30us time points.

Note: Variables @m1[vto]@t1, @m1[vto]@t2, and @m1[vto]@t3 are undefined until simulation gets to the corresponding time. Ternary operator condition is FALSE until some of the variables are undefined.

Additionally there is possibility to access model parameter values from previous time points. The following set of parameters have to be specified in netlist.

Example

```
.param t1=-1  t2=-2  t3=-3
```

In this example the parameter @m1[vto]@t1 means VTO value on previous time point from current simulation point and so on.

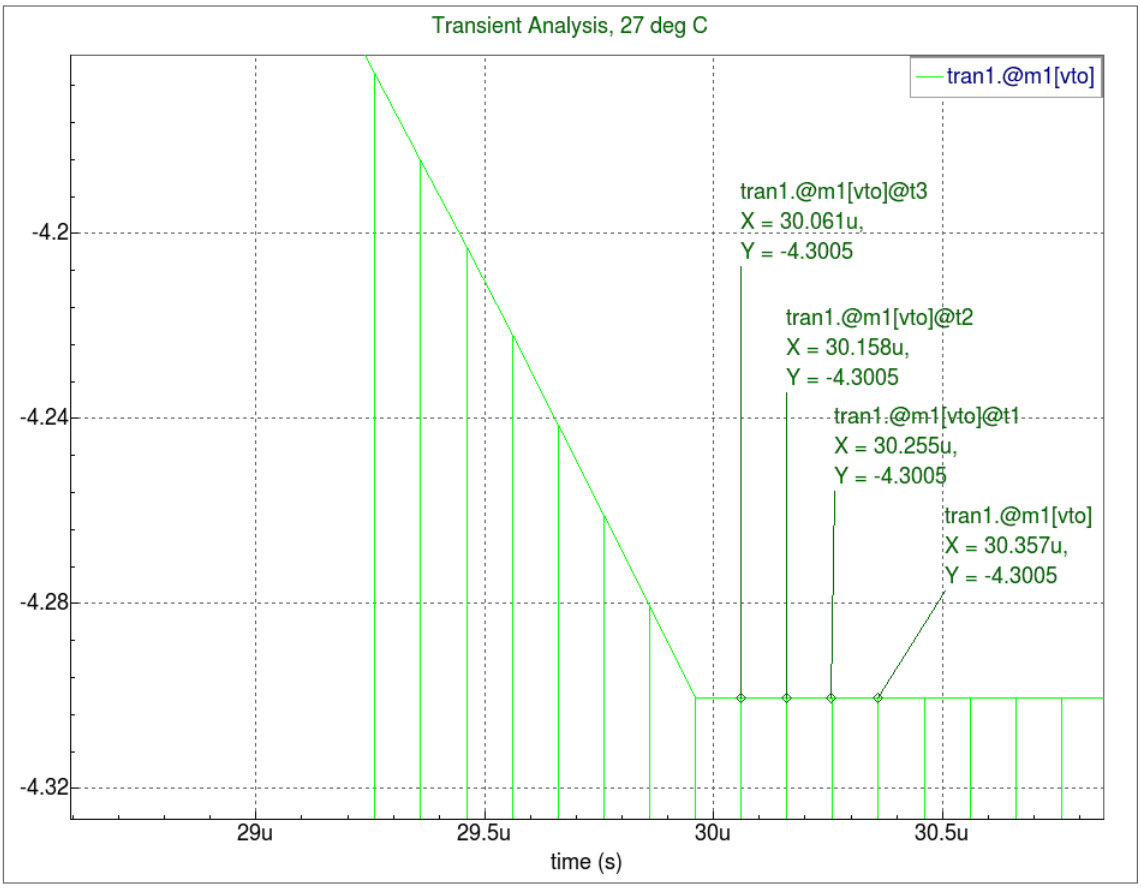


Figure 11-2 Model parameter VTO at current, @t1,@t2, and @t3 time points

Chapter 12
TSMC Modeling Interface (TMI2)

12.1 Overview

TMI is the TSMC's Model Interface for compact SPICE device modeling. TMI is written in C that can support TSMC's extensions of any standard compact models for the process technologies of 45nm and below. Customized model equations become unavoidable for advanced technologies. TMI is proposed to provide an alternative approach for this request in addition to adding more macro models or waiting for standard model updates. With TMI flow, instead of macro models, customized model equations are performed within dynamically linked library provided by TSMC. SmartSpice is TMI2 compliant which means TMI libraries can also be run.

For information about TMI workflow please consult the TSMC Modeling Interface specification document.

12.2 Connecting a Library to SmartSpice

To connect dynamic TMI library to SmartSpice, the following option statement should be added into the input deck:

```
.OPTION TMIFLAG=1 TMIPATH='path_to_tmi_install_directory'
```

You should not give a full path to the actual `.so` or `.dll` library. Instead, a path up to and including TMI model installation directory has to be supplied. SmartSpice auto-selects a platform from there and loads the required library according to the platform. For example, you have Linux 64 bit TMI libraries:

```
/home/user/sdk/TMI/RH_64/libTMImodel.so
```

then the following statements should be specified:

```
.OPTION TMIFLAG=1 TMIPATH='/home/user/sdk/TMI'
```

SmartSpice will select library automatically.

Both `TMIFLAG` and `TMIPATH` options must be set for TMI flow to be activated. Moreover, dot model statements that are intended for TMI workflow must have the `tmimodel` parameter set. For information on TMI model card setup please consult the TSMC Modeling Interface specification document and TMI example netlists.

12.3 Supported Platforms

SmartSpice supports TMI flow for the following platforms:

- Linux 64 bit - TMI library path "RH_64"
- Windows - TMI library path "WIN"

12.4 ETMI Package with BSIM-CMG Model

To use this version 2012_1015 of the TMI package and the associated BSIM-CMG model release 106.1.0 you can specify `LEVEL=72` and `VERSION=106.1`. This will pick up the BSIM-CMG 106.1.0 released model and is equivalent to having `LEVEL=1061` in a non-ETMI simulation.

12.5 Additional Control Options

In addition to the mandatory TMIFLAG and TMIPATH options, the following option is recognized by SmartSpice:

```
.OPTION TMIPRINTMODEL
```

If TMIPRINTMODEL is set and equals 1 then the TMI library will be requested to print out debug information on models.

Example

```
.OPTION TMIFLAG=1 TMIPATH='/home/user/sdk/TMI' TMIPRINTMODEL=1
```

By default, this option is set to 0 and debug information is not printed.

Chapter 13
Monte Carlo and Worst-Case Analysis

13.1 Introduction

13.1.1 Monte Carlo Analysis

Monte Carlo analysis is a form of statistical analysis. When Monte Carlo analysis is performed on a circuit, the values of components and parameters of the circuit are varied within a specified statistical distribution. At each iteration, when all varied parameter values are defined, the simulator performs one or more standard analyses in the steady-state, time, or frequency domains. After each analysis, one or more circuit performance measures are calculated. At the end of n iterations, the mean, variance, and other statistics for the circuit performance measures are calculated and displayed. Monte Carlo analysis can be performed on all parameters, including model parameters, that have user-defined device tolerances.

13.1.2 .MC and .MODIF Statements

In SmartSpice, Monte Carlo analysis may be performed either with the `.MC` statement or with the `.MODIF` statement. With the `.MC` statement, Monte Carlo analysis is performed only on model parameters that have user-defined device tolerances. With the `.MODIF` statement, Monte Carlo analysis is performed both on parameters specified in the `.MODIF` statement and on model parameters with user-specified tolerances.

13.1.3 Monte Carlo in Other Analysis Statements

SmartSpice supports Monte Carlo statistical analysis through the use of `.AC`, `.DC`, `.TRAN`, `.PARAM` and `.MEASURE` statements (as part of Silvaco's continuing policy in maintaining syntax compatibility with HSPICE).

13.1.4 Worst-Case Analysis

Worst-case analysis varies one parameter per iteration and performs a standard analysis for each iteration. After each analysis is finished, SmartSpice calculates the sensitivity of the user-defined circuit performance measure with respect to the varied parameter. At the end of the worst-case analysis, a final run is performed using a combination of the varied parameter values. The result of the final run will represent the worst-case performance of the circuit with respect to the performance measure. Worst-case analysis can only be performed on those model parameters that have user-defined device tolerances.

13.2 Monte Carlo Analysis with .MODIF Statement

Monte Carlo analysis in SmartSpice can be performed with the .MODIF statement.

Syntax

```
.MODIF <param1 <+|-|*|/> = rhs1 <param2_spfc ...>>
+ <LOOP <=nreps>> <PROFF> <PRTBL> <PRMC>
+ <MODIF parset2>
```

param1, **param2**	Names of parameters. Each is one of the following: • the name of a device parameter • the name of a model parameter • the TEMP (temperature) parameter • a parameter label defined in a global .PARAM statement

It is possible to specify a parameter for all devices of the same type. The syntax for this modification is ALL@devtype<parname>. SmartSpice will modify the parameter parname of all devices of the type devtype.

+, **-**, *****, **/**	The arithmetic operators.
rhs1	Parameter value, increment, or multiplier containing distribution function specification. It is one of the following: • UNIF (nomval relvar <mult>) • AUNIF (nomval absvar <mult>) • GAUSS (nomval relvar sigma <mult>) • AGAUSS (nomval absvar sigma <mult>) • LIMIT (nomval relvar) • ALIMIT (nomval absvar) • WEIBULL (nomval relvar <csign> val rlim) • AWEIBULL (nomval absvar <csign> cval lim) Each parameter specified in the .MODIF statement has its own random generator.
AUNIF	Uniform distribution function with absolute variation.
UNIF	Uniform distribution function with relative variation.
AGAUSS	Gaussian distribution function with absolute variation.
GAUSS	Gaussian distribution function with relative variation.
ALIMIT	Random limit distribution function with absolute variation.
LIMIT	Random limit distribution function with relative variation.
AWEIBULL	Weibull distribution function with absolute variation.
WEIBULL	Weibull distribution function with relative variation.

nomval	Nominal parameter value.
absvar	The absolute variation. The AUNIF, AGAUSS and AWEIBULL distributions vary the nominal value by $\pm absvar$.
relvar	The relative variation. The UNIF, GAUSS and WEIBULL distributions vary the nominal value by $\pm nomval \times relvar$.
mult	The multiplier. Calculation of the random function will be performed mult times and will save the largest deviation. Default is 1.
sigma	The parameter for Gaussian distribution. The effective standard deviation of a random sample will be $\frac{3absvar}{sigma}$. The default is 3. Care must be taken to avoid generating parameter values outside allowed limits.
csign	Plus or minus. This sign is used to specify the orientation of the non-symmetrical Weibull distribution.
val	The parameter $c > 1$ of the standard Weibull density function: $p_x(x) = cx^{c-1}\exp(-x^c),(x > 0)$
rlim	The parameter for Weibull distribution.
param2_spfc	The specifications for param2. These specifications have the same style as the specifications for param1.
LOOP=nreps	Where nreps is the number of Monte Carlo iterations.
PROFF	Suppresses on line printing of measures specified and calculated by means of .MEASURE statements for the current set of parameters.
PRTBL	Causes SmartSpice to print the final table of all parameters and measures calculated for the current set of parameters.
PRMC	Causes SmartSpice to print mean, variance, sigma, and average deviation for each measure calculated during Monte Carlo analysis. $Mean = \frac{meas(1) + meas(2) + \ldots + meas(nreps)}{nreps}$ $Sigma = \sqrt{Variance}$ $Variance = \frac{(meas(1) - Mean)^2 + \ldots + (meas(nreps) - Mean)^2}{nreps - 1}$ $AverageDeviation = \frac{\lvert meas(1) - Mean\rvert + \ldots + \lvert meas(nreps) - Mean\rvert}{nreps}$
MODIF	This keyword, followed by a set of parameters, begins the next set of parameters. This keyword is not preceded by a period (.).

MODIF parset2	The specifications for the second set of parameters.

In any circuit there may be several devices that can use the same model. In Monte Carlo analysis, the model parameter values for each device can be altered individually. The statistical correlations of the parameter values are controlled in one of three ways:

1. A model parameter can be identical for all devices in the circuit (100% correlation).
2. Model parameters can be varied by some small amount around a nominal value (partial correlation).
3. The model parameters can be completely independent of each other (0% correlation).

On the first iteration, SmartSpice performs all analyses and calculates all measures for the nominal values specified in the sourced input deck. Subsequent iterations are performed with parameter values that are changed according to the distributions specified in the .MODIF statement and in the device tolerance specifications of the .MODEL statements (See Section .MODEL (Model Definition)).

If the keyword EACH is not specified in a .MODEL statement for a parameter, and this parameter is specified in the .MODIF statement, then the parameter value generated by the .MODIF statement will be used for all devices (100% correlation).

If the keyword EACH is specified in a .MODEL statement for a parameter, and this parameter is specified in the .MODIF statement, then the parameter value generated by the .MODIF statement will be used as a new nominal value from which individual parameter values for each device are generated (partial correlation).

If the keyword EACH is specified in a .MODEL statement for a parameter, and this parameter is not specified in the .MODIF statement, then the parameter is varied according to the distribution given in the .MODEL statement. Each device will have an independently generated parameter value (0% correlation).

The following example illustrates the relationship between the .MODIF statement and device tolerances in .MODEL definitions.

Example

```
.MODIF loop=50
+ NMOD(TOX)=GAUSS(2e-8 0.02 1)
+ NMOD(VTO)=GAUSS(1 0.1 3)
.MODEL NMOD NMOS (LEVEL=3 TOX=2e-8
+ VTO=0.7 EACH 2%
+ WMLT=1 EACH/GAUSS 5%)
M1 1 2 0 0 NMOD W=10u L=1.2u
M2 3 4 0 0 NMOD W=10u L=1.2u
```

In the input deck fragment above, the parameters for the Monte Carlo analysis have been specified in the .MODIF and .MODEL statements. On the first iteration, all analyses are performed for nominal parameters, namely TOX=2e-8, VTO=1 and WMLT=1.

On subsequent iterations from 2 to 50, SmartSpice first generates TOX and VTO parameters using GAUSS distributions specified in the .MODIF statement. The TOX and VTO values are selected from the intervals (1.88e-8, 2.12e-8) and (0.9, 1.1), respectively.

Assuming the values on the second iteration are TOX=2.1e-8 and VTO=0.95, SmartSpice then generates parameter values according to the device tolerances specified in the .MODEL statement. The nominal value VTO=0.95, the absolute variation of 0.02 * 0.95, and the default uniform distribution are used to produce two parameter values: the first one for transistor M1,

and the second one for transistor M2. Assume that the VTO value for M1 is 0.94 and M2 is 0.96 on the second iteration. For parameter WMLT, the nominal value WMLT=1 and GAUSS distribution are used to produce the parameter values for transistors M1 and M2. The values are selected from the interval (0.85, 1.15).

Assume that the value of WMLT for M1 is 0.9 and for M2 is 0.95 on the second iteration. The model parameter values on the second iteration are:

	M1	**M2**	
TOX	2.1e-8	2.1e-8	(100% correlation)
VTO	0.94	0.96	(partial correlation)
WMLT	0.90	0.95	(0% correlation)

13.3 Standard Distributions

The distributions shown in Figure 13-1 through Figure 13-5 can be specified in the `.MODIF` statement.

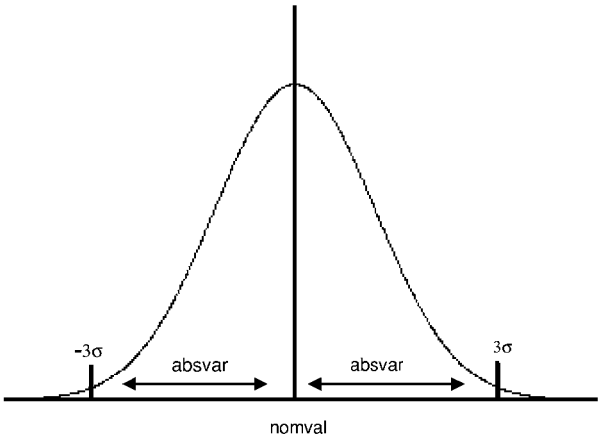


Figure 13-1 Gaussian Distribution

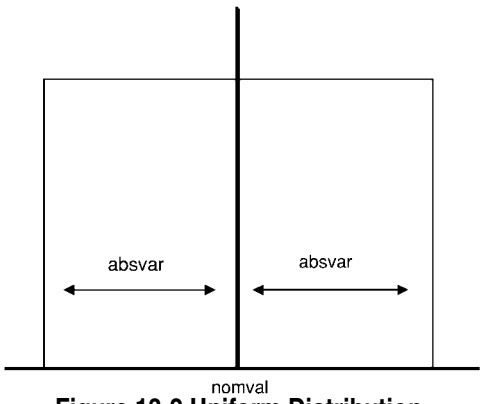


Figure 13-2 Uniform Distribution

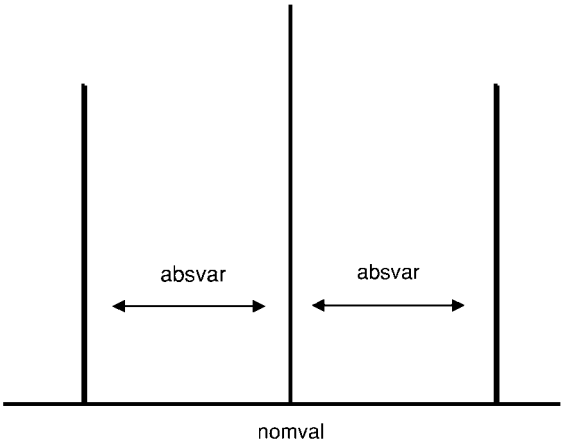


Figure 13-3 Random Limit Distribution

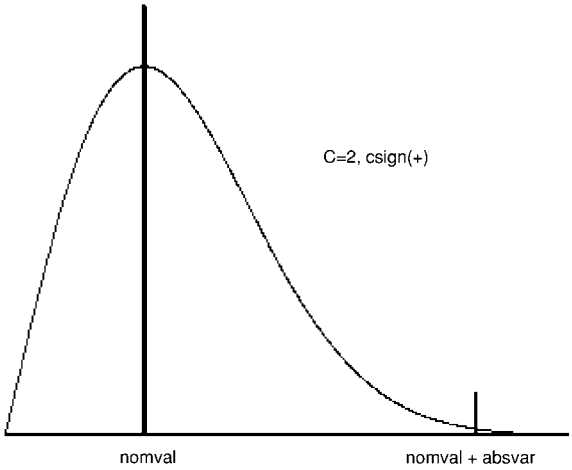


Figure 13-4 Weibull Density Functions

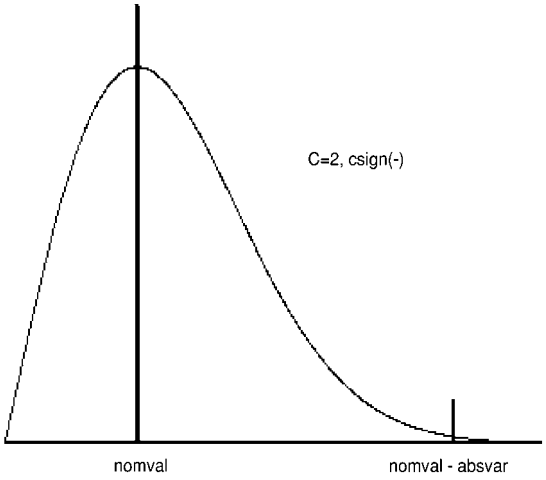


Figure 13-5 Weibull Density Functions

13.4 Monte Carlo Example with .MODIF Statement

```
*
* MONTE CARLO EXAMPLE
* INPUT FILE
*
*
VIN 1 0 DC 1 PULSE (1 0 0 1NS 1NS 19NS)
R1 1 2 500
R2 2 0 200
C1 2 0 10PF
*
.TRAN 1.NS 40NS CALLV SAVEV

.MEASURE TRAN MAX_V2  MAX V(2)
.MEASURE TRAN DEL_RISE DELAY V(1) RISE=1 VAL=0.5
+ TARG = V(2) RISE=1 VAL0=0 VAL1=MAX_V2
.MEASURE TRAN FALL_TIME WAVE V(2) FALL=1 VAL0=0
+ VAL1=MAX_V2
*
.MODIF LOOP=30 R2(RES)=AUNIF(200 50)
+          C1(CAP)=UNIF(10PF 0.5)  PRMC PROFF
+MODIF LOOP=30 R2(RES)=AGAUSS(200 50 3)
+         C1(CAP)=GAUSS(10PF 0.5 3 10)PRMC PROFF
+MODIF LOOP=30 R2(RES)=ALIMIT(200 50)
+          C1(CAP)=LIMIT(10PF 0.5) PRMC PROFF
+MODIF LOOP=30 R2(RES)=AWEIBULL (200 50, 2,2)
+          C1(CAP)=WEIBULL(10PF 0.5 -2 2)PRMC PROFF
.END
```

An illustration of the Monte Carlo example circuit is shown in Figure 13-6.

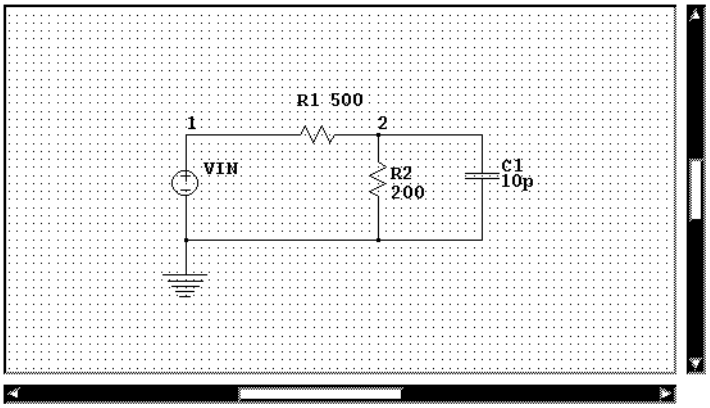


Figure 13-6 Monte Carlo Example Circuit

The input deck in this example defines a simple circuit comprised of two resistors and one capacitor. Monte Carlo analysis is performed four times using uniform, Gaussian, limit and Weibull distribution functions. In each case, the resistance R2 and capacitance C2 are varied and the statistics for the measures `max_v2`, `del_rise` and `fall_time` are calculated. The transient analysis is performed for every Monte Carlo iteration.

The parameter `LOOP=30` specifies the number of Monte Carlo iterations. The `PMRC` option causes SmartSpice to print out the results of the analysis, and the `PROFF` option suppresses the on line output of measures as they are calculated.

The numerical results of Monte Carlo analysis are shown below:

```
MONTE-CARLO STATISTICS FOR SET #1
Measurement 'max_v2':
MEAN         VARIANCE          SIGMA        AVER. DEV.
2.846e-01    6.436e-04         2.537e-02    2.059e-02
MAX. ABS. DEV. FROM NOMINAL    VALUE        STEP
             -5.359e-02        2.323e-01    11
Measurement 'del_rise':
MEAN         VARIANCE          SIGMA        AVER. DEV.
1.589e-09    2.939e-19         5.421e-10    4.629e-10
MAX. ABS. DEV. FROM NOMINAL    VALUE        STEP
             1.085e-09         2.779e-09    15
Measurement 'fall_time':
MEAN         VARIANCE          SIGMA        AVER. DEV.
3.101e-09    8.374e-19         9.151e-10    7.846e-10
MAX. ABS. DEV. FROM NOMINAL    VALUE        STEP
             1.906e-09         5.172e-09    15

MONTE-CARLO STATISTICS FOR SET #2
Measurement 'max_v2':
MEAN         VARIANCE          SIGMA        AVER. DEV.
2.891e-01    1.712e-04         1.309e-02    1.008e-02
MAX. ABS. DEV. FROM NOMINAL    VALUE        STEP
             3.266e-02         3.185e-01    24
Measurement 'del_rise':
MEAN         VARIANCE          SIGMA        AVER. DEV.
1.619e-09    3.193e-19         5.651e-10    5.348e-10
MAX. ABS. DEV. FROM NOMINAL    VALUE        STEP
             -7.955e-10        8.985e-10    5
Measurement 'fall_time':
MEAN         VARIANCE          SIGMA        AVER. DEV.
3.160e-09    9.365e-19         9.677e-10    9.157e-10
MAX. ABS. DEV. FROM NOMINAL    VALUE        STEP
             -1.310e-09        1.955e-09    5
```

```
MONTE-CARLO STATISTICS FOR SET #3
Measurement 'max_v2':
MEAN          VARIANCE          SIGMA         AVER. DEV.
2.737e-01     2.472e-03         4.972e-02     4.857e-02
MAX. ABS. DEV. FROM NOMINAL     VALUE         STEP
              -5.505e-02        2.308e-01     30
Measurement 'del_rise':
MEAN          VARIANCE          SIGMA         AVER. DEV.
1.476e-09     5.890e-19         7.674e-10     7.015e-10
MAX. ABS. DEV. FROM NOMINAL     VALUE         STEP
              1.305e-09         2.999e-09     26
Measurement 'fall_time':
MEAN          VARIANCE          SIGMA         AVER. DEV.
2.947e-09     1.672e-18         1.293e-09     1.182e-09
MAX. ABS. DEV. FROM NOMINAL     VALUE         STEP
              2.293e-09         5.559e-09     26

MONTE-CARLO STATISTICS FOR SET #4
Measurement 'max_v2':
MEAN          VARIANCE          SIGMA         AVER. DEV.
2.891e-01     3.176e-04         1.782e-02     1.370e-02
MAX. ABS. DEV. FROM NOMINAL     VALUE         STEP
              -4.573e-02        2.402e-01     11
Measurement 'del_rise':
MEAN          VARIANCE          SIGMA         AVER. DEV.
1.807e-09     1.527e-19         3.908e-10     3.193e-10
MAX. ABS. DEV. FROM NOMINAL     VALUE         STEP
              8.317e-10         2.526e-09     8
Measurement 'fall_time':
MEAN          VARIANCE          SIGMA         AVER. DEV.
3.467e-09     4.601e-19         6.783e-10     5.547e-10
MAX. ABS. DEV. FROM NOMINAL     VALUE         STEP
              1.464e-09         4.730e-09     8
```

13.5 User-Defined Device Tolerance

Monte Carlo and worst-case analysis device tolerances are defined in .MODEL definitions.

Syntax

```
pname=val <EACH|ALL> </num_gen> </dist_name>=val2<%>
```

pname	The parameter name specified in a .MODEL definition.
val	The value of the parameter.
EACH|ALL	Defines the tolerance type for a parameter. If ALL is used, all devices using this model will be given the same parameter value. If EACH is used, for each device that uses this model, the parameter value will be changed independently. ALL and EACH can both be set for one parameter.
num_gen	Index of the random generator for this parameter. There are ten generators for EACH (0-9) and ALL (0-9). A generator can be referred to in different device tolerances.
dist_name	Name of distribution. Can be one of the following: • **UNIFORM:** Generates a uniformly distributed random variable ranging from $val - val2$ to $val + val2$. • **GAUSS:** Generates a random variable with Gaussian distribution over a range of $\pm 3\sigma$. The effective standard deviation of a random sample will be $3val2$. • **user_name:** Generates random variables with user-defined distribution ranging from $-val2$ to $val2$. See Section .DISTRIBUTION (User-Defined Distribution) for more information.
val2	The absolute or relative variation. A percentage sign (%) after val2 means the absolute variation is calculated as $0.01val \times val2$.

Note: It is important to define the tolerance value carefully. The tolerance value should not exceed the nominal value. Keep in mind that in Gaussian distribution real tolerance ranges from 0 to $3val2$. Keep the parameter values within a reasonable set of values.

The keyword EACH specifies that the parameter will be independently changed for each device of the specified model. This means that the correlation between parameter values for different devices is equal to 0. For example:

```
.MODEL RMOD1 R(RES=1 each=10%)
```

In this example, for every run, the value of RES will be independently changed in the interval (0.9, 1.1) for each resistor of the model RMOD1.

The keyword ALL specifies that the model parameter will have the same value for all corresponding devices. This means that the correlation between parameter values for different devices is equal to 1. For example:

```
.MODEL RMOD1 R(RES=1 ALL=10%)
```

In this example, for every run the value of RES will be simultaneously changed in the interval (0.9, 1.1) for each resistor of the model RMOD1.

If both ALL and EACH are specified after a parameter definition, the parameter will be changed by the keyword ALL for all devices. Then, the parameter will be changed as specified by the keyword EACH for each device separately. This is an attempt to simulate the correlation between parameter values for different devices ranging from 0 to 1.0.

Example

```
.MODEL RMOD1 R(RES=1 ALL=10% EACH=5%)
```

In this example, the value of RES is simultaneously selected from the interval (0.9, 1.1) for each resistor of the model RMOD1. Assuming that the selected value of RES is val, then the value of RES is selected independently from the interval (0.95val,1.05val) for each resistor of the model RMOD1.

There are ten random generators for Monte Carlo analysis. To initialize a random generator, set a seed value. The seed value is defined in the .MC statement. Change the seed value to change all values generated by a random generator. This changes the results of Monte Carlo analysis because it changes all parameter values. The number of the random generator is defined for each parameter in the .MODEL statement. The default is 0.

Worst-case analysis also uses tolerance definitions from the .MODEL statement. The keyword EACH causes SmartSpice to calculate sensitivity for this parameter differently for each specified device. During the final run, the parameter values may vary for different devices.

Example

```
Q1 1 2 3 QMOD
Q2 4 5 6 QMOD
....
.MODEL QMOD NPN(RB=100 EACH=10%)
```

In this example, SmartSpice first calculates sensitivity with respect to @Q1[RB], then calculates sensitivity with respect to @Q2[RB], and during the final run changes @Q1[RB] and @Q2[RB] independently in accordance with calculated sensitivities.

The keyword ALL causes SmartSpice to calculate sensitivity for all devices specified in the model during one run.

Example

```
Q1 1 2 3 QMOD
Q2 4 5 6 QMOD
....
.MODEL QMOD NPN(RB=100 ALL=10%)
```

In this example, SmartSpice calculates sensitivity with respect to @QB1[RB] and @Q2[RB] by changing these values simultaneously. During the last step, @QB1[RB] and @Q2[RB] are also changed simultaneously.

13.6 Monte Carlo Analysis with the .MC Statement

All information about Monte Carlo analysis is determined in the .MC statement. The .MC statement specifies the number of runs, the type of analysis, the name of the output variable, and the way the output variable is measured (YMAX, MAX, MIN, RISE_CROSS, FALL_CROSS). The keyword RANGE(from, to) is used to set a measurement range.

Note: The output variable measurement methods for transient analysis in the .MC statement are different from those in the .MEASURE statement. The .MEASURE statement is more accurate because it uses real simulation steps to calculate values. The .MC statement uses linearized values in output steps.

The .MC statement also controls the output stream from various sources. The keyword LIST prints each changed parameter before every run in Monte Carlo analysis. The keyword OUTPUT and its parameters control the output of other output statements (.PRINT, .PLOT, .MEASURE, etc.). For more information on this feature, see Section .MC (Monte-Carlo Analysis).

The .MC statement can also produce its own output. It prints output after the last run, sorting the results of the measurements for all runs. The .MC statement calculates absolute deviation, minimum, median, maximum, 10th percentile, and 90th percentile if the number of runs is greater than 10.

SmartSpice first runs all analyses specified in the input deck for nominal parameter values. Subsequent runs are performed with changed parameter values according to the tolerance definitions in the .MODEL statement. Only analyses defined in the .MC statement are run. After the runs have been completed, statistical characteristics are calculated.

The type of distribution for each parameter is specified in the .MODEL statement. The possible distribution types are Gauss, Uniform or user-specified. For information about the user-specified functions, see Section .DISTRIBUTION (User-Defined Distribution).

In the .DISTRIBUTION statement, the probability density is defined as a piecewise function. The type of distribution is specified in the .OPTIONS statement. For more information, see Section 3.14 .OPTIONS (Option Specification).

13.6.1 Monte Carlo Example with .MC Statement

This section contains a sample circuit, input deck, and results for Monte Carlo analysis performed with the .MC statement.

The input deck below defines a simple inverter. Monte Carlo analysis is performed ten times, including the first nominal run. For each run, parameters BF, CCS and CJC are varied in accordance with tolerance definitions; statistics for RISE_CROSS (4.0V) with respect to V(3) are calculated.

```
*Example for Monte Carlo Analysis
.OP
.TRAN
.MC 10 TRAN V(3) RISE_CROSS(4.0)
VCC 6 0 5.2
VIN 7 0 PULSE(0 5 2NS 2NS 90NS)
R1 7 2 10K
R2 6 3 2K
R3 3 4 10K
R4 6 5 2K
```

```
Q1 3 2 0 QMC
Q2 5 4 0 QMC
.PRINT TRAN V(3) V(1) V(7)
.MODEL QMC NPN(BF=70 EACH=10% RB=80 RC=50
+ CCS=2.1PF ALL=20% TF=0.11NS TR=9NS CJE=0.9PF
+ CJC=1.9PF EACH=0.5PF PC=0.8 VA=50)
* Threshold value for RISE_CROSS calculation
VTR 1 0 4.0
RTR 1 0 1K
```

An example of a circuit diagram is shown in Figure 13-7.

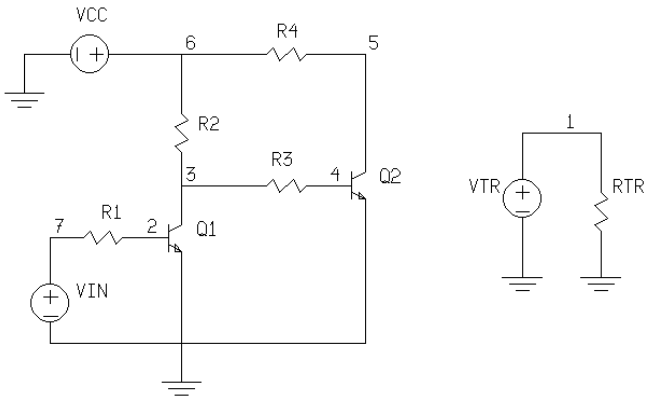


Figure 13-7 Example Circuit Diagram

The numerical results of Monte Carlo analysis, using the example input deck, are shown below:

```
                    :SORTED DEVIATIONS
                MONTE CARLO SUMMARY FOR rise(v(3))
Mean            =  1.7796e-07
Sigma           =  8.3527e-09
Mean absolute deviation=5.9855e-09
Minimum         =  1.6321e-07
10th %ile       =  1.6454e-07
Median          =  1.7774e-07
90th %ile       =  1.8728e-07
Maximum         =  1.9180e-07
run # 51.6321e-07diff.=-1.4753e-08(-1.77 sigma)
at arg = 1.6321e-07
run # 91.6587e-07diff.=-1.2096e-08(-1.45 sigma)
at arg = 1.6587e-07
run # 31.7609e-07diff.=-1.8679e-09(-0.22 sigma)
at arg = 1.7609e-07
NOMINAL run 1.7697e-07diff.=-9.8827e-10(-0.12 sigma) at arg = 1.7697e-07
run # 71.7774e-07diff.=-2.2206e-10(-0.03 sigma)
at arg = 1.7774e-07
run # 21.8053e-07diff.= 2.5643e-09(0.31 sigma)
at arg = 1.8053e-07
 run # 61.8191e-07diff.= 3.9500e-09(0.47 sigma)
 at arg = 1.8191e-07
 run # 101.8273e-07diff.= 4.7701e-09(0.57 sigma)
 at arg = 1.8273e-07
 run # 41.8277e-07diff.= 4.8097e-09(0.58 sigma)
 at arg = 1.8277e-07
```

```
run # 81.9180e-07 diff.=  1.3833e-08(1.66 sigma) at arg = 1.9180e-07
```

The graphic results of Monte Carlo analysis, using the example input deck, are shown in Figure 13-8.

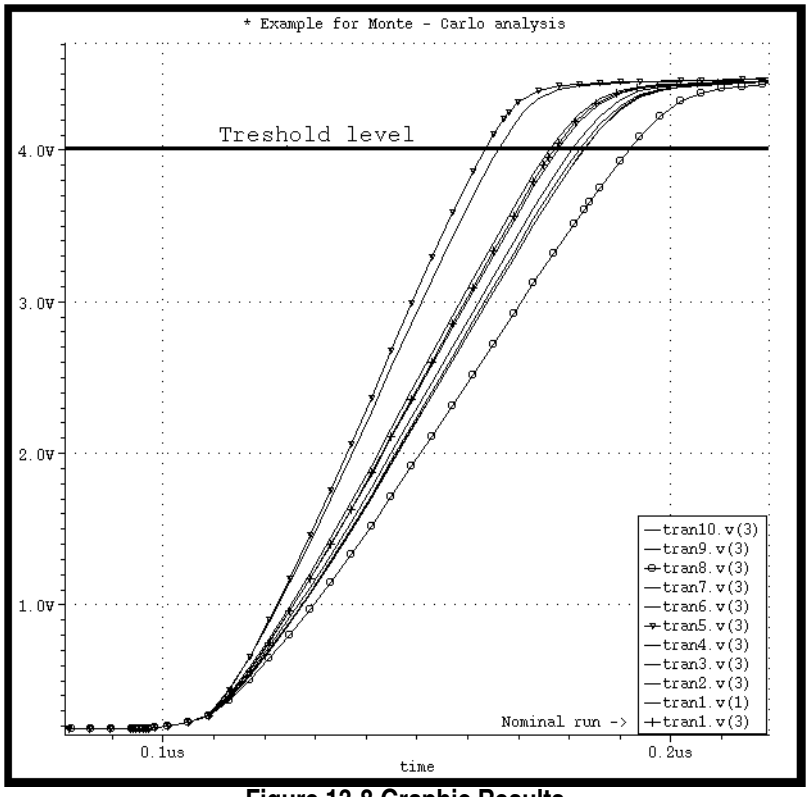


Figure 13-8 Graphic Results

13.7 Monte Carlo Analysis with .AC, .DC, .TRAN, .PARAM and .MEASURE Statements

SmartSpice supports Monte Carlo statistical analysis through the use of .AC, .DC, .TRAN, .PARAM and .MEASURE statements (as part of Silvaco's continuing policy in maintaining syntax compatibility with HSPICE).

.DC, .AC and .TRAN statements can be used for the selection of analysis type and for specification of the number of Monte Carlo repetitions to be performed.

.PARAM statements can set parameters to a Gaussian, Uniform, or Limit function distribution.

.MEASURE statements set the name and type of measurement and names of output variables (for details see Section .MEASURE (Analysis Measurements)).

```
 Analysis Syntax
.DC vin 0 1 .1 <SWEEP> MONTE=val | list(numl) | val firstrun=numf |
list( [num0] [num1:num2] [num3] ... )  <PRMC<=val>>

.AC  dec  10  1k  10meg  <SWEEP>  MONTE=val  |  list(numl)  |  val
firstrun=numf | list( [num0] [num1:num2] [num3] ... )  <PRMC<=val>>

.TRAN .1n 50n <SWEEP> MONTE=val | list(numl) | val firstrun=numf |
list( [num0] [num1:num2] [num3] ... )  <PRMC<=val>>
```

val	The number of Monte Carlo repetitions.
list	Defines the sequence of iterations. Can write only one number after list (numl). For example, in [num1:num2] the colon represents "from ... to ...". Specifying only one number makes SmartSpice run only at that single point. The numbers after list can not be the parameter.
firstrun=numf	Specifies the desired number of iterations. SmartSpice runs from numf to numf+val-1. numf can be a parameter.
PRMC<=val>	Causes SmartSpice to print the values of modified and measured parameters on each Monte Carlo iteration. In HSPICE mode, the flag PRMC saves distributed parameters in the measure file, if the .MEASURE statement is used. The PRMC=2 saves all parameters in the raw file, if the option sweepmonte is present. The feature doesn't work in HSPICE compatible mode.

Example 1

SmartSpice runs from the 10th to 19th Monte Carlo iterations:

```
.tran 1n 10 sweep monte=10 firstrun=10
.DC VIN 1 10 1   sweep monte=10 firstrun=10
.AC dec 5 100 1khz  sweep monte=10 firstrun=10
```

Example 2

SmartSpice begins running at the 5th iteration, then continues from the 10th to the 15th, at the 20th, and finally from the 25th to 30th Monte Carlo iteration.

```
.tran 1n 10n sweep monte=list(5 10:15 20 25:30)
```

```
.DC VIN 1 10 1   sweep monte=list(5 10:15 20 25:30)
.AC dec 5 100 1khz  sweep monte=list(5 10:15 20 25:30)
```

In HSpice mode, SmartSpice recalculates a distributed parameter each time this parameter is used.

.PARAM Distribution Function Syntax

```
.PARAM  parname=UNIF(nomval, relvar <,mult>)
.PARAM  parname=AUNIF(nomval, absvar <,mult>)
.PARAM  parname=GAUSS(nomval, relvar, sigma <,mult>)
.PARAM  parname=AGAUSS(nomval, absvar, sigma <,mult>)
.PARAM  parname=LIMIT(nomval, absvar)
```

parname	The parameter whose value is calculated by distribution function.
UNIF	Uniform distribution function with relative variation.
AUNIF	Uniform distribution function with absolute variation.
GAUSS	Gaussian distribution function with relative variation.
AGAUSS	Gaussian distribution function with absolute variation.
LIMIT	Random limit distribution function with absolute variation, +/- absvar is added to nomval based on whether the random outcome of a -1 to 1 distribution is greater or less then 0.
nomval	Nominal value for Monte Carlo analysis.
absvar	The absolute variation, the AUINF, AGAUSS vary nomval by +/- absvar.
relvar	The relative variation, the UNIF and GAUSS vary nomval by +/- (nomval x relvar).
sigma	The parameter for Gaussian distribution. The effective standard deviation of a random sample will be absvar/sigma.
mult	The multiplier repeats function calculation mult times, saving only the largest deviation. Default value for mult is 1.

Syntax Example

```
*MONTE CARLO PARAMETERS
.param nvt0 = 10e-3      $ V*um
.param pvt0 = 15e-3      $ V*um
.param nbeta = 2.3e-2     $ 100%*um
.param pbeta = 3.2e-2     $ 100%*um
.param delta_l = 0.0      $ m
.param sigma = 1.0 multiplier = 1

* Subcircuit Definitions
.subckt n    d g s b chw=0 chl=0 mult=1 g=0
.param   dvt0   =   agauss(0.0,   'nvt0/sqrt(mult*chw*chl*1e12)',
sigma,multiplier)
.param   wdw   =   gauss('chw',   'nbeta/sqrt(mult*chw*chl*1e12)',
```

```
sigma,multiplier)
.param ldl = agauss('chl', delta_l, sigma, multiplier)
mn d g s b    n w=wdw l=ldl m=mult geo=g delvto=dvt0
.ends n

.subckt nl    d g s b chw=0 chl=0 mult=1 g=0
.param dvt0 = agauss(0.0, 'nvt0/sqrt(mult*chw*chl*1e12)', sigma )
.param wdw  =  gauss('chw','nbeta/sqrt(mult*chw*chl*1e12)',sigma)
.param ldl  = agauss('chl', delta_l, sigma)
 mnl d g s b      nl w=wdw l=ldl  m=mult geo=g delvto=dvt0
.ends nl

.subckt p    d g s b chw=0 chl=0 mult=1 g=0
.param dvt0 = agauss(0.0, 'pvt0/sqrt(mult*chw*chl*1e12)', 'sigma')
.param   wdw      =   gauss('chw','pbeta/sqrt(mult*chw*chl*1e12)',
'sigma')
.param ldl = agauss('chl', delta_l,'sigma')
mp d g s b    p w=wdw l=ldl m=mult geo=g delvto=dvt0
.ends p

*Analysis
.TRAN 0.01n 150n SWEEP MONTE=30 PRMC

*MEASURE PERIOD AND DUTY CYCLE
.MEASURE  TRAN  period  TRIG  v(out)  VAL=0.75  RISE=9  TARG  v(out)
VAL=0.75 RISE=10
.MEASURE TRAN high TRIG v(out) VAL=0.75 RISE=9 TARG v(out) VAL=0.75
FALL=9
.MEASURE TRAN low TRIG v(out) VAL=0.75 FALL=9 TARG v(out) VAL=0.75
RISE=10
.
.END
```

Result Format Example

```
 Monte Carlo (Steps = 30) : statistical descriptors
====================================
 Name of measurement: period
 Type of measurement: trig
 Type of analysis   : TRAN
                Mean: 1.00004e-08
              Median: 1.000000-08
            Variance: 6.57963e-23
               Sigma: 8.11149e-12
   Average Deviation: 6.13011e-12
             Maximum: 1.00171e-08
             Minimum: 9.97861e-09
----------------------------------------
 Name of measurement: high
 Type of measurement: trig
 Type of analysis   : TRAN
                Mean: 4.80545e-09
            Variance: 4.45355e-23
               Sigma: 6.67349e-12
   Average Deviation: 5.3254e-12
             Maximum: 4.81927e-09
             Minimum: 4.79334e-09
----------------------------------------
 Name of measurement: low
 Type of measurement: trig
 Type of analysis   : TRAN
                Mean: 5.19498e-09
            Variance: 3.90009e-23
               Sigma: 6.24507e-12
   Average Deviation: 5.04407e-12
             Maximum: 5.20655e-09
             Minimum: 5.18205e-09
----------------------------------------
```

HSpice Compatible Monte Carlo Analysis Distributed Parameters Modification

An improvement has been made to distributed parameter modification under Monte Carlo analysis. Now distributed parameters get reevaluated at each access only if it is accessed directly. In case of indirect access (i.e., through other parameters), it retains the previously calculated value.

Example

```
.param r_local=agauss(0,1,1) ; distributed param set for local
variations
.param r_random=agauss(0,1,1) ; distributed param set for global
variations
.param r_global=r_random ; r_random calculated and assigned into
r_global
r1 n1 n2 r_global ;global variation -r_random value will not be
recalculated
r2 n1 n2 r_local ;local variation -r_local value is recalculated on
this access
r3 n1 n2 r_local ;local variation -r_local value is recalculated on
this access
```

Output

Modified parameter information which is printed if `'prmc'` flag is set has also been improved. Now it lists distributed parameters, their info and final value and all dependent parameters. Dependent parameters have their value listed and also seed value (distributed parameter value) used in the calculation. If parameter is dependent on several distributed parameters, it will be listed under all of them as dependent.

Below is the output format:

```
distr_param         = value [distribution info]
dependent_param1  =  value  (distr_param  value  used  to  calculate
dependent_param1)
dependent_param2  =  value  (distr_param  value  used  to  calculate
dependent_param2) ...
```

Example

Parameter modification:

```
r_random = -0.32287 [agauss(0, 1, 1, 1)]
r_global = -0.32287 (-0.32287)
r_local  = 1.33087 [agauss(0, 1, 1, 1)]
@r3[res] = 0.105705 (0.105705)
@r4[res] = 1.33087 (1.33087)
```

13.8 Advanced Sampling for Monte Carlo Analysis with .AC, .DC, .TRAN, .PARAM and .MEASURE Statements

SmartSpice supports Monte Carlo statistical analysis with an advanced sampling algorithm like LHS (Latin Hypercube Sampling).

To invoke the advanced sampling method, the following global option should be used:

```
.option sampling_method=advanced_sampling
```

where `advanced_sampling` could be SRS or LHS.

If the option `sampling_method` is not specified in the netlist or option `sampling_method` is set to SRS, SmartSpice will use the traditional Monte Carlo flow with traditional simple sampling.

If option `sampling_method` is set to LHS, SmartSpice will use the Monte Carlo analysis with advanced LHS sampling.

LHS (Latin Hypercube Sampling)

LHS is a statistical method for generating a distribution of plausible collections of parameter values from a multidimensional distribution. A square grid containing possible samples is a Latin square, if there is only one sample in each row and each column of a 2D-dimensional distribution. A Latin Hypercube is the generalization of this method to the multidimensional case. The technique was first described by McKay in 1979[1]. The SmartSpice implementation is based on [2]. LHS generates more efficient estimates of desired parameters than SRS.

Syntax

```
.option sampling_method=LHS
```

If this option is set, and it is allowed by the simulator (see Section Known limitations for LHS (Latin Hypercube Sampling) below), SmartSpice will print out the following message during the simulation:

```
Sampling method LHS is used.
```

Note 1: Beyond the sampling method itself, LHS will force SmartSpice to run an additional nominal run

Note 2: LHS could be used with .option replicates.

Syntax

```
.option replicates=val
```

where `val` is the number of replicates. For example, if `replicates` is equal to 3 and `monte`=2, SmartSpice will do 1(nominal run) +3*2 runs.

Example

SmartSpice runs 10 MC steps with advanced sampling LHS:

```
.option sampling_method=LHS
    .dc vin 0 4.0 0.2 sweep monte=10 prmc
```

Output

```
Sampling method LHS is used.

MONTE CARLO STEP #1
                                        *
```

```
                DC Analysis, 27 deg C,Tue Sep 18 09:42:16 2012

par1          =  1.0629e+000
par2          =  1.0629e+000

    . . .

  Monte Carlo (Steps = 11) : statistical descriptors
=======================================================
-----------------------------------------
 Name of measurement: par1
 Type of measurement: find
 Type of analysis   : DCAN
                Mean: 1.06299
            Variance: 8.90049e-006
               Sigma: 0.00298337
   Average Deviation: 0.00190965
             Maximum: 1.06681
             Minimum: 1.05953
```

Known limitations for LHS (Latin Hypercube Sampling)

1. LHS sampling is supported only when Monte Carlo analysis is used with `.AC`, `.DC` and `.TRAN`, `.PARAM` and `.MEASURE` statements as described in Section 13.7 Monte Carlo Analysis with .AC, .DC, .TRAN, .PARAM and .MEASURE Statements. It is not supported in `.MC` analysis or the `.MODIF` statement.
2. LHS sampling is supported with the following distributions: GAUSS, AGAUSS, UNIF, AUNIF.
3. Options firstrun and list are not supported for LHS.

References

1. McKay, M.D.; Beckman, R.J.; Conover, W.J. (May 1979). A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code.
2. Stein, M. (1987). Large sample properties of simulations using Latin hypercube sampling. Technometrics 29, 2, 143-51.

13.9 Worst-case and Sensitivity

Worst-case analysis with the .WCASE statement is similar to Monte Carlo analysis with the .MC statement. With the .WCASE statement in the input deck, SmartSpice first performs a nominal run. On each subsequent run, SmartSpice calculates sensitivity with respect to each parameter that has a device tolerance definition in the .MODEL statement. The number of runs is a function of the number of parameters and devices that reference these parameters. SmartSpice then performs a final run for worst-case calculation. Parameters change in accordance with device tolerances to increase (or decrease) the measurement value defined in the .WCASE statement using parameters Hi or Low. An example circuit for worst-case analysis is shown in Figure 13-9.

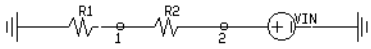


Figure 13-9 Example Circuit for Worst-case Analysis

The following example will minimize MAX(V(1)), changing R1 and R2 independently according to the device tolerance set in the .MODEL statement.

```
**EXAMPLE FOR WORST-CASE **
*-
* In this example the following are
* calculated by hand:
* d(V(1))/d(R2) = -(R1 vin)/(R1 + R2)^2)
* d(V(1))/d(R1) = (R2 vin)/(R1 + R2)^2)
*-
R1 0 1 RMOD1
R2 1 2 RMOD2
vin 2 0 DC 1
.MODEL RMOD1 R(RES=1 EACH=10%)
.MODEL RMOD2 R(RES=2 EACH=10%)
.DC vin 0.1 1 0.1
.PRINT DC V(1)
.WCASE DC V(1) MAX LIST LOW
.OPTIONS RELTOL=0.001
.END
```

An illustration representing worst-case analysis is shown in Figure 13-10.

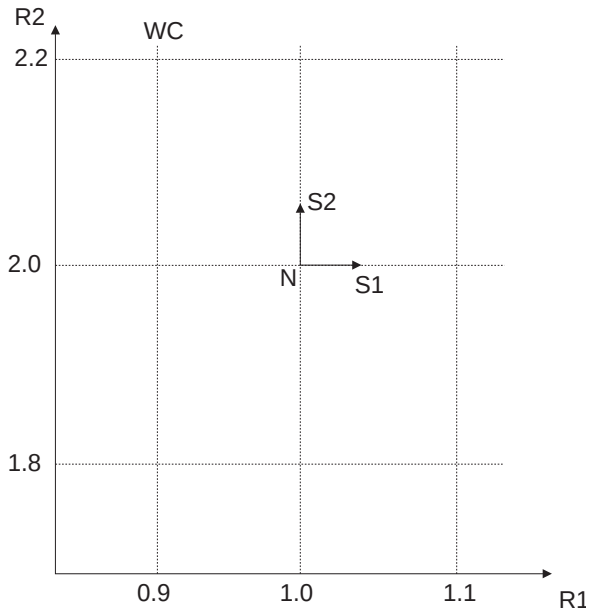


Figure 13-10 Graphical Representation of Worst-case Analysis

SmartSpice first calculates:

$$f(N) = MAX(V(1))$$

for the nominal run. The point N in Figure 13-10 represents the nominal run. Next SmartSpice runs step 1 to calculate the sensitivity with respect to R1.

The resulting value is:

$$R1 + R1 \cdot (1 + RELTOL)$$

The point S1 in Figure 13-10 represents step 1. Next SmartSpice calculates:

$$F(S1) = MAX(V(1))$$

The sensitivity for step 1 is:

$$\frac{(f(S1) - f(N))}{(R1.RELTOL)}$$

Next, SmartSpice runs step 2 to calculate the sensitivity with respect to R2. Using the sensitivities determined in this process, the worst-case direction is calculated, and the last step is run in this direction. For the final step, SmartSpice uses the device tolerance specified in the `.MODEL` statement. The point WC in the illustration on the previous page represents the final step.

Worst-case analysis produces information about each step, a sensitivity summary for each step, information about the final step, and the value of the measured function at the worst-case point. SmartSpice also produces a linearized prediction of the worst-case point. The linearized approximation at point N is used to calculate the linearized prediction of the worst-case point.

The worst-case analysis results are shown below.

```
Worst-case: Nominal Run
Sensitivity: Changed parameter values for MAX(V(1))
RMOD1 RES R1 1.0000e+00 ->> 1.0010e+00
RMOD2 RES R2 2.0000e+00 ->> 2.0020e+00
```

```
Sensitivity Summary for MAX(V(1))
# 2 RMOD2 RES R2 3.3311e-01 at arg=1.0000e+00
d(MAX(V(1)))/d(RES) = -1.1104e-01
(-0.666223% change per 1% in Model Parameter)
NOMINAL RUN 3.3333e-01 at arg=1.0000e+00
# 1 RMOD1 RES R1 3.3356e-01 at arg=1.0000e+00
d(MAX(V(1)))/d(RES) = 2.2215e-01
(0.666445% change per 1% in Model Parameter)

Worst-case: Parameter Values for MAX(V(1))
RMOD1 RES R1 1.0000e+00 ->> 9.0000e-01 (-)
RMOD2 RES R2 2.0000e+00 ->> 2.2000e+00 (+)

Worst-case Summary for MAX(V(1))
Nominal Run  3.3333e-01 (100.000), at arg=1.0000e+00
Worst-case Run2.9032e-01 (87.0986%) at arg=1.0000e+00
Linearized Prediction  2.8891e-01 (86.6733%)
```

Chapter 14
Pole-Zero Analysis

14.1 Introduction

Pole-zero analysis is useful for analyzing linear circuits. Generally, such circuits are defined by a few key parameters such as period of oscillation and Q factor. These parameters can be obtained using the standard SmartSpice transient and AC analyses, but the pole-zero analysis provides a short cut to the same information without performing a full circuit simulation. This chapter explains pole-zero analysis, demonstrates how pole-zero analysis is equivalent to performing full AC and transient analyses, and describes how to use the results generated by the pole-zero analysis.

If a circuit is linear and time invariant, it can be considered a network. A transfer function for the network can be generated. The network transfer function is a real rational function of the form:

$$F(s) = \frac{N(s)}{D(s)} = \frac{a_m s^m + a_{m-1} s^{m-1} + \ldots + a_1 s + a_o}{b_n s^n + b_{n-1} s^{n-1} + \ldots + b_1 s + b_o}$$

where a_i and b_j coefficients are real numbers, and s is the complex variable (complex frequency).[1] If the polynomials *N(s)* and *D(s)* are written in factorized form, *F(s)* becomes:

$$F(s) = \frac{a_m (s - z_1)(s - z_2)\ldots(s - z_m)}{b_n (s - p_1)(s - p_2)\ldots(s - p_n)} = \frac{a_m N_1(s)}{b_n D_1(s)}$$

The values z_1, z_2, .. z_m of s that make the numerator *N(s)* zero are called the zeros, and the values p_1, p_2, ... p_n of s that make the denominator *D(s)* zero are called the poles of the network transfer function.

The degrees m and n of the polynomials *N(s)* and *D(s)* depend on the elements and on the topology of the circuit. The sum of the number of capacitors and inductors of the circuit may be used to estimate the maximum number of possible poles and zeros.[2]

The poles of the network transfer function are the natural frequencies of the circuit and define the dynamic performance.

14.2 .PZ Statement

Syntax

```
.PZ <node1 node2 node3 node4 CUR|VOL POL|ZER|PZ> | <output input>
```

The .PZ statement causes a pole-zero analysis to be performed on the circuit. SmartSpice first creates a linearized small-signal model at the operating point of the circuit and then computes the poles and zeros of the transfer function.

node1, **node2**	Input nodes
node3, **node4**	Output nodes
CUR	Calculate the (output voltage)/(input current) transfer function.
VOL	Calculate the (output voltage)/(input voltage) transfer function.
POL	Calculates poles
ZER	Calculates zeros

PZ	Calculates poles and zeros
output	Node voltage `v(n)` or `v(n1, n2)`.
input	Name of any independent voltage or current source.

Examples

```
.PZ V(3) VIN
.PZ V(11) IN
```

In the first example, SmartSpice computes poles and zeros for (output voltage)/(input voltage) transfer function.

In the second example, SmartSpice computes poles and zeros for (output voltage)/(input current) transfer function.

When the `.PZ` statement is specified in an input deck, SmartSpice calculates the DC operating point, linearizes the circuit around the operating point, and then performs the pole-zero analysis.

14.3 Pole-Zero Computation

A four-terminal (two-port) network is shown in Figure 14-1.

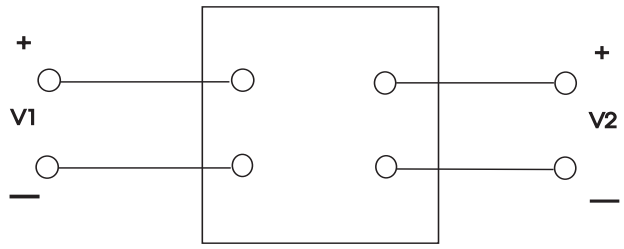


Figure 14-1 Four-Terminal (Two-Port) Network

The behavior for this network can be described by a matrix equation of the form:

$$YX = W$$

where the solution vector *X* is composed of voltages and currents. The entries y_{ij} of the matrix *Y* depend on the variable *s*:

$$y_{ij} = g + cs + \frac{1}{ls}$$

where *g*, *c*, and *l* are real numbers. The transfer function $F(s) = V_2/V_1$ for the network shown above can be written in the form:

$$F(s) = \frac{detY_N(s)}{detY(s)}$$

where the determinants $detYN(s)$ and det *Y(s)* are real rational functions of the variables.

SmartSpice performs pole and zero analysis by finding the roots of the numerator det *YN(s)* and denominator det *Y(s)*. It solves these generalized eigenvalue problems by means of the Muller method of parabolic interpolation.[3] This iteration method creates a sequence of complex numbers that converges to a root of the polynomial.

Assuming that S_k, S_{k+1} and S_{k+2} are the last three complex numbers of the iteration process, and that det $Y(S_k)$, det $Y(S_{k+1})$, and det $Y(S_{k+2})$ are corresponding values of the determinant det Y, then the Muller method is defined by the following algorithm.

1. Calculate a quadratic equation that fits through the three points (S_k, det $Y(S_k)$), (S_{k+1}, det $Y(S_{k+1})$), and (S_{k+2}, det $Y(S_{k+2})$)
2. Calculate two roots (S and SS) of the quadratic equation
3. Calculate det $Y(S)$
4. If $|\det Y(S)| < E_1$ or $|S - S_{k+2}| < E_2$, then S is a root of the polynomial det Y, else $S_{k+3} = S$, and repeat from step 1

14.4 Pole-Zero Computation for Large Circuits

The convergency rate for pole-zero calculation iteration process, when the size of circuit n>30, has become slower; and pole-zero calculations for large circuits need to have more gross accuracy, and a higher number of iterations. On the basis of our experience, we recommend the following values for accuracy (`.OPTION PZRELTOL=val`) and iteration limit (`.OPTION PZITLIM=val`):

Circuit size	PZRELTOL	PZITLIM	Number of roots
30	1.e-5 - 1.e-6	10-30	9-17
6000	1.e-5 - 1.e-6	100-150	18-39
50000	1.e-4 - 1.e-5	200-300	56-118
150000	1.e-2 - 1.e-3	300-500	150-250

Low frequency poles and zeros are well isolated, and can be found with high precision (for example, for circuit with 143000 elements, `PZRELTOL` can be set to 1.e-4). For large circuits, high frequency poles and zeros are poorly isolated and joined into clusters. In this case, pole and zero calculation requires more gross accuracy, and we will find only several poles and zeros for each cluster.

14.5 Example 1: Pole-Zero, Transient, and AC Analyses

This example illustrates the relationship between the results of the transient and AC analysis of a circuit and the results of pole zero analysis for the same circuit.

The following input file describes the linear circuit shown below for pole zero, transient, and AC analyses. Three sets of values for the capacitors and inductors of the circuit will be used. An example of the linear circuit is shown in Figure 14-2.

```
* LINEAR CIRCUIT
* POLE-ZERO, TRANSIENT AND AC ANALYSES
*
VIN 10 0 DC 1 AC 1
R1 10 1 50
R2 1 2 0.5
C1 1 0 0.2NF
C2 2 0 5NF
L1 1 0 50NH
.MODIF
+ MODIF C2(CAP)=20NF
+ MODIF C2(CAP)*=0.5 L1(IND)*=0.5 C1(CAP)*=0.5
.PZ 10 0 2 0 VOL PZ
.TRAN 1N 0.5US 0 2NS UIC
.AC DEC 10 1KHZ 10GHZ
.END
```

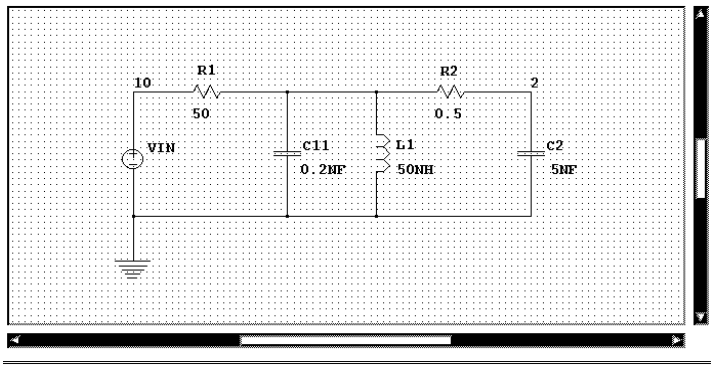


Figure 14-2 Linear Circuit

SmartSpice performs three `.MODIF` iterations for the following sets of parameters:

1. `C2=5NF L1=50NH C1=0.2NF`
2. `C2=20NF L1=50NH C1=0.2NF`
3. `C2=10NF L1=25NH C1=0.1NF`

For each set of parameters, SmartSpice performs pole zero, transient, and AC analyses. The results of the circuit simulations are shown below:

```
Pole Zero Results
.c2=5nF, l1=50nH, c1=0.2nF
pole(1) = -1.04870e+10,0.000000e+00
pole(2) = -6.49307e+06,-6.14173e+07
pole(3) = -6.49307e+06,6.141727e+07
zero(1) = 0.000000e+00,0.000000e+00

.c2=20nF, l1=50nH, c1=0.2nF
pole(1) = -1.01893e+10,0.000000e+00
pole(2) = -5.34966e+06,-3.08675e+07
pole(3) = -5.34966e+06,3.086750e+07
zero(1) = 0.000000e+00,0.000000e+00

.c2=10nF, l1=25nH, c1=0.1nF
pole(1) = -2.03786e+10,0.000000e+00
pole(2) = -1.06993e+07,-6.17350e+07
pole(3) = -1.06993e+07,6.173501e+07
zero(1) = 0.000000e+00,0.000000e+00
```

The transient and AC analysis outputs are shown in Figure 14-3 and Figure 14-4.

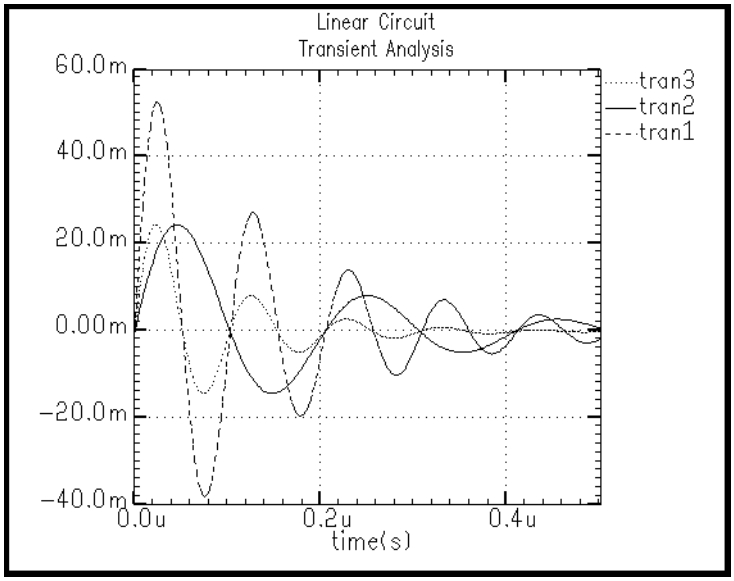


Figure 14-3 Transient Analysis Output

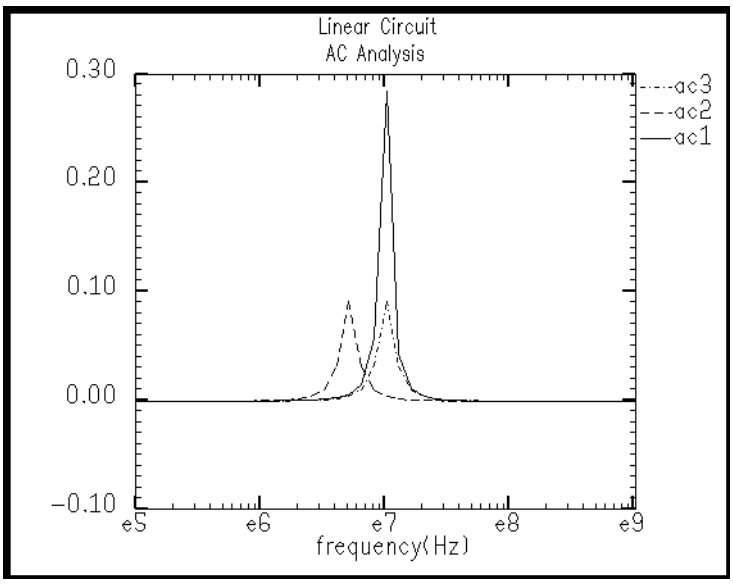


Figure 14-4 AC Analysis Output

The result of the pole-zero analysis is a set of complex numbers for the three simulations performed. There are three poles and one zero.

Each of the poles will satisfy the denominator of the network transfer function: *F(s)=V(2)/V(10)*.

$$F(s) = \frac{0.004Ls}{C1.C2.L1.s^3 + 2.L1(C1 + 1.01C2)s^2 + (0.04L1 + C2)s + 2}$$

This equation is derived directly from the circuit shown on Figure 14-1. The values for R1 and R2 have been substituted for clarity**.**

Time Domain Calculations

The poles and zeros of a network can be plotted as points in the s plane. For a circuit to be stable, its poles must be on the left part the s plane (the real part of the complex pole must be negative), and any poles that have non-zero imaginary parts must form conjugate pairs. For a linear, time-invariant system, poles will always form conjugate pairs.

Since $s = \sigma \pm j\omega$ and $\omega = 2\pi f$, the period of oscillation and decay time constant can be calculated for each of the circuits.

Examples

c2=5nF,l1=50nH,c1=0.2nF

ω	= 6.14173e7
f	= 9.77MHz
Period of oscillation	= 0.1
Decay time constant	= 6 σ t
	= $e^{-6.49307e6t}$

c2=20nF,l1=50nH,c1=0.2nF

ω	= 3.08675e7
f	= 9.83MHz

Period of oscillation	=	0.2ms
Decay time constant	=	$e^{-5.34966e6t}$

c2=10nF,l1=25nH,c1=0.1nF

ω	=	6.173501e7
f	=	4.91MHz
Period of oscillation	=	0.1ms
Decay time constant	=	$e^{-1.06993e7t}$

In each example, there is only one fundamental period of oscillation. The remaining pole has a zero imaginary component and only causes a high frequency roll off in each circuit.

The period and decay time values calculated above can be verified against the previous graph.

Frequency Domain Calculations

The poles can also be used to determine the frequency at which the output of the circuit is at its maximum and to determine the corresponding Q factor:

$$\omega_o^2 = (Re(P_j))^2 + (Im(P_j))^2$$

$$Q = \frac{\omega_o}{2Re(P_j)}$$

and

$$\omega = 2\pi f$$

```
c2=5nF, l1=50nH, c1=0.2nF
f1=9.83MHz, Q1=4.76
f2=1.67GHz, Q2=0.5

c2=20nF, l1=50nH, c1=0.2nF
f1=4.99MHz, Q1=2.93
f2=1.62GHz, Q2=0.5

c2=10nF, l1=25nH, c1=0.1nF
f1=9.97MHz, Q1=2.93
f2=3.24GHz, Q2=0.5
```

The frequencies of circuit 1 and circuit 3 are smaller, but the Q factor for circuit 1 is much greater than that of circuit 3.

The Q factor of circuit 2 is the same as that of circuit 3, but resonance occurs at a lower frequency.

The f2 frequencies have a corresponding Q factor of 0.5. This indicates that the circuits have a high frequency roll off.

These results can be compared with the results in the AC Analysis graph in Figure 14-3 and Figure 14-4.

14.6 Example 2: Fourth-Order High-Pass Filter

The following input file describes a fourth order high-pass filter. [4]

```
* FOURTH-ORDER HIGH-PASS FILTER
* POLE ANALYSIS
*
VIN 10 0 DC 1 AC 1
C1 10 1 1
R1 1 3 0.3827
C2 1 2 1
R3 2 0 0.3827
E1 3 0 2 0 1
C3 3 4 1
R2 4 6 0.9239
C4 4 5 1
R4 5 0 0.9239
E2 6 0 5 0 1
.MODIF
+ MODIF R3(RES)=2.61301 R4(RES)=1.08237
.PZ 10 0 6 0 VOL POL
.END
```

An example of fourth order high-pass filter is shown in Figure 14-5.

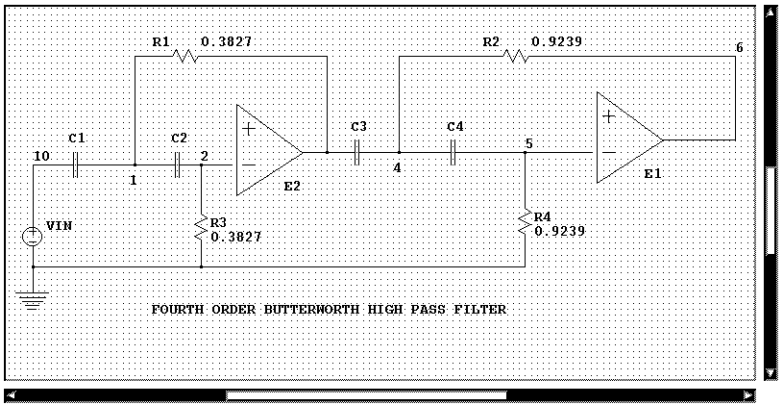


Figure 14-5 Fourth-Order High-Pass Filter

SmartSpice performs two `.MODIF` iterations for the following sets of parameters:

```
R3 = 0.3827 R4 = 0.9239
R3 = 2.61301 R4 = 1.08237
```

The results of the pole analysis are shown below:

```
pole(1) = -2.61301e+00,0.000000e+00
pole(2) = -2.61301e+00,0.000000e+00
pole(3) = -1.08237e+00,0.000000e+00
pole(4) = -1.08237e+00,0.000000e+00

r3(res)=2.61301 r4(res)=1.08237

pole(1) = -3.82700e-01,9.238731e-01
pole(2) = -3.82700e-01,-9.23873e-01
pole(3) = -9.23898e-01,3.826355e-01
pole(4) = -9.23898e-01,-3.82636e-01
```

References

1. Budak, Aram, Passive and Active Network Analysis and Synthesis, Houghton Mifflin Company: Boston, 1979, pp. 703.
2. Vlach, Jiri and Singhal Kishore, Computer Methods for Circuit Analysis and Design, Van Nostrand Reinhold, 1983, pp. 594.
3. Wilkinson, J. H., The Algebraic Eigenvalue Problem, Clarendon Press, Oxford, 1965.
4. Budak, Aram, Passive and Active Network Analysis and Synthesis, Houghton Mifflin Company: Boston, 1974, pp. 569.

Chapter 15
Transient Noise Analysis

The accurate prediction of the noise performance, especially on analog sub-blocks in mixed-signal system design, is crucial to ensure that the overall system will work correctly. SmartSpice now offers a novel time-domain, non-Monte Carlo method for simulation of the electrical noise in nonlinear dynamic circuits. Arbitrary circuit excitations and arbitrary circuit large-signal waveforms can be used, and noise variances and covariances of circuit variables as function of time will be simulated.

This time-domain noise simulation method[1] is based on results from the theory of stochastic differential equations. It has no restrictions on the circuits with time-invariant or periodic large-signal steady-state waveforms with wide-sense stationary, or cyclo-stationary noise at the output. This method can calculate the complete auto-correlation for the non-stationary noise, and can be used for phase noise characterization of open-loop oscillators. There are no pseudorandom number generators involved in the simulation. Noise simulation results are represented as the noise correlation matrix of circuit variables as a function of time.

Time-domain noise simulation is performed along with the transient simulation in the time interval specified by the user. At each time point t_n of the noise simulation, after the transient routines have calculated the solution $x(t_n)$, the Lyapunov matrix equation $MK + KM^T = N$, where K is $(m \times m)$ correlation matrix, is solved. Here, m is (roughly) the number of nodes in a circuit which are connected to a capacitor. The noise simulation algorithm (with the Bartels-Stewart algorithm used to solve the Lyapunov matrix equation) requires $O(m^3)$ flops at every time point compared with the roughly $O(m^{1.5})$ flops required by the transient analysis algorithm. Hence, the CPU time usage of the noise simulation will be largely dominated by the noise analysis routines.

The computational cost of noise simulation would be high for “large” circuits, but this noise simulation method is intended for evaluating the noise performance of small sub-blocks (containing 20 - 100 elements).

15.1 Transient Noise Analysis

Note: Transient Noise Analysis can be used to produce the noise variances of the circuit variables as a function of time, provided that Noise Sources of thermal, shot, and flicker noise models are available ONLY from devices in the circuit.

Syntax

To perform Transient Noise Analysis in SmartSpice, the `.TRAN` statement must be used.

```
.TRAN  tstep  tstop  <tstart>  <tmax>  <UIC>
+ <CALLV>  <SAVEV < =tsave>> <TRANOP < =top>>  <STORE=num>
+ <sw2_spcf>
+ NOISE<=val>  < FREQ=val  <FREQH=val>>
+ <NOISE_RELTOL=val> <NOISE_IC=ZERO|AUTO>
```

1. Alper Demir, Edward W. Y. Liu, Alberto L. Sangiovanni-Vincentelli, “Time-Domain Non-Monte Carlo Noise Simulation for Nonlinear Dynamic Circuits with Arbitrary Excitations”, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. CAD-15, NO. 5, May 1996, pp.493-505.

NOISE<=val>	Causes SmartSpice to perform Transient Noise Analysis with EXISTING circuit noise sources depending on `val` to account for:
val=1, **2**	Thermal noise only. This type of noise is produced by the resistors; by passive and active devices which have equivalent circuits that consist of resistors and conductances; and by the channel of unipolar devices.
val=3	(default) Thermal, shot and flicker (1/f) noise.
val=4	Shot noise only. This type of noise is produced mostly by the pn-junction currents of the diodes and bipolar devices.
val=5	Flicker (1/f) noise only. This type of noise is produced only by bipolar and unipolar devices.

To account for flicker (1/f) noise in the transient simulation, additional parameters must be defined:

FREQ=val	Single frequency/low frequency of the band `[Freq, FreqH]`.
FREQH=val	High frequency of the band `[Freq, FreqH]`.

If `NOISE` or `NOISE=3` is given in a `.TRAN` statement, but the parameter `FREQ` is not specified, flicker noise is not taken into account, if the circuit consist of such noise sources.

If `NOISE=5` is given in a `.TRAN` statement, and the frequency parameter `FREQ` is not specified, SmartSpice uses the following default value: `FREQ=20 Hz`.

NOISE_RELTOL=val	Relative tolerance for LTE algorithm applied to calculate timestep during noise simulation. `NOISE_RELTOL` provides separate automatic timestep control mechanisms for transient simulation and noise simulation. If the calculated timestep from noise simulation is smaller than current timestep from transient analysis, then the timepoint is set back (timestep reversal) and the calculated timestep is used to increment the time for transient simulation. Default is 0.05. `NOISE_RELTOL=0` turns off this additional automatic timestep control algorithm.
NOISE_IC=ZERO	Sets Initial Conditions for the noise voltage variances at time=0 to 0 for noise simulation.
NOISE_IC=AUTO	Forces Initial Conditions calculation at `time=0`. Default is `AUTO`.

Examples

```
.TRAN  1ps 100ns NOISE   Freq=20 Freqh=2kHz
.TRAN  1ps  50ns NOISE=5 Freq=20 Freqh=20megHz
.TRAN  1ns  20us NOISE Freq=2kHz NOISE_RELTOL=0.02 NOISE_IC=ZERO
```

15.2 Transient Noise Analysis Output

The transient noise analysis calculates the noise correlation matrix of circuit variables (voltages at the nodes, which are connected to capacitors) *K(t)* at each time point during the transient process.

$$K(t) = \varepsilon\left[x_{noise}(t) \times x_{noise}(t)^{T}\right]$$

Correlation matrix element $K_{i,j}(t)$ is the noise voltage variance (mean-squared noise power) or the noise voltage correlations between i and j noise node voltages. The SmartSpice statements `.PRINT`, `.PLOT`, `.IPRINT`, `.IPLOT` and `.MEASURE` can be used to generate, print, plot, and measure noise voltage variance vectors.

Noise voltage variances (or noise voltage correlations) between nodes `nodename1` and `nodename2` are specified by the syntax:

```
k(nodename1, nodename2)
```

where `nodename` is either a number or a descriptive text string.

Noise voltage auto-correlation (correlation matrix diagonal elements $K_{i,i}(t)$) at node with name `nodename` is specified by the syntax:

```
k(nodename) or k(nodename, nodename).
```

Examples

`k(2)`: Noise voltage variance at node 2.

`k(out)`: Noise voltage variance at node out.

`k(4,5)`: Noise voltage variance between nodes 4 and 5.

15.3 Noise Models

Transient noise analysis is supported for circuits that contain the following devices:

- independent voltage and current sources;
- analog behavior devices;
- current-controlled voltage sources;
- voltage-controlled voltage sources;
- current-controlled switches;
- voltage-controlled switches;
- capacitors;
- ferroelectric capacitors;
- inductors;
- resistors;
- diodes - all models: Standard junction (level=1, 3), Fowler-Nordheim (level=2), JUNCAP (level=9), Philips 500 (level=500);
- Bipolar: BJT (level=1, 2), VBIC (level=4), HICUM (level=8), HBT (level=20), MODELLA (level=500), and MEXTRAM (level=503, 504);
- JFET/ MESFET (all levels);
- MOSFET: MOS (level=1, 2, 3), BSIM1 (level=4, 13), BSIM3v3 (level=11, 49, 53, 81), MOS11 (level=43), BSIM4 (level=14, 54), EKV (level=44), BSIM3H (level=88);

- SOI: LETISOI (level=32), BSIM3SOI3 (level=57);
- Thin Film Transistors: RPI (level=35) models; Polysilicon RPI (level=36) models.

We have accounted for the thermal, shot and flicker noise sources in the simulation of devices.

Thermal noise is due to the random thermal motion of electrons. It exists in resistive materials and is unaffected by the presence or absence of a direct current. Thermal noise is modeled as a shunt current noise source $h_{th}(t)$ with auto-correlation function

$$Z_{thermal}(t+\tau, t) = \varepsilon[h_{th}(t+\tau)h_{th}(t)] = 2kTG(t)\delta(\tau) = IN_{thermal}(t)^2\delta(\tau)$$

where k is the Boltzmann's constant, T is the temperature in degrees Kelvin, $G(t)$ is the time-varying conductance, and $IN_{thermal}(t) = \sqrt{2kTG(t)}$ is the intensity.

Shot noise is due to random emission of charge carriers across a pn-junction. It is always associated with a direct current flow and is presented in diode and bipolar transistor. Shot noise is modeled as current noise source $h_{sh}(t)$ with auto-correlation function

$$Z_{sh}(t+\tau, t) = \varepsilon[h_{sh}(t+\tau)h_{sh}(t)] = qI_D(t)\delta(\tau) = IN_{sh}(t)^2\delta(\tau)$$

where $IN_{sh}(t) = \sqrt{qI_D(t)}$ is the intensity and $I_D(t)$ is the time-varying current.

Flicker noise, also called 1/f noise, is present in all active devices and has various origins. It is now widely believed that 1/f noise in a MOSFET is due to traps in the gate oxide, yet depends on the total trapped electron number (i.e *dc* current), and if the current through device is kept low enough, thermal noise will be predominant in the circuit. Flicker noise is modeled as current noise source $h_{fl}(t)$ with auto-correlation function:

$$Z_{fl}(t+\tau, t) = \varepsilon[h_{fl}(t+\tau)h_{fl}(t)] = K(f, t)\delta(\tau) = IN_{fl}(t)^2\delta(\tau)$$

where $IN_{fl}(t) = k(f)I_D(t)$ is the intensity,

$k(f) = \frac{K^2}{f}$ for given fixed frequency f', and $k(f) = K^2\log(f_h/f_l)$ for integrating noise in given frequency band (f_l is the lowest frequency and f_h is the highest frequency),

K^2 is appropriate device constant. As the 1/f noise cannot be described by a single unified model, K^2 expression for every device is given in the SPICE Models Manual, and $I_d(t)$ is the time-varying current.

Although the mechanisms that generate flicker noise are not well understood, these models have not yet been thoroughly tested and proven correct.

Table 15-1 Noise sources are inserted into the large-signal model

Model	Type noise source	Intensity, IN
Resistor	Resistor thermal noise	$\sqrt{(2kT)/R}$
Standard and Philips Diodes	Diode resistance thermal noise	$\sqrt{(2kT)/R_S}$
	Diode junction shot noise	$\sqrt{q \times I_D}$
Fowler - Nordheim and JUNCAP Diodes	Diode junction shot noise	$\sqrt{q \times I_D}$
BJT	Base resistor thermal noise	$\sqrt{(2kT)/R_B}$
	Collector resistor thermal noise	$\sqrt{(2kT)/R_C}$
	Emitter resistor thermal noise	$\sqrt{(2kT)/R_E}$
	Base junction shot noise	$\sqrt{q \times I_B}$
	Collector junction noise	$\sqrt{q \times I_C}$
VBIC	Base resistor (Rbx, Rbp, Rbi) thermal noise	$\sqrt{(2kT)/R_B}$
	Collector resistor (Rcx, Rci) thermal noise	$\sqrt{(2kT)/R_C}$
	Emitter resistor thermal noise	$\sqrt{(2kT)/R_E}$
	Substrate resistor thermal noise	$\sqrt{(2kT)/R_S}$
	Base junction shot noise	$\sqrt{q \times I_{be}}$
	Collector junction shot noise	$\sqrt{q \times I_{CC}}$
	Parasitic device base current shot noise	$\sqrt{q \times I_{bep}}$
	Parasitic device transport current shot noise	$\sqrt{q \times I_{ccp}}$

Model	Type noise source	Intensity, IN
HICUM	Base resistor (Rbx, Rbi) thermal noise	$\sqrt{(2kT)/R_B}$
	Collector resistor thermal noise	$\sqrt{(2kT)/R_C}$
	Emitter resistor thermal noise	$\sqrt{(2kT)/R_E}$
	Collector current shot noise	$\sqrt{q \times I_C}$
	Collector-base avalanche current shot noise	$\sqrt{q \times I_{avl}}$
	Base current components (Ibei, Ibci, Ibep, Ibcx) shot noise	$\sqrt{q \times I_B}$
	Parasitic transistor substrate current shot noise	$\sqrt{q \times I_{Sci}}$
HBT	Base resistor (Rbx, Rbi) thermal noise	$\sqrt{(2kT)/R_B}$
	Collector resistor (Rcx, Rci) thermal noise	$\sqrt{(2kT)/R_C}$
	Emitter resistor (Re, Rex) thermal noise	$\sqrt{(2kT)/R_E}$
	Base-emitter current (Ibei) shot noise	$\sqrt{q \times I_{bei}}$
	Base-emitter current (Ibex) shot noise	$\sqrt{q \times I_{bex}}$
	Collector current shot noise	$\sqrt{q \times I_C}$

Model	Type noise source	Intensity, IN
MODELLA	Base-collector resistor thermal noise	$\sqrt{(2kT)/R_{bc}}$
	Base-emitter resistor thermal noise	$\sqrt{(2kT)/R_{be}}$
	Collector resistor (Rcex, Rcin) thermal noise	$\sqrt{(2kT)/R_{C}}$
	Emitter resistor (Reex, Rein) thermal noise	$\sqrt{(2kT)/R_{E}}$
	Substrate resistor thermal noise	$\sqrt{(2kT)/R_{Sub}}$
	Collector lateral (forward, reverse) current shot noise	$\sqrt{q \times I_{Clat}}$
	Collector-emitter forward current shot noise	$\sqrt{q \times I_{Fver}}$
	Collector-emitter reverse current shot noise	$\sqrt{q \times I_{Rver}}$
	Base-emitter current shot noise	$\sqrt{q \times I_{be}}$
MEXTRAM	Base resistor (Rbc, Rbv) thermal noise	$\sqrt{(2kT)/R_{B}}$
	Collector resistor (Rcc, Rcv) thermal noise	$\sqrt{(2kT)/R_{C}}$
	Emitter resistor thermal noise	$\sqrt{(2kT)/R_{Eeff}}$
(all levels)	Base current (forward, reverse) shot noise	$\sqrt{q \times I_{B}}$
	Collector junction shot noise	$\sqrt{q \times I_{C}}$
	Emitter-base sidewall current shot noise	$\sqrt{q \times I_{Sib}}$
	Extrinsic current shot noise	$\sqrt{q \times I_{ex}}$
Level = 504	Extending modelling current shot noise	$\sqrt{q \times I_{Xex}}$
	Substrate current shot noise	$\sqrt{q \times I_{Sub}}$
	Extending modelling substrate current shot noise	$\sqrt{q \times I_{XSub}}$

Model	Type noise source	Intensity, IN
JFET/ MES-FET, BSIM3	Channel resistivity thermal noise	$\sqrt{2kT \times \frac{2}{3} \times (g_m + g_{ds})}$
	Source resistor thermal noise	$\sqrt{(2kT)/R_S}$
	Drain resistor thermal noise	$\sqrt{(2kT)/R_D}$
MOS, BSIM3v3, BSIM4, BSIM3H, EKV	Channel resistivity thermal noise	$\sqrt{2kT \times g_{dc}}$ switch
		$\sqrt{2kT \times \frac{2}{3} \times g_m}$ in saturation
	Source resistor thermal noise	$\sqrt{(2kT)/R_S}$
	Drain resistor thermal noise	$\sqrt{(2kT)/R_D}$
BSIM1, Philips MOSFET's	Channel resistivity thermal noise	$\sqrt{2kT \times \frac{2}{3} \times g_m}$
	Source resistor thermal noise	$\sqrt{(2kT)/R_S}$
	Drain resistor thermal noise	$\sqrt{(2kT)/R_D}$
SOI transistors, TFT	Channel resistivity thermal noise	$\sqrt{2kT \times g_{dc}}$ switch
		$\sqrt{2kT \times \frac{2}{3} \times g_m}$ in saturation
	Source resistor thermal noise	$\sqrt{(2kT)/R_S}$
	Drain resistor thermal noise	$\sqrt{(2kT)/R_D}$

Specification of noise model level, using `val` value in `NOISE=val`, allows you to turn off a part of noise sources in the models.

Table 15-2 Noise Sources used in model vs. noise model level NOISE=val

Model	Noise source	1	2	3 (Default)	4	5
Resistor	Resistor thermal noise		Yes	Yes	Yes	
Diode	Diode resistance thermal noise	Yes		Yes		
	Diode junction shot noise			Yes	Yes	
	Diode flicker (1/f) noise			Yes		Yes
BJT	Base resistor thermal noise	Yes		Yes		
	Collector resistor thermal noise		Yes	Yes		
	Emitter resistor thermal noise		Yes	Yes		

Model	Noise source	1	2	3 (Default)	4	5
	Base junction shot noise			Yes	Yes	
	Collector junction noise			Yes	Yes	
	Flicker (1/f) noise			Yes		Yes
VBIC	Base resistor (Rbx) thermal noise	Yes		Yes		
	Base resistor (Rbp, Rbi) thermal noise		Yes	Yes		
	Collector resistor (Rcx, Rci) thermal noise		Yes	Yes		
	Emitter resistor thermal noise		Yes	Yes		
	Substrate resistor thermal noise		Yes	Yes		
	Base junction shot noise			Yes	Yes	
	Collector junction shot noise			Yes	Yes	
	Parasitic device base current shot noise			Yes	Yes	
	Parasitic device transport current shot noise			Yes	Yes	
	Flicker (1/f) noise			Yes		Yes
HICUM	Base resistor (Rbx) thermal noise	Yes		Yes		
	Base resistor (Rbi) thermal noise		Yes	Yes		
	Collector resistor thermal noise		Yes	Yes		
	Emitter resistor thermal noise		Yes	Yes		
	Collector current shot noise			Yes	Yes	
	Collector-base avalanche current shot noise			Yes	Yes	
	Base current components of shot noise			Yes	Yes	
	Parasitic transistor substrate current shot noise			Yes	Yes	
	Flicker (1/f) noise			Yes		Yes
HBT	Base resistor (Rbx) thermal noise	Yes		Yes		
	Base resistor (Rbi) thermal noise		Yes	Yes		
	Collector resistor (Rcx, Rci) thermal noise		Yes	Yes		
	Emitter resistor (Re, Rex) thermal noise		Yes	Yes		
	Base-emitter current (Ibei) shot noise			Yes	Yes	
	Base-emitter current (Ibex) shot noise			Yes	Yes	
	Collector junction shot noise			Yes	Yes	
	Flicker (1/f) noise			Yes		Yes

Model	Noise source	1	2	3 (Default)	4	5
MODELLA	Base-collector resistor thermal noise	Yes		Yes		
	Base-emitter resistor thermal noise		Yes	Yes		
	Collector resistor (Rcex, Rcin) thermal noise		Yes	Yes		
	Emitter resistor (Reex, Rein) thermal noise		Yes	Yes		
	Substrate resistor thermal noise		Yes	Yes		
	Collector lateral (forward, reverse) current shot noise			Yes	Yes	
	Collector-emitter current (forward, reverse) shot noise			Yes	Yes	
	Base-emitter current shot noise			Yes	Yes	
	Flicker (1/f) noise			Yes		Yes
MEXTRAM	Base resistor (Rbc) thermal noise	Yes		Yes		
	Base resistor (Rbv) thermal noise		Yes	Yes		
	Collector resistor (Rcc, Rcv) thermal noise		Yes	Yes		
	Emitter resistor thermal noise		Yes	Yes		
	Base current (forward, reverse) shot noise			Yes	Yes	
	Collector junction shot noise			Yes	Yes	
	Emitter-base side wall current shot noise			Yes	Yes	
	Extrinsic current shot noise			Yes	Yes	
	Extending modelling current shot noise			Yes	Yes	
	Substrate current shot noise			Yes	Yes	
	Extending modelling substrate current shot noise			Yes	Yes	
	Flicker (1/f) noise			Yes		Yes
JFET/MESFET (all levels)	Channel resistivity thermal noise	Yes		Yes		
	Source resistor thermal noise		Yes	Yes		
	Drain resistor thermal noise		Yes	Yes		
	Flicker (1/f) noise			Yes		Yes
MOSFET transistors (all levels)	Channel resistivity thermal noise	Yes		Yes		

Model	Noise source	1	2	3 (Default)	4	5
	Source resistor thermal noise		Yes	Yes		
	Drain resistor thermal noise		Yes	Yes		
	Flicker (1/f) noise			Yes		Yes
SOI, TFT transistors (all levels)	Channel resistivity thermal noise	Yes		Yes		
	Source resistor thermal noise		Yes	Yes		
	Drain resistor thermal noise		Yes	Yes		
	Flicker (1/f) noise			Yes		Yes

15.4 Singular Matrix Problem

When forming the matrix *M* and *N* for matrix equation $MK + KM^T = N$, the singular matrix problem can occur. These problems are caused:

1) by voltage source, which has:

- both nodes connected to a capacitor (capacitors from the device equivalent circuit too).
- one node connected to a capacitor, and the other is grounded.

2) by existing nodes in the circuit, which are connected to the capacitor, but haven't a capacitive path to ground. These nodes are not a voltage source nodes.

The term **voltage source** includes the following devices: independent voltage source (`V`), voltage-controlled voltage source (`E`), current-controlled voltage source (`H`), and analog behavior devices (`V-`, `A-`, `E-` and `H-type`).

To avoid these problems, a serial resistor (RS=0.1 Ohm) and a grounded capacitor (CG=1 fF) are added automatically to non-ground nodes for all voltage sources in the circuit. Figure 15-1 shows an equivalent circuit of independent voltage source.

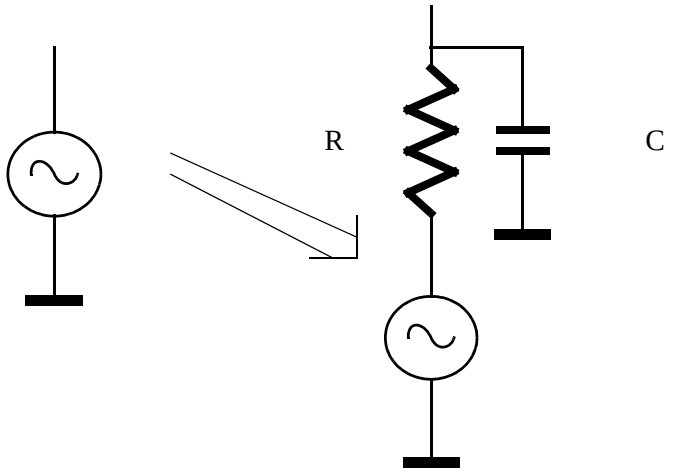


Figure 15-1 Example of an independent voltage source

To provide control for additional devices in the circuit, a few parameters were added into the source line:

```
Vxxx  n+  n-  ...  <NORC> <NORS> <NOCG> <RS=val> <CG=val>
Axxx  n+  n-  ...  <NORC> <NORS> <NOCG> <RS=val> <CG=val>
Exxx  n+  n-  ...  <NORC> <NORS> <NOCG> <RS=val> <CG=val>
Hxxx  n+  n-  ...  <NORC> <NORS> <NOCG> <RS=val> <CG=val>
```

NORC	Suppresses connection of both serial resistor RS and grounded capacitor CG.
NORS	This flag suppresses connection of the serial resistor RS.
NOCG	This flag suppresses connection of the grounded capacitor CG.
RS=val	Specifies the value of the serial resistor in Ohms. Default is 0.1 Ohm.
CG=val	Specifies the value of the grounded capacitor in Farads. Default is 1e-15 F.

15.5 Examples

```
Example 1: Transient Noise Analysis
.options nomod accurate

.param Vdrain=1v
Vdd vdd 0 dc Vdrain
Vinput vin 0 PULSE(0 5 200n 10n 10n 1000n 1500n)

.param nchwidth=2.6U
MU vdd vin vss 0 NENH   L=.28U  W=nchwidth
+ as='nchwidth*0.49u'   ad='nchwidth*0.49u'
+ ps='nchwidth*2+0.98u' pd='nchwidth*2+0.98u'
+ nrd='0.15u/nchwidth'  nrs='0.15u/nchwidth'
Cload vss 0 C=1p
.tran .1n 1500n  NOISE SWEEP Vdrain LIST 2 1v 5v
.probe k(vss)
.MEASURE TRAN CORMAX MAX K(vss) FROM=400n TO=1000n
.MEASURE TRAN CORMIN MIN K(vss) FROM=400n TO=1000n
.MEASURE TRAN CORAVG AVG K(vss) FROM=400n TO=1000n

.model nenh NMOS
+ VERSION=3.1
+ Level=49
+ Tnom=27.0
.end
```

This example demonstrates the noise simulation for single MOSFET transistor with `Cload=1pF` in linear (`Vdrain=1V`) and in saturation (`Vdrain=5V`) regions. It is known that for linear case noise variance should be equal to `kT/Cload = 1.38e-23 x 300 / 1.e-12 = 4.14e-9`, and in saturation `kT/(2 x Cload) = 1.38e-23 x 300 / 2.e-12 = 2.07e-9`.

The simulation shows the close results:

```
                    Example 1: Transient Noise Analysis
      Transient Analysis, 27 deg C, step vdrain = 1,Tue Jan 11
12:04:54 2005

cormax         =   4.1380e-09 at=   8.0732e-07
cormin         =   4.1380e-09 at=   8.0732e-07
coravg         =   4.1380e-09 from=   8.0000e-07 to=   1.2000e-06

 *** SWEEP : parameter  vdrain =  5.0000e+00  ( step # 2 ) ***
                    Example 1: Transient Noise Analysis
      Transient Analysis, 27 deg C, step vdrain = 5,Tue Jan 11
12:04:55 2005

*cormax         =   2.3028e-09 at=   1.1927e-06
*cormin         =   2.2974e-09 at=   8.0270e-07
*coravg         =   2.3010e-09 from=   8.0000e-07 to=   1.2000e-06
```

Note: * Values are slightly different from the formula KT/(2 x Cload) which is approximation for saturation mode.

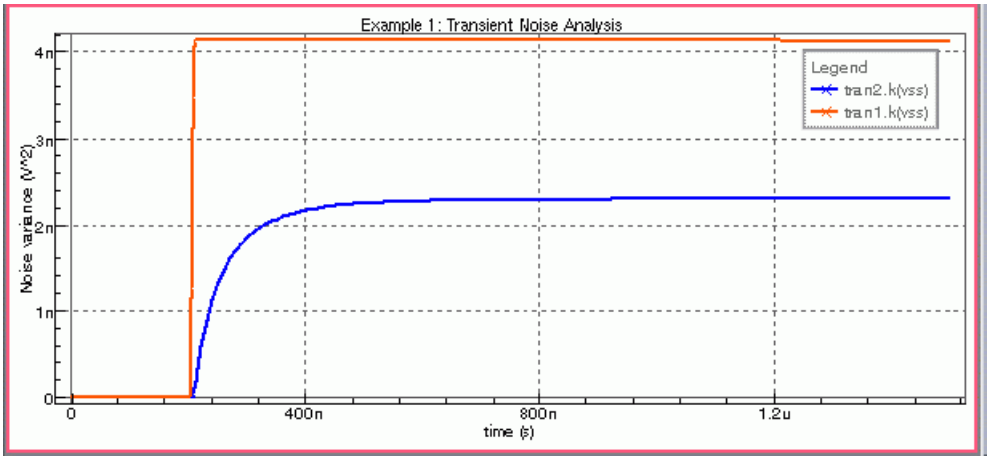


Figure 15-2 Noise variance at the vss node

```
Example 2: Inverter Transient Noise Analysis
.OPTIONS nomod DEFNRD=1 DEFNRS=1
.TRAN 1n 50n NOISE
.probe v(inp) v(out) k(out)

*========= Circuit ================================
.PARAM w2p=5.6u w2n=4.0u bulkv=0.0

MP1 mid1 inp vss pbulk pmos w=5.6u l=1.3u
+   PS=33.45U   AS=35.10P   PD=33.45U     AD=35.10P
MN1 mid1 inp gnd nbulk    nmos w=4.0u l=0.8u
+   PS=19.50U   AS=17.10P   PD=19.50U     AD=17.10P
MP2 mid2 mid1 vss pbulk pmos w='w2p' l=1.3u
+   PS=33.45U   AS=35.10P   PD=33.45U     AD=35.10P
MN2 mid2 mid1 gnd nbulk    nmos w='w2n' l=0.8u
+   PS=19.50U   AS=17.10P   PD=19.50U     AD=17.10P
MP3 out mid2 vss pbulk pmos w=5.6u l=1.3u
+   PS=33.45U   AS=35.10P   PD=33.45U     AD=35.10P
MN3 out mid2 0 nbulk    nmos w=4.0u l=0.8u
+   PS=19.50U   AS=17.10P   PD=19.50U     AD=17.10P
* **    Parallel Transistors
MP33 out mid2 vss pbulk pmos w=5.6u l=1.3u
+    PS=33.45U   AS=35.10P   PD=33.45U     AD=35.10P
MN33 out mid2 0 nbulk    nmos w=4.0u l=0.8u
+    PS=19.50U   AS=17.10P   PD=19.50U     AD=17.10P

Cout out 0 1pf

vs   vss 0   DC 5
vin inp 0 pulse(0 5 0 1n 1n 10n 22n)
Vpbulk pbulk 0 DC 4.7
Vnbulk nbulk 0 DC 0.7

*========                                                    Models
================================================
.model nmos nmos level=49
.model pmos pmos level=49
.end
```

This inverter loaded with a 1 pF capacitor was driven with a periodic waveform at the input, and a noise simulation was performed.

In Figure 15-3 the large-signal waveforms at the input and output of this inverter can be seen.

As seen in Figure 15-4, the noise at the output is non-stationary. The noise variance (mean-squared noise power) is highest during low-to-high and high-to-low transitions of the large-signal output waveform.

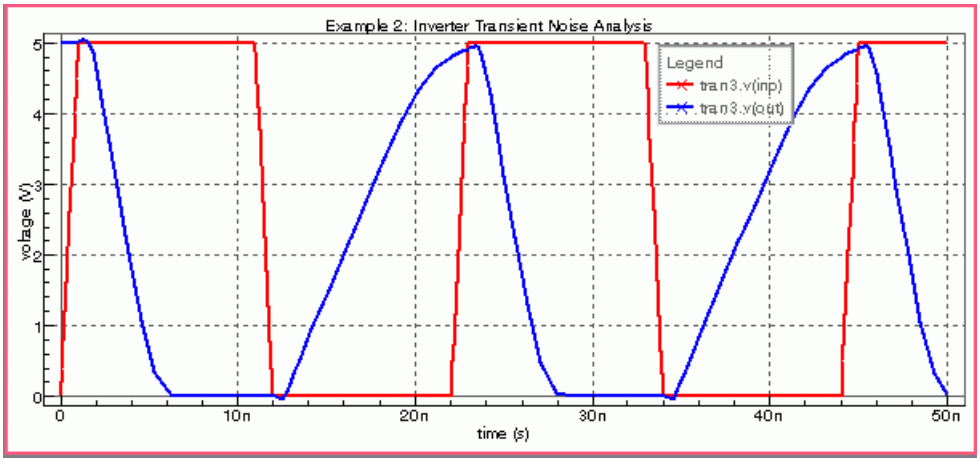


Figure 15-3 Large-signal input and output waveforms

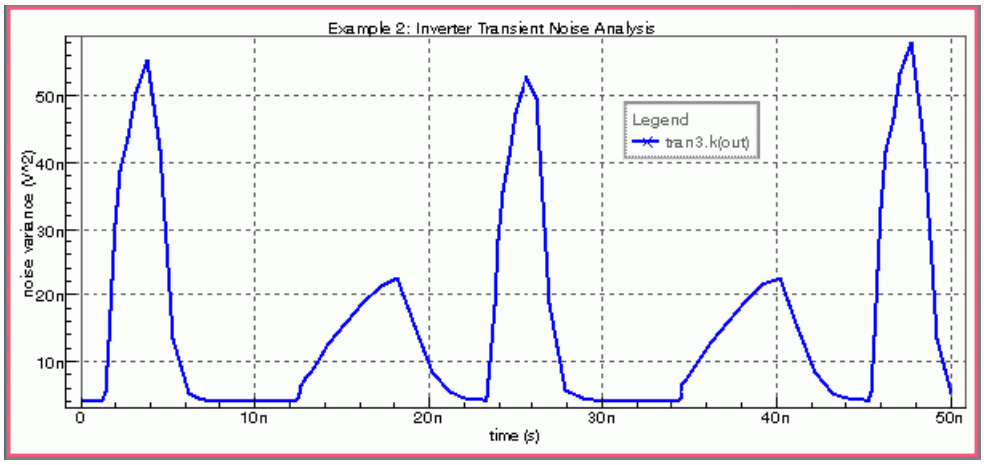


Figure 15-4 Noise voltage variance at the out node

```
Example 3: Parallel RLC Circuit
.option nomod

R1  2 0 1k
L1  2 0 1uH
C1  2 0 1pF
.IC v(2)=5V
.TRAN 0.0001n 10n uic NOISE NOISE_IC=ZERO
.probe k(2) v(2)
.end
```

Transient Noise Analysis computed the variance of the noise voltage across the simple parallel RLC circuit as a function of time from the zero initial condition, which was set by the keyword `NOISE_IC=ZERO`. The only noise source in this circuit models the thermal noise in the resistor. The result of noise simulation in Figure 15-5 shows that variance of noise voltage settles to the time-invariant steady-state value, as should be for the stable system.

In Figure 15-6 the voltage across RLC circuit can be seen.

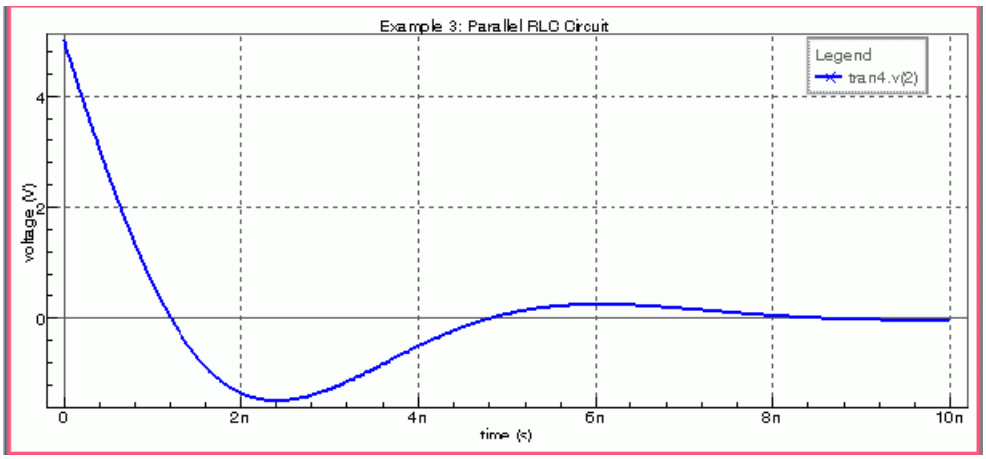


Figure 15-5 Noise voltage variance across parallel RLC circuit

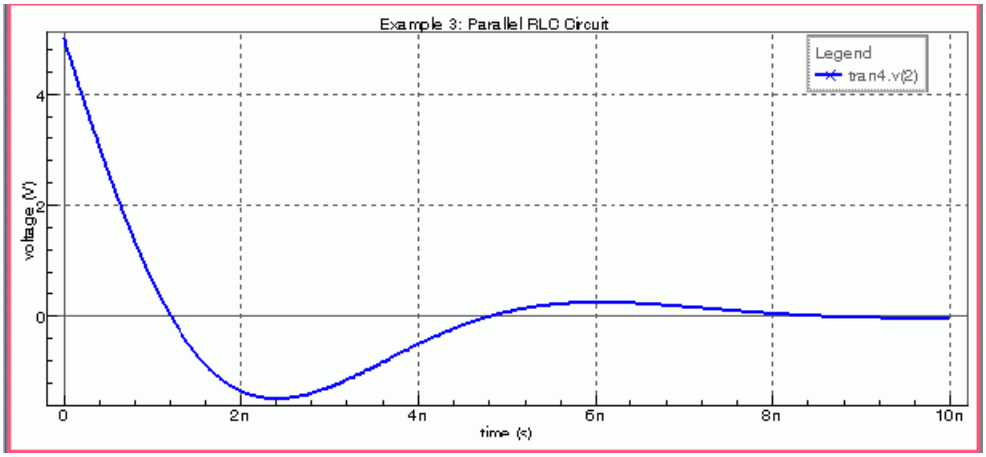


Figure 15-6 Voltage across parallel RLC circuit

Chapter 16
Timing Jitter Analysis

16.1 Introduction

Noise performance is a major concern for temporal circuit design, where most of the signals are binary. Noise in such circuits as oscillators, clock data recovery, PLL etc., are commonly characterized in terms of Jitter. Jitter is the time-domain characteristic, and is defined as undesired perturbation or uncertainly in the timing of the signal transitions. Jitter is caused by the effects of the device noise, power supply variations, non-stability of the reference clock, loading conditions, etc.

SmartSpice provides Timing Jitter Analysis as a single run method with the following postprocessing evaluation. Large signal noise generated by each device in the circuit is added to the signal at each iteration of every time step of the transient analysis. The thermal, shot and flicker noise that is generated within the integrated-circuit devices has a stochastic character. To model their behavior, Monte-Carlo technique is used to generate random numbers with Gaussian distribution based on intensity of the noise each device should produce. To get appropriate waveforms for jitter metrics measurements, transient analysis has to be run over many cycles of operation.

Syntax

To perform Timing Jitter Analysis in SmartSpice, the `.TRAN` statement must be used:

```
.TRAN 10p 1us JITTER NOISE<=val>
+ <NOISESCALE = val> <NOISESEED = val>
+ <NOISEFMIN = val> <NOISEFMAX = val> <NOISETMIN = val>
```

JITTER	Causes SmartSpice to perform Timing Jitter Analysis.
NOISE<=val>	Causes SmartSpice to account for existing circuit noise sources, depending on `val`: • `val=1,2`: Thermal noise only. This type of noise is produced by the resistors, passive and active devices which equivalent circuit consist of resistors and conductances, and by the channel of the unipolar devices. • `val=3`: (default) Thermal, shot and flicker (1/f) noise. • `val=4`: Shot noise only. This type of noise is produced mostly by the pn-junction current of the diodes, bipolar and unipolar devices. • `val=5`: Flicker (1/f) noise only. This type of noise is produced only by poly resistors, bipolar and unipolar devices.
NOISESCALE=val	Noise scale factor applied to all noise sources to artificially increase the small noise in the circuit. Default is 1.0.
NOISESEED=val	Seed for random number generator. Default is `1`.
NOISEFMIN=val	The minimum frequency of pseudorandom noise sources. Default is `NOISEFMAX value`.
NOISEFMAX=val	The bandwidth of pseudorandom noise sources. Default is `10GHz`.

NOISETMIN=val	Minimum time interval to update noise sources. Default is 1/NOISEFMAX.

16.2 Monte Carlo Approach

Transient noise analysis shows the effect of noise on the signal magnitude. From the transient noise analysis results, jitter can be measured. The two jitter measurements are time interval error and auto-correlation function. Time interval error measures the timeshift behavior relative to a reference signal. This section describes Monte Carlo approaches, where the device noise is modeled as uncorrelated random signal sources to predict the statistical characteristics of the circuit performance due to device noise.

Syntax

To perform Monte Carlo Approach Trannoise Analysis in SmartSpice, the .TRANNOISE statement must be used:

```
.TRANNOISE output <SWEEP> <MONTE=valm
+ | list(numl) | valm firstrun=numf |
+ list ([num0] [num1:num2] [num3] ... ) <PRMC<=val>>>
+ [FMIN=val][FMAX=val][SCALE=val][SEED=vals]
```

output	Output variables. Can be node voltage or voltage difference.
val	The number of Monte Carlo repetitions.
list	Defines the sequence of iterations. Can write only one number after list (numl). For example, in [num1:num2], the colon represents "from ... to ...". Specifying only one number makes SmartSpice run only at that single point. The numbers after list cannot be the parameter.
firstrun=numf	Specifies the desired number of iterations. SmartSpice runs from numf to numf+val-1. numf can be a parameter.
PRMC<=val>	Causes SmartSpice to print the values of modified and measured parameters on each Monte Carlo iteration. In HSPICE mode, the flag PRMC saves distributed parameters in the measure file, if the .MEASURE statement is used. The PRMC=2 saves all parameters in the raw file, if the option sweepmonte is present. The feature doesn't work in HSPICE compatible mode.
FMIN	Base frequency used for modeling frequency dependent noise sources. Sets bandwidth for contributing noise sources.
FMAX	Maximum frequency used for modeling frequency dependent noise sources. Sets bandwidth for contributing noise sources.
SCALE	Scale factor that can be applied to uniformly amplify the intensity of all device noise sources to exaggerate their contributions.

The transient noise analysis requires an accompanying .TRAN analysis, which determines the time-sampling, matrix solutions, and deterministic output waveforms. The .TRANNOISE command is used to activate transient noise and to compute the additional noise variables.

When you specify .TRANNOISE without an appropriate .TRAN analysis statement, a transient analysis statement will be created with follow settings:

```
.TRAN STEP = '1/(FMAX*4)' STOP = '1/FMIN' START = 0 tmax = 'STEP'
```

The Monte Carlo approach can capture very non-linear noise behaviors. This is useful, for example, when the responses of circuits with noise are known to have non-Gaussian variations about their noise-less simulations.

Examples

```
.TRANNOISE v(out) SWEEP MONTE=10
.TRANNOISE v(out) SWEEP MONTE=1 FIRSTRUN=50 SCALE=10.0
.TRANNOISE v(out) SWEEP MONTE=20 FIRSTRUN=30 FMIN=10k FMAX=100MEG
```

Example 1 generates 10 Monte Carlo noise simulations.

Example 2 generates a single noise simulation, with seed value of 50, with all noise sources amplified by a factor of 10.

Example 3 generates 20 Monte Carlo noise simulations starting with the seed value (i.e., index) of 30 for the first simulation, placing a lower bound on flicker noise to be 10kHz, and an upper bound on all noise power at 100MHz.

16.3 Jitter Metrics Measurement

The following jitter metrics define the sequence of times for rise or fall threshold crossings at transitions that occur in the output noisy waveform.

- Jper (period jitter): The time difference between a measured cycle period and the ideal cycle period. Due to its random nature, this jitter can be measured as Peak-to-Peak or by Root Mean Square (RMS).
- Jcc (cycle-to-cycle jitter): The time difference of two adjacent clock periods. This is also called “Short Term Jitter”. The metric depends only on the adjacent clock cycles. Jcc is the only jitter metric that is suitable for use when flicker noise is present. It is measured by its Peak-to-Peak value in a given time interval.
- Jacc(n) (accumulated jitter): The time displacement of the edges of a clock relative to the triggering edge of the same clock. Jacc(n) is a function of n and it's the general case of Jper. It is measured by its Peak-to-Peak value or RMS.

Syntax

To perform Jitter Metrics Measurement in SmartSpice, the .MEASURE statement must be used:

```
.MEASURE ResName JITTER OutVar RISE | FALL
+ <FROM=val|name> <TO=val|name>
+ <VAL=val|name>
```

ResName	Measurement name is using as prefix of the vector name.
JITTER	Name of the Measure function.
OutVar	Name of the output vector from .TRANxxx plot.

RISE|FALL	Edge type to provide measurement.
FROM=val	Time point where measurement begins.
TO=val	Time point where measurement ends.
VAL	Threshold absolute value for OutVar. Default is 0.5(Max-Min).

Example

```
.measure VCO jitter V(FOUT) RISE From=15ns To=500ns
```

This statement provides measurement over the vector V(FOUT) at the RISE edges that occur in time interval 15 - 500 ns with threshold 0.5.

```
.measure Clock jitter V(sm) From=40n To=5u FALL VAL=1.15
```

This statement provides measurement over the vector V(sm) at the FALL edges that occur in time interval 40ns - 5u with threshold 1.15V.

Measurement Results

The .measure statement produces a .MEASxxx plot, which consists of the following vectors:

- ResName_Jacc: Accumulated jitter (time unit seconds)
- ResName_Jcc: Cycle-to-Cycle jitter (time unit seconds)
- ResName_Jper: Period jitter (time unit seconds)

- `Period`: The signal period with the jitter variation (time unit seconds)

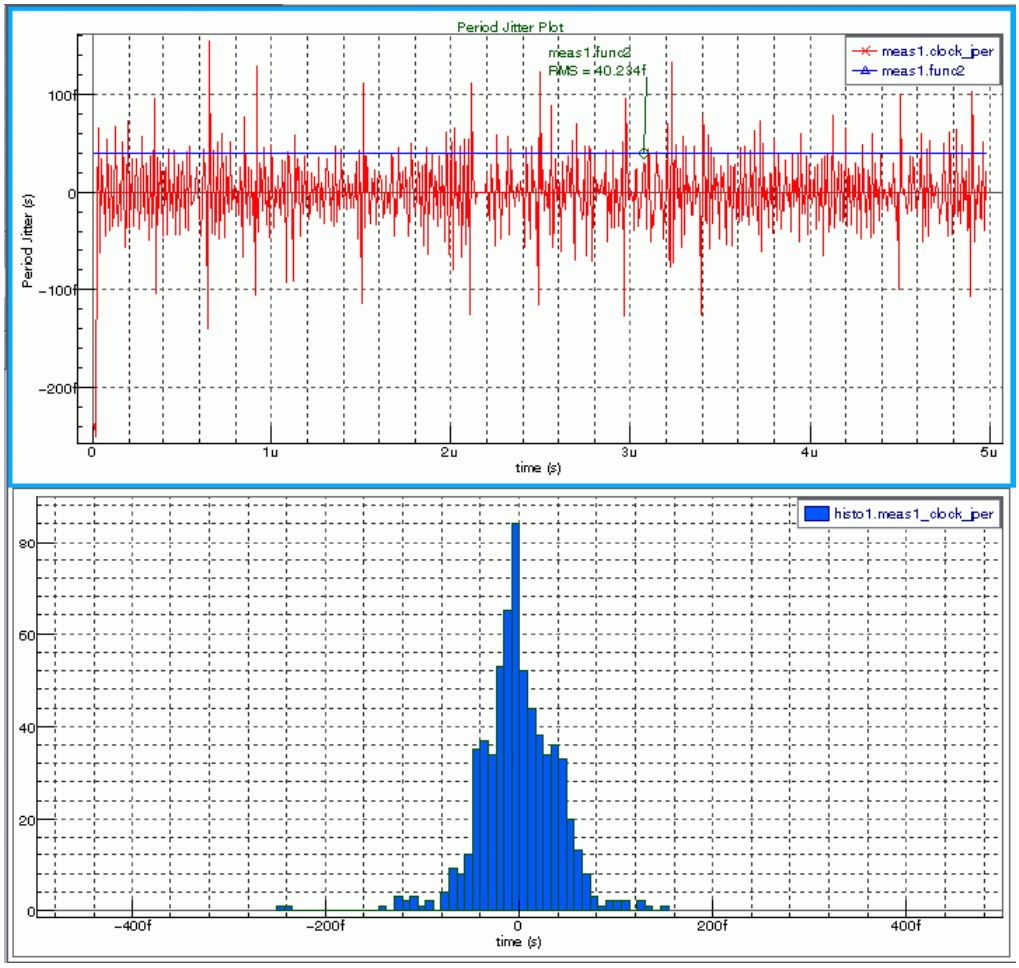


Figure 16-1 Period jitter Plot and Histogram

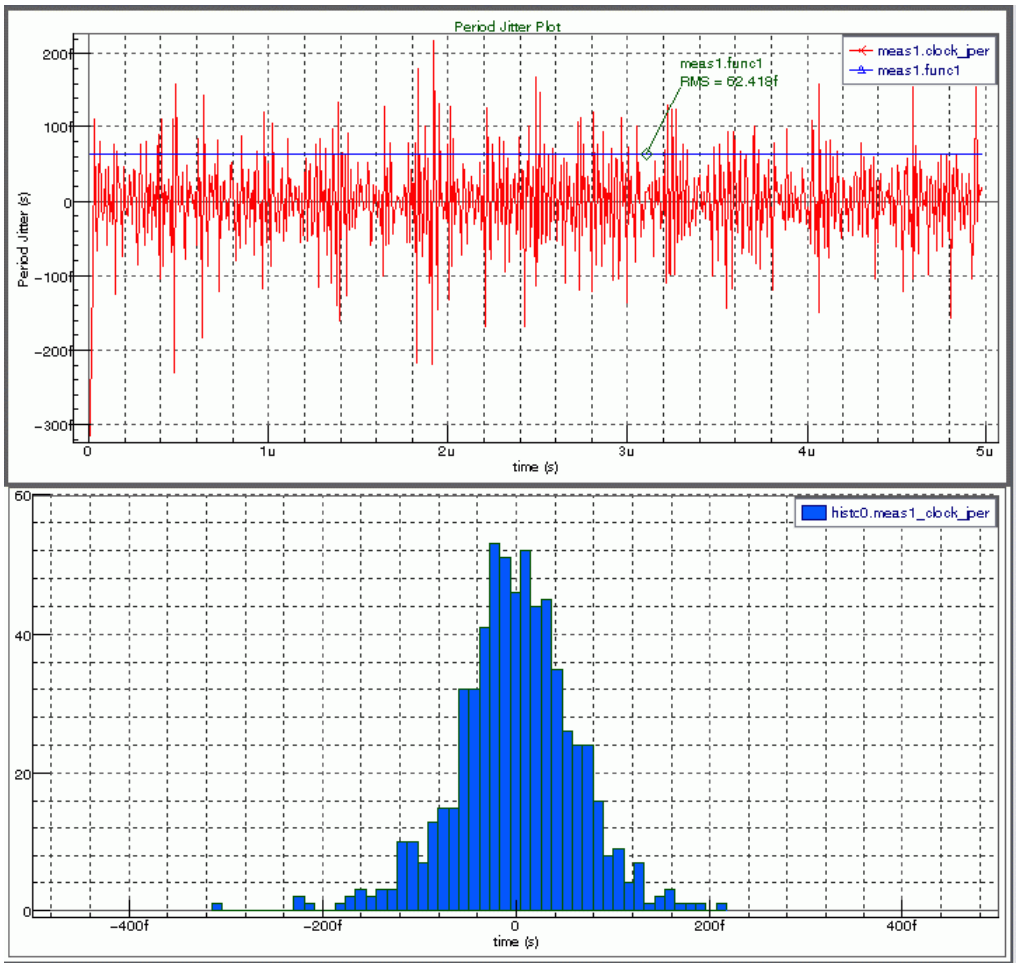


Figure 16-2 Period jitter Plot and Histogram

Chapter 17
Reliability Analysis Using Single Event Effects

17.1 Introduction

Single Event Effects (SEEs) are caused by a single, energetic particle, and can take on many forms. Single Event Upsets (SEUs) are soft errors, and non-destructive. They normally appear as transient pulses in logic or support circuitry, or as bitflips in memory cells or registers. Several types of hard errors, which are potentially destructive, can appear. Single Event Latchup (SEL) results in a high operating current, above device specifications, and must be cleared by a power reset. Other hard errors include burnout of power MOSFETS, gate rupture, frozen bits, and noise in CCDs.

17.2 Single Event Effect

In the space environment, spacecraft designers have to be concerned with two main causes of a Single Event Effect (SEE): cosmic rays and high energy protons. For cosmic rays, SEEs are typically caused by its heavy ion component. These heavy ions cause a direct ionization SEE; if an ion particle transversing a device deposits sufficient charge, an event such as a memory bit flip or transient may occur. Cosmic rays may be galactic or solar in origin.

Protons, usually trapped in the earth's magnetic field or from solar flares, may cause direct ionization SEEs in very sensitive devices. However, a proton may more typically cause a nuclear reaction near a sensitive device area, and create an indirect ionization effect potentially causing an SEE.

As spacecraft become driven more and more to reduce parameters, such as power, weight, volume and cost, while requiring increased functionality, emerging commercial technologies - often vulnerable to SEEs - have come to the forefront. These technologies include high speed and low power CMOS, HBTs and fiber optics. Types of integrated circuits (ICs) that utilize these technologies range from complex microprocessors to dense SRAMs. All mission-critical devices must be tested for SEE; one Latchup or Gate Rupture in a critical system can destroy a mission.

17.3 Single Event Upset

When semi-conductor devices are used at high altitudes or in space they are exposed to energetic particles that can penetrate the active region of an operating device and cause signal corruption. The Single Event Effects simulation in SmartSpice is to mimic particle penetration effects that cause a trail of charge along the trajectory path. This charge will mainly re-combine in the presence of no electric field, but across a devices biased junction produces a current spike that can cause logic bits to flip state or amplifier circuits to saturate and give false outputs. This analysis allows a current spike to be introduced tailored to the energy of the incident particle and study the circuits response to this disturbance.

Single Event Upset (SEU) occurs mainly when a cosmic high-energetic particle strikes a transistor, and causes a change in a logic state and a disruption in the system logic. Even if dynamic memories or flip-flop registers are sensitive to this effect, this may occur in analog and optical components.

The incident SEU-inducing particles must deposit enough energy to produce errors in the circuit output. The impact of incident particle induces the generation of hole-electron pairs. The recombination of the hole-electron pairs is generally modeled in SPICE by a double-exponential source current.

A current generator is inserted in the circuit to model the charge collected on an assumed susceptible node as a result if the particle hit. For ASICs, the sensitive nodes can be localized and the critical charge can be estimated from SPICE simulations [4], [5], [9]. The critical charge Qcrit is the minimum charge incident at the memory storage node that causes a change of charge state (a bit flip).

The shape of the generated current is closely approximated by double-exponential source available in SmartSpice. Messenger's equation may be used to take into account this phenomena in SPICE [7]. However, it may be desired to model the data with a PWL source function or other waveforms available in SmartSpice.

Note: Only BSIM3v3 (Level=49, 53), BSIMSOIv3 (Level=57),Modified Gummel Poon (Level=1), Quasi-RC (Level=2) and MEXTRAM (Level=503, 504) models can be impacted by SEU effects.

17.4 Messenger Current Modeling

By default, SmartSpice SEE uses Messenger's fault model to account for errors in the digital circuit due to the impact of incident particles. Messenger's fault model is a double-exponential current source.

17.4.1 Theoretical Expression

In CMOS digital circuits, Single Event Upsets are modeled by injecting the following double exponential current pulse into the affected node [3]:

$$i(t) = I_{SEU} \cdot (\exp(-t/\tau_F) - \exp(-t/\tau_R))$$

with:

- I_{SEU} depends on the amount of injected charge and may be positive or negative.
- τ_F represents the collection time-constant of the junction.
- τ_R represents the ion-track establishment time constant.

The expression in the previous equation can also be expressed using the deposited charge dependence:

$$i(t) = \frac{Q_{dep}}{\tau_F - \tau_R} \cdot (\exp(-t/\tau_F) - \exp(-t/\tau_R))$$

17.4.2 Deposited and Critical Charges

Deposited Charge

The deposited charge is the charge deposited on the impacted node of one device by the generated current source from the impact time to the end of the computation.

This can simply be computed integrating the generated current source, such as:

$$Q_{deposited} = \int_{t0}^{t1} i(t)dt$$

with:

- i(t): current generator.
- t0: begin time when the node is impacted.
- t1: end time of computation.

The relation between the charge deposited Q_{dep} and the Linear Energy Transfer (LET) is given by the following relation:

$$Q_{dep} = \frac{q \cdot \rho}{E_{e,h}} \cdot L_f \cdot LET$$

with:

- q: Electron charge (1.6e-19 C).
- ρ : Material density (2.33 g/cm^3) for silicon.
- $E_{e,h}$: Energy needed to create electron-hole pairs (3.6eV in silicon).
- Lf: Funnel length (m).

Critical Charge

The critical charge is the minimum charge collected to get an upset.

Critical charge leads to evaluate threshold LET. Threshold LET is mainly used to characterized the SEU sensitivity of a circuit. It can be measured with a particle accelerator to give the Cross section LET curves. The goal is the evaluate it by SPICE simulation.

The relation between the critical charge and the threshold Linear Energy Transfer (LET_{th}) is:

$$Q_{crit} = \frac{q \cdot \rho}{E_{e,h}} \cdot L_f \cdot LET_{th}$$

with:

- q: Electron charge (1.6e-19 C).
- ρ : Material density (2.33 g/cm^3) for silicon.
- $E_{e,h}$: Energy needed to create electron-hole pairs (3.6eV in silicon).
- Lf: Funnel length (m).

The methodology to evaluate the critical charge consists of increasing the amplitude of the current until the upset is observed in the simulation. At this point, the charge computed can reveal the critical charge.

17.4.3 Analysis Statement

Ionizing effects like SEU is strongly associated with the transient analysis statement `.TRAN`. For this reason, the `.RAD` statement must be coupled at least with one `.TRAN` statement.

Two syntax forms are proposed. In Syntax form 1, it is necessary to set directly the maximum current `ISEU` of Messenger's equation. In Syntax form 2, it is necessary to set `LET` and `LF`. In this syntax, SmartSpice computes the maximum current using the equation of deposited charge defined above.

Syntax Form 1

```
.RAD SEE<=integer>
+ <START=val0> <val1> ... <valn>
+ <TAUR=val0> <val1> ... <valn>
+ <TAUF=val0> <val1> ... <valn>
+ <DEVICE=string0> <string1> ... <stringn>
+ <NODE=string0> <string1> ... <stringn>
+ <ISEU=val0> <val1> ... <valn>
+ <QCRIT>
+ <ANNOTATE=integer>
```

```
+ <UPSET_TARG=integer>
+ <UPSET_NODE=val0> <val1> ... <valk>
+ <UPSET_THRESHOLD=val0> <val1> ... <valk>
+ <UPSET_ERROR=val0> <val1> ... <valk>
+ <UPSET_DELTATIME=val>
```

SEE	Causes SmartSpice to perform a `.RAD` analysis during simulation. A Multi-ions strike can be performed by setting the number of strikes wanted by the user (`SEE=integer`).
START	Time of the impact.
TAUF	Collection time-constant of the junction.
TAUR	Ion-track establishment time constant.
DEVICE	Device impacted by heavy ion.
NODE	Intrinsic node of the device impacted by the heavy ion (default node: drain).
ISEU	Maximum current (A).
QCRIT	Flag to perform critical charge computation.
ANNOTATE	Degree of annotations (0, 1 or 2). Default is 0.
UPSET_TARG	Define the number of nodes used to detect upsets (k+1).
UPSET_NODE	Name of node used to detect an upset (output node of the circuit).
UPSET_THRESHOLD	Threshold used to detect an upset.
UPSET_ERROR	Error tolerance used to detect an upset.
UPSET_DELTATIME	Duration after impact for the computation of the upset.

Syntax Form 2

```
.RAD SEE<=integer>
+ <START=val0> <val1> ... <valn>
+ <TAUR=val0> <val1> ... <valn>
+ <TAUF=val0> <val1> ... <valn>
+ <DEVICE=string0> <string1> ... <stringn>
+ <NODE=string0> <string1> ... <stringn>
+ <LF=val0> <val1> ... <valn>
+ <LET=val0> <val1> ... <valn>
+ <QCRIT>
+ <ANNOTATE=integer>
+ <UPSET_TARG=integer>
+ <UPSET_NODE=val0> <val1> ... <valk>
+ <UPSET_THRESHOLD=val0> <val1> ... <valk>
+ <UPSET_ERROR=val0> <val1> ... <valk>
+ <UPSET_DELTATIME=val>
```

SEE	Causes SmartSpice to perform a `.RAD` analysis during simulation. A Multi-ions strike can be performed by setting the number of strikes wanted by the user (`SEE=integer`).
START	Time of the impact.
TAUF	Collection time-constant of the junction.
TAUR	Ion-track establishment time constant.
DEVICE	Device impacted by heavy ion.
NODE	Intrinsic node of the device impacted by the heavy ion (default node: drain).
LF	Particle funnel length (m).
LET	Linear Energy Transfer (MeV.cm^2/mg).
QCRIT	Flag to perform Critical charge computation.
ANNOTATE	Degree of annotations (0, 1 or 2). Default is 0.
UPSET_TARG	Define the number of nodes used to detect upsets (k+1).
UPSET_NODE	Name of node used to detect an upset (output node of the circuit).
UPSET_THRESHOLD	Threshold used to detect an upset.
UPSET_ERROR	Error tolerance used to detect an upset.
UPSET_DELTATIME	Duration after impact for the computation of the upset.

Note: Device naming conventions - Devices within subcircuits are identified with Device=M.X* for MOSFETS, and Device=Q.X* for Bipolar transistors.

Messenger's Fault Model Impacting a Transistor

By default, Messenger source is connected to the NODE of the DEVICE set by the user.

TAUF represents the time constant of the collection of the junction (typical values of the order within 100-1000 ps); TAUR accounts the time constant of the ion-track establishment (typical values of the order 1-100 ps).

17.5 User Defined Current Modeling

You have the capability to define the current fault model instead of Messenger's approach.

17.5.1 Analysis Statement

```
.RAD SEE<=integer>
+ <SOURCE=string0> <string1> ... <stringn>
+ <DEVICE=string0> <string1> ... <stringn>
+ <NODE=string0> <string1> ... <stringn>
+ <QCRIT>
+ <ANNOTATE=integer>
+ <UPSET_TARG=integer>
+ <UPSET_NODE=val0> <val1> ... <valk>
+ <UPSET_THRESHOLD=val0> <val1> ... <valk>
+ <UPSET_ERROR=val0> <val1> ... <valk>
+ <UPSET_DELTATIME=val>
```

SEE	Causes SmartSpice to perform a .RAD analysis during simulation.
SOURCE	Voltage source name controlling the ionization level.
DEVICE	Device impacted by heavy ion.
NODE	Intrinsic node of the device impacted by the heavy ion (default node: drain).
QCRIT	Flag to perform Critical charge computation.
ANNOTATE	Degree of annotations. Default is 0.
UPSET_TARG	Defines the number of nodes used to detect upsets (k+1).
UPSET_NODE	Name of node used to detect an upset (output node of the circuit).
UPSET_THRESHOLD	Threshold used to detect an upset.
UPSET_ERROR	Error tolerance used to detect an upset.
UPSET_DELTATIME	Duration after impact for the computation of the upset.

When SOURCE parameter is defined in the transient statement, the current within the source defines the current generated by ion impact.

17.5.2 Control Statement

The expression of the photo-current may be defined using a behavioral device. Voltage source specifications like PWL or EXP, or using Verilog-A may define the waveform of the photo-currents.

Example 1

The subcircuit composed of the voltage source Vcontrol in parallel with the resistance is used to control the dose rate effect. In this example, the photo-current shape can be approximated by double-exponential time dependent current generator described as:

```
Vcontrol  node_1   ground_node    EXP   0   Vmax   Td1    Tau1   Td2
Tau2
Rcontrol  node_1   ground_node    1
```

In this case, `i(Vcontrol)` is the current generated by voltage generator `Vcontrol` within resistance `Rcontrol`. The current `i(Vcontrol)` imposes the waveform of photo-currents generated in p-n junctions of each devices.

Example 2

```
Acontrol  node_1   ground_node   I='my_expression'
Rcontrol  node_1   ground_node    1
```

In the case, the current `i(Aconrol)` defined by the expression `"my_expression"` will be used to compute the different photo-currents.

17.6 Impacted Node Name Convention

`NODE` defines the device node impacted by the fault model. The following possibilities can be selected for:

- `NODE = ndp` for an impact on the intrinsic drain node.
- `NODE = nsp` for an impact on the intrinsic source node.
- `NODE = ng` for an impact on the intrinsic gate node.
- `NODE = nb` for an impact on the intrinsic bulk or body nodes respectively for MOSFET and MOSFET SOI models.

`NODE` defines the device node impacted by the fault model. The following possibilities can be selected for MOSFET and SOI MOSFET:

- `NODE = ndp` for an impact on the intrinsic drain node.
- `NODE = nsp` for an impact on the intrinsic source node.
- `NODE = ng` for an impact on the intrinsic gate node.
- `NODE = nb` for an impact on the intrinsic bulk or body nodes respectively for MOSFET and MOSFET SOI models.

The following possibilities can be selected for BJT models:

- `NODE = ncp` for an impact on the intrinsic collector node.
- `NODE = nbp` for an impact on the intrinsic base node.
- `NODE = nep` for an impact on the intrinsic emitter node.
- `NODE = nsub` for an impact on the substrate node.

The following possibilities can be selected for MEXTRAM models:

- `NODE = nc1` or `nc2` for an impact on `c1` or `c2` intrinsic collector nodes respectively.
- `NODE = nb1` or `nb2` for an impact on `b1` or `b2` intrinsic base nodes respectively.
- `NODE = ne1` for an impact on the intrinsic emitter node.
- `NODE = nsub` for an impact on the substrate node.

Note: The convention for the node names is model-dependent. See the SPICE MODELS MANUAL to get the equivalent circuit of the different models.

By default, the intrinsic drain node is impacted.

17.7 Multi-ions Strike

To perform a Multi-ions strike, the number of strikes must be given by setting a value of the SEE parameter (SEE=n (with n an integer)). In this case, n Messenger's fault models are required for the normal behavior of the circuit. Then the parameters of each Messenger's source must be set and a list of n values must be given for each parameter. For example:

```
.RAD  SEE=3
+ DEVICE =  MNA  MNB MPA
+ TSTART =   10n     11n    15n
+ TAUR =       0.01n  0.05n 0.02n
+ TAUF =       0.1n     0.5n   0.2n
+ LET =          0.2       1.1      3.0
```

17.8 Upset Detection

The user has the capability to define their own criterions to detect an upset.

- UPSET_NODE defines the node name used like reference for the upset detection. Generally, the output of the cell is chosen like reference node.
- UPSET_THRESHOLD defines the minimum or the maximum threshold to detect an upset. At each time step after the impact, the potential of the reference node (defined by the UPSET_NODE) is compared with this obtained at the impact time, and UPSET_THRESHOLD value is defined by the user.

At each time step after the impact:

$$\left|V(ref)_{impact}\right| >= \text{UPSET\_THRESHOLD} > \left|V(ref)\right| \Rightarrow \text{Upset detected}$$

$$\left|V(ref)_{impact}\right| <= \text{UPSET\_THRESHOLD} < \left|V(ref)\right| \Rightarrow \text{Upset detected}$$

UPSET_ERROR is proposed to define a relative tolerance for the upset detection. At each time step after the impact:

$$\frac{\left|V(ref) - V(ref)_{impact}\right|}{MAX(V(ref), V(ref)_{impact})} > \text{UPSET\_ERROR} \Rightarrow \text{Upset detected}$$

UPSET_DELTATIME defines the time duration at during the detection is performed with the parameters defined above: UPSET_NODE, UPSET_THRESHOLD or UPSET_ERROR.

The potentials of different nodes can be watched over too. For this, the number of nodes must be set by UPSET_TARG=k, with k being the number of nodes.

17.9 Important Parsing Limitations

The following notes described the limitations of SEU analysis in SmartSpice.

For parsing, the UPSET_TARG parameter must be set before the lists of UPSET_NODE, UPSET_THRESHOLD or UPSET_ERROR parameters.

To select devices defined in sub-circuits, the prefix "m." or "q." must be added to the device names of .RAD commands respectively for MOSFET/SOI and Bipolars.

Example

```
.RAD SEE=2
+ DEVICE =  M.X0.M1      Q.X1.Q1     M3
+ ...
```

17.10 SEU Detection on SRAM Cell

Single Event effect is accounted by a .RAD statement in the input deck. An example of modification is given below.

```
.TRAN 0.2n 20n
.RAD SEE
+ DEVICE=MN1
+ START=10n TAUF=100p TAUR=5p
+ LF=5u LET=1.5
```

In this example, the impact of a heavy ion with LET=1.5 on the drain node of MN1 is simulated. Output nodes of inverters composing a simple SRAM are presented in Figure 17-1. An upset may be observed after the impact. Figure 17-2 presents the current generated at the drain node of MN1 which provoked the upset.

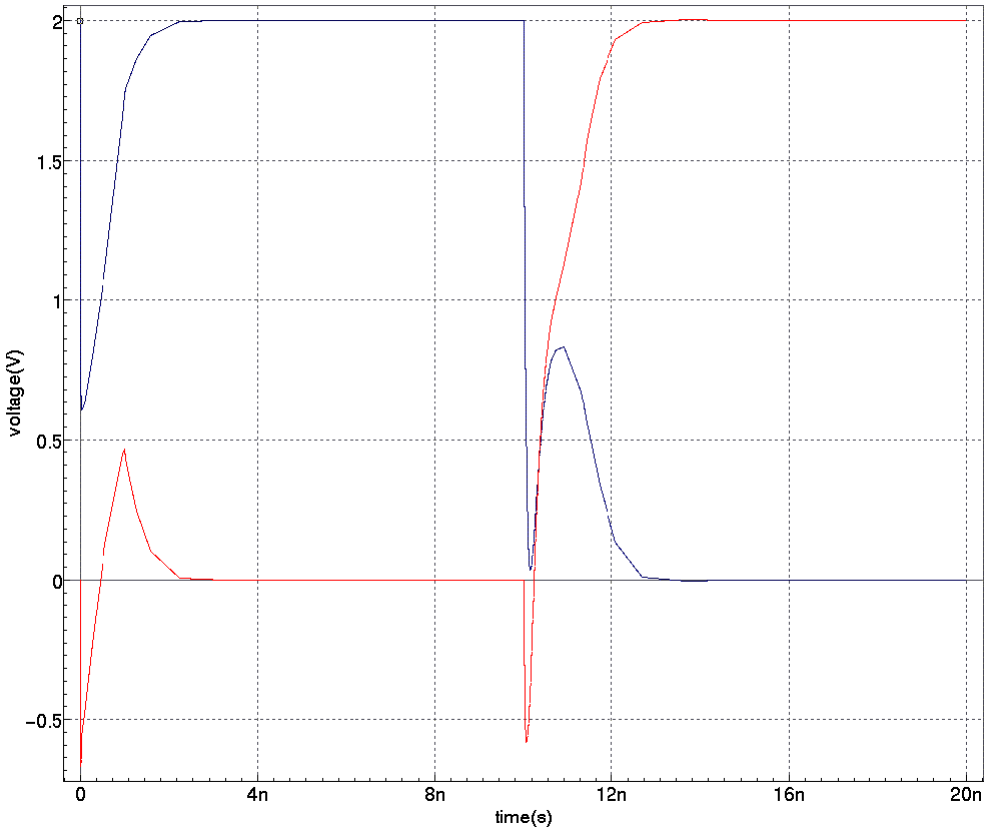


Figure 17-1 Drain potentials of inverter cells

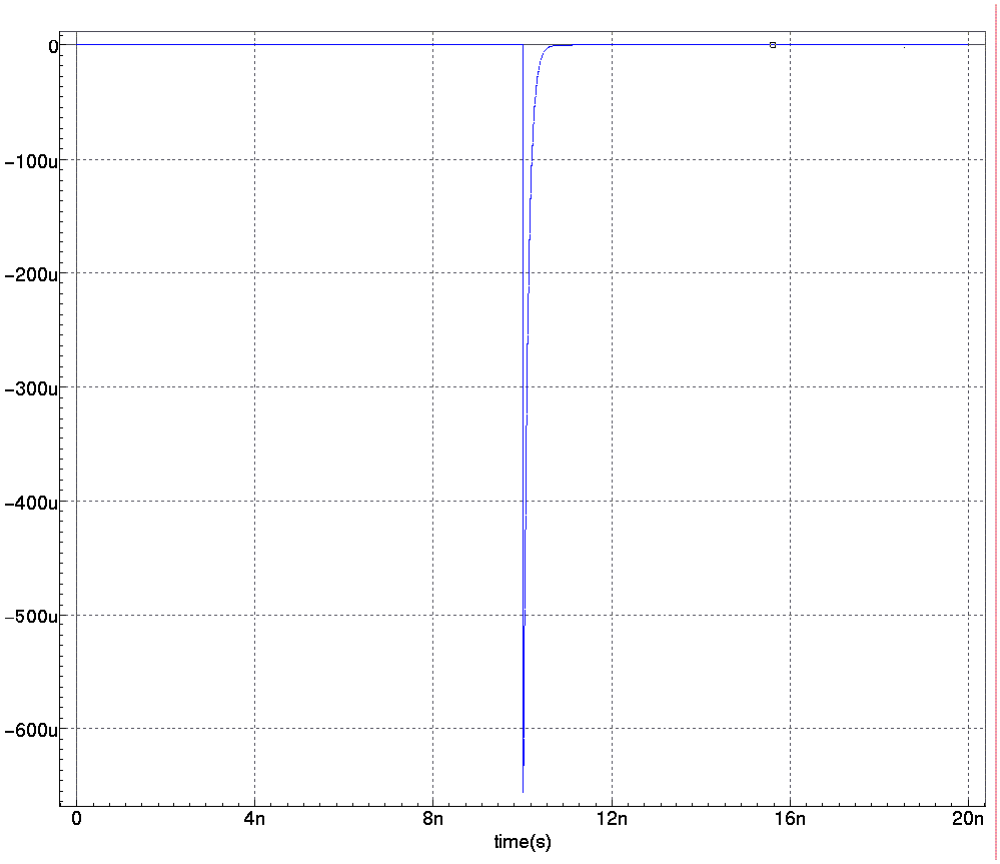


Figure 17-2 Current at the site of the particle strike

17.11 SEE Parameters of the .RAD Statement

The SEE parameters of the .RAD statement can be changed using a .MODIF statement or TRAN sweep.

Example

using a .MODIF statement:

```
.PARAM  DLET=12
.PARAM  COLLECTTIME=200p

* .TRAN 1n 25u  SWEEP DLET LIST 4 12  10  8  6
.TRAN 1n 25u
.RAD SEE=1
+ DEVICE=M.X8.M2
+ START=16u TAUF=COLLECTTIME TAUR=5p
+ LF=3u LET= DLET
+ ANNOTATE=2

.MODIF LOOP=20
+ COLLECTTIME = AGAUSS(200p 20p 3) PRTBL   PRMC
```

or

```
.PARAM  DLET=4
.TRAN 1n 25u
.RAD SEE=1
+ DEVICE=M.X8.M2
+ START=16u TAUF=200p TAUR=5p
+ LF=3u LET= DLET
+ ANNOTATE=2

.MODIF DLET=4
+MODIF DLET=8
+MODIF DLET=10
+MODIF DLET=12
```

Example

using a TRAN sweep:

```
.PARAM AS=100p

.TRAN 1n 25u SWEEP As LIST 5 100p 125p 150p 200p 250p
.RAD SEE
+ DEVICE=M1 TAUR=75p TAUF='AS'
+ START=10u
+ LF=2u LET=5
```

17.12 All Supported Methods for Defining an SEU Pulse

```
*
* TEST Circuit 4X1SRAM

*****Call   to   include   the   Verilog_A   Module   defining   a
Pulse**********
.HDL ISEU.VA
*
C1 BL GND 10f
C2 BL_ GND 10f
M1 Q_ Q GND GND NFET L=0.18u W=0.3u AD=0.12p AS=0.15p PD=1u PS=1u
M2 Q Q_ GND GND NFET L=0.18u W=0.3u AD=0.12p AS=0.15p PD=1u PS=1u
M3  Q  Q_  PWR  PWR  PFET  L=0.2u  W=0.25u  AD=0.15p  AS=0.12p  PD=1.2u
PS=1.2u
M4  Q_  Q  PWR  PWR  PFET  L=0.2u  W=0.25u  AD=0.15p  AS=0.12p  PD=1.2u
PS=1.2u
M5  BL_  WL  Q_  GND  NFET  L=0.27u  W=0.22u  AD=0.2p  AS=0.2p  PD=1.2u
PS=1.2u
M6 BL WL Q GND NFET L=0.27u W=0.22u AD=0.2p AS=0.2p PD=1.2u PS=1.2u
V1 WL GND DC 1.8
VVDD PWR GND DC 1.8
*
* End of the netlist

*
* Markers to save
*
.SAVE V(BL) V(BL_) @M2[iseu] all(v) all(i)

*************************************************************
.LIB 'fet.lib' NOM

*****************************Initial
Condition*****************************
.IC V(Q)=1.8      V(Q_)=0

.TRAN 1p 1u

*********************************Ion
Strike*********************************
**********************User        defined        Pulse        with
Verilog_A*******************
.RAD SEE = 1
+ DEVICE = M2
+ SOURCE =XSEU
+ NODE = NDP
*****************************
XSEU POS 0 VSEU
RSEU POS 0 1
*************************************************************

#COM

******************Other    methods    that    a    pulse    can    be
defined***************
```

```
*******************Messenger**********************************
.PARAM HIT_RISE=25p
.PARAM COLLECTION_TIME = 100p
.RAD SEE=1
+ DEVICE=M2
+ TAUR='HIT_RISE'
+ TAUF='COLLECTION_TIME'
+ START=300n
+ LF=1u
+ LET=5
****************************************************************

****************************ISEU                                Max
Defined*****************************
.PARAM HIT_RISE=25p
.PARAM COLLECTION_TIME = 100p
.RAD SEE=1
+ DEVICE=M2
+ TAUR='HIT_RISE'
+ TAUF='COLLECTION_TIME'
+ START=300n
+ ISEU=1E-3
****************************************************************

**************************             User              Defined
Pulse*****************************
.RAD SEE = 1
+ DEVICE = M2
+ SOURCE = VSEU
+ NODE = NDP

VSEU POS 0     EXP  0  1m 30n 25p  30.1n  100p
RSEU POS 0 1
****************************************************************

************************      User      Defined      Data      File
Pulse*******************
.RAD SEE = 1
+ DEVICE = M2
+ SOURCE = VSEU
+ NODE = NDP

VSEU POS  0    PWLFILE SEU_PULSE.DATA
RSEU POS 0 1
****************************************************************

************************       User       Defined       Behavioral
Pulse*********************
.PARAM TAUR =25p
.PARAM TAUF=100p
.PARAM HIT_TIME=300n
.RAD SEE = 1
+ DEVICE = M2
+ SOURCE = ASEU
+ NODE = NDP
```

```
ASEU  POS  0   V= 'IF (TIME >'HIT_TIME')
+ THEN 1m*(EXP(-(TIME-'HIT_TIME')/'TAUF')- EXP(-(TIME-'HIT_TIME')/
'TAUR'))
+ ELSE 0.0'
RSEU  POS  0   1
*************************************************************

**********************User             defined             Pulse
Verilog_A**********************
.RAD SEE = 1
+ DEVICE = M2
+ SOURCE =XSEU
+ NODE = NDP
*****************************
XSEU POS 0 VSEU
RSEU POS 0 1
*************************************************************

#ENDCOM

.END
```

References

1. M. Baze, “A digital CMOS design technique for SEU hardening”, *IEEE Transactions on Nuclear Science*, vol. 47, n.6 Dec. 2000.
2. K. Clark, “Modeling Single-Event Effects in a Complex Digital Device“, *IEEE Transactions on Nuclear Science*, vol.50, n.6, Dec. 2003.
3. A. Dharchoudhury, “Fast timing simulation of transient faults in digital circuits”, Proc. Intl. Conf. on computer Aided design, pp. 719-726, 1994.
4. P.E. Dodd, F.W. Sexton et al., “Impact of technology trends on SEU in CMOS SRAMs”, *IEEE Transactions on Nuclear Science*, vol. 43, n.6, Dec. 1996.
5. F. Faccio et al., “Single Event Effects in Static and Dynamic Registers in 0.25um CMOS technology”, *IEEE Transaction on Nuclear Science*, vol. 46, n.6, Dec. 1999.
6. V. Ferlet-Cavrois et al., “Characterization of the Parasitic Bipolar Amplification in SOI technologies submitted to transient irradiation”, *IEEE Transaction on Nuclear Science*, Vol.49, n. 3, June 2002.
7. G. Messenger, M. Milton, “Single Event Phenomena”, Chapman & Hall editor, 1997.
8. C. Nicklaw “Multi-Level Modeling of total ionizing dose in SiO2: first principles to circuits”, thesis, Vanderbuilt University, Tennessee, USA, 2003.
9. Ph. Roche et al., “Determination of key parameters for SEU occurrence using 3-D full cell SRAM simulations”, *IEEE Transaction on Nuclear Science*, vol.46, n.6, Dec. 1999.
10. D. Van Nort “TCAD driven dose-rate photocurrent spice models for partially depleted SOI MOSFETS”, thesis, Vanderbilt university, Tennessee, USA, 2002.

Chapter 18
Optimizer

18.1 Introduction

The Optimizer is a fully integrated feature of SmartSpice that performs parametric optimization of circuits. At the core of the SmartSpice Optimizer, is a general purpose optimizing engine that requires initial and target parameter values to be set. The Optimizer then iterates these parameters until the target values are reached. The Optimizer provides a comprehensive interface and an interactive display system for visualizing the optimization process as it is executed.

There are two types of optimization: Performance Measure Optimization and Function Optimization. In performance measure optimization, the values and parameters of circuit components are altered so that individual electrical characteristics meet specifications. In function optimization, values and parameters are altered so that the overall function of the circuit meets specifications.

Performance Measure Optimization

Performance measure optimization automatically changes device and model parameters in order to match the user-defined electrical characteristics of the circuit. The Optimizer can fine-tune delay, rise/fall times, trip point, maximum and minimum current, and any other circuit performance measurement that can be calculated by a SmartSpice `.MEASURE` statement. Simultaneous multi-target optimization of several performance measures is also possible.

Function Optimization

Function optimization is based on the same mathematical methods as performance measure optimization. Function optimization matches calculated curves with desired curves for DC, AC, and transient analyses. The desired curves can represent the results of theoretical research or physical measurement, or they can be output curves of a device simulator such as S-PISCES.

There are no restrictions on the type of circuit analysis that can be performed. Circuits can be optimized in steady state, frequency, and time domain.

18.2 Optimizer Syntax

To perform optimization in SmartSpice, the `.MODIF` statement must be used. For a circuit with a known topology, the `.MODIF` statement allows you to:

- Solve a separate optimization problem.
- Sequentially solve several optimization problems.
- Sequentially solve optimization problems and perform parametric analysis.

Syntax

```
.MODIF <PROFF> <PRTBL> <RESTORE> <constpar_spfc>
+ OPTIMIZE param_spfc
+ TARGETS targ_spfc
+ OPTIONS <opt_spfc> <<modif2_spfc> ... >
```

PROFF	This flag suppresses online printing of `.MEASURE` statement results.
PRTBL	This flag causes SmartSpice to print the final table of parameter and measure values obtained from all iterations.

RESTORE	This keyword causes SmartSpice to restore all modified parameters in the specified set to their original values.
constpar_spfc	Definitions for all parameters that will be held constant through the optimization process.
OPTIMIZE	This keyword is followed by the list of optimization parameters.
param_spf	Definitions for all parameters that will be changed during optimization. For each parameter a minimum, maximum, and initial value must be specified.
TARGETS	This keyword is followed by the list of targets for optimization.
targ_spf	Definition of all targets that will be optimized. For each target, the name of a measured and a desired value must be specified.
OPTIONS	This keyword is followed by the list of optimization control options.
opt_spfc	Optimizer control options. You can specify an option and value to replace a default option.
modif2_spfc	This definition follows the keyword `MODIF`. The procedure defined here is called immediately after the final iteration of the current optimization. The procedure could be another optimization or a parametric analysis. In either case, the circuit parameters correspond to those just calculated by the current optimization.

18.2.1 Input Deck

A typical input deck created for optimization contains:

- One or more analysis statements to simulate a circuit in either steady state, frequency, or time domain.
- One or more `.MEASURE` statements to calculate performance measurements or differences between simulated and desired curves.
- One or more `.DATA` statements to describe desired curves for function optimization.
- One or more `.PARAM` statements to define parameter labels if needed.
- A `.MODIF` statement to define the names of device and model parameters; their minimum, maximum and initial values; targets; and control options for optimization.
- A `.IPLOT` statement to plot output variables while the optimization is in progress.

In order to match circuit specifications, the Optimizer performs a number of iterations, beginning with the user-defined initial values of parameters. On each iteration, the Optimizer simulates the circuit, calculates target electrical specifications, updates parameters using defined mathematical strategies, and simulates the circuit again. The Optimizer will stop the optimization process when one or more of the stop criteria specified in the Optimizer control statements is satisfied.

If SmartSpice is running in window mode, the intermediate results of the optimization process are immediately visible. SmartSpice allows you to observe the history of parameter, and target values obtained from all iterations. The **Stop** button can be used to stop the

optimization process on any iteration (to change parameter values, targets, or control options) and continue the optimization process from the last parameter values calculated.

18.2.2 Inverter Example

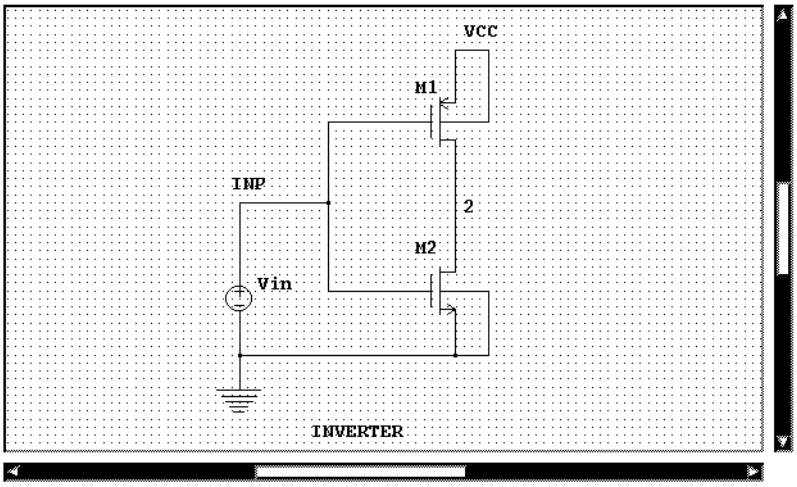


Figure 18-1 Example Inverter Circuit

For the inverter shown in Figure 18-1, the input voltage `vin` is applied to the gates of `m1` and `m2`, and the output of the circuit is at `node2`. The voltage at `node2` will be high when `vin` is low, and the voltage at `node2` will be low when `vin` is high. The value of `vin`, when the output switches from high to low or from low to high, is known as the trip point.

This example shows how to use the Optimizer to select the transistor widths of `m1` and `m2` so that the trip point can be set at 1.55V. In addition, the maximum current through the inverter is set at 1.9mA. The trip point and the maximum current values are known as the optimization targets. The widths of the transistors are the circuit parameters that will be varied until the target values are reached. The circuit parameters that are varied in an Optimizer session are known as optimization parameters.

Examples

```
Inverter DC Optimization
***** Circuit
vcc 0 vss dc -5v
vin inp 0 dc 1.55v
m1 2 inp vss vss pm w=wp l=lp
m2 2 inp 0 0 nm w=20u l=2.0u
***** Models
.MODEL pm PMOS (level=3 tox=.02e-6)
+ phi=0.576 gamma=0 vto=0 alpha=0 kappa=0)
.MODEL nm NMOS (level=3 tox=.02e-6)
+ phi=0.576 gamma=0 vto=0 alpha=0 kappa=0)
***** Analysis statement
.DC vin 0 5 0.1
***** Measure statements
.MEASURE DC maxcur MAX i(vcc)
.MEASURE DC trip CROSS v(2) v(inp)
***** Parameter labels
.PARAM wp=10u lp=1.6u
***** Optimization specifications
.MODIF
```

```
+ OPTIMIZE
+ wp=OPT(2.0e-7 30e-6 10e-6)
+ m2(w)=OPT(5u 100u 20u)
+ TARGETS trip=1.55 maxcur=0.0019
+ OPTIONS MEASOFF=0 AVG=0.001
***** Interactive plot
.IPLOT v(inp) v(2)
.OPTIONS iplot_one
.END
```

In this input file, the optimization parameters `wp` and `m2(w)` follow the keyword `OPTIMIZE` in the `.MODIF` statement. The parameter label `wp` is defined in the `.PARAM` statement and is used in the `m1` transistor description. The full path name `m2(w)` contains the name of the device `m2` and the name of the parameter `w` in parentheses. For both parameters, the minimum, maximum, and initial values are specified.

The target specifications follow the keyword `TARGETS`. The `trip` and `maxcur` measures are calculated by `.MEASURE` statements. The trip point is calculated as a cross point of the input and output voltages `v(inp)` and `v(2)`. The measurement `maxcur` is defined as the maximum value of `i(vcc)`.

The Optimizer control option AVG follows the keyword `OPTIONS`. AVG defines the minimum average error as 0.001. The default value is 0.01. Optimization results are shown in the following tables. The optimization process stopped when the average relative error was less than 0.001.

Parameters

Name	Final Value	Init. Value
wp	4.93385e-06	1.00000e-05
m2(w)	3.26027e-05	2.00000e-05

TARGETS

Name	Final Value	Init. Value
trip	1.54940e+00	2.22659e+00
maxcur	1.89982e-03	2.50698e-03

```
Number of iterations     =  6

Number of func evals     =  13

Number of Jac evals      =  3


Residual sum of          =  4.685e-04
squares

Norm of the gradient     =  2.708e-02

Marquardt parameter      =  1.193e-03


RMS relative error       =  2.818e-04

Average relative error   =  2.409e-04

Maximum relative error   =  3.871e-04


Termination code         =  7 : RMS criterion
```

The SmartSpice Optimizer used the initial values of `wp` and `m2(w)` and simulated the circuit to calculate the initial values for the trip point and the maximum current. Six iterations were performed where the values of `wp` and `m2(w)` were varied. The optimization process stopped because the average relative error reached the value specified in the `.OPTIONS` statement. From the optimization results, it can be seen that the desired trip voltage of 1.55V has been achieved and that the maximum current through the inverter is just under 1.9mA.

The widths of the transistors `m1` and `m2` have been optimized to 4.9µm and 32.6µm, respectively.

18.2.3 Parameters

Each parameter included in the .MODIF statement is specified in the form:

```
parname = rhs
```

parname	Name of the parameter
rhs	Value of the parameter

Each of these is described below.

Parameter Name

The parameter name depends on the parameter type and can be specified in one of the following forms:

```
devname(parname)
```

devname	User-specified device name
parname	Name of a parameter of the device devname.

or

```
modname(parname)
```

modname	User-specified model name.
parname	Name of a parameter of the model modname or for all devices of the type devtype:

```
ALL@devtype(parname)
ALL@devtype
```

devtype	Prefix that defines a device type.
parname	Name of a device parameter.

or

```
parlabel | TEMP | GMIN | RELTOL | ABSTOL | VNTOL
```

parlabel	User-specified parameter label in the .PARAM statement.
TEMP	Temperature
GMIN	Model and computational parameter
RELTOL	Relative tolerance
ABSTOL	Absolute current error tolerance
VNTOL	Absolute voltage error tolerance

Any model and device parameter can be specified in one of the following ways:

- The full-path name that consists of the model or device name and the name of the parameter.
- A parameter label defined in the .PARAM statement and referred to in the model or device specification.

Each of these methods offers certain advantages.

Full Path

The advantages of the full-path name are:

- It allows the use of an input netlist without any changes in the circuit description.
- It provides access to any model or device when subcircuits are used.

Transient analysis voltage and current source parameters have notations but do not have names. For example, V1, V2, td, tr, tf, pw and per are notations for initial value, pulsed value, delay time, rise time, fall time, pulse width, and period of a pulse waveform. These notations can not be used in full-path names. The names PAR1, PAR2, ..., PAR7 can be used in place of these notations. For example, the name vin (PAR3) can have any of the following meanings:

- Rise delay time for an exponential waveform
- Delay time for a pulse waveform
- Carrier frequency for a single frequency waveform
- Frequency for a sinusoidal waveform

Parameter Label

The advantages of the parameter label method are:

- The same value can be set or updated for different models and devices.
- Arbitrary dependencies between different device and model parameter can be introduced by parameter expressions in the .PARAM statement.
- Labels can be used in the .MEASURE statement for most parameters.

If the value of a model or device parameter is loaded by means of a parameter label defined in the .PARAM statement, then the full-path name of this parameter can not be used in a .MODIF statement. In the example above, the width w of the transistor m1 is loaded by means of the parameter label wp defined in the .PARAM statement. Because of this, the full-path name m1(w) can not be used in a .MODIF statement.

A parameter label can not be used in a .MODIF statement if:

- A parameter label is not defined in a .PARAM statement.
- A parameter label is defined as a function of another parameter label.

18.2.4 Parameter Value

For constant parameters, the parameter value rhs can be a number or the name of a measure calculated by the .MEASURE statement. When a parameter is defined as a parameter for optimization, its value rhs must be specified as follows:

```
OPT (minval maxval <initval>)
```

OPT	Mandatory keyword followed by minval, maxval, and initval in parentheses.
minval	Minimum limit for the parameter to be changed.
maxval	Maximum limit for the parameter to be changed. During optimization the parameter can be changed within the limits of the values of minval and maxval.
initval	Initial value of the parameter to be changed. If initval is not specified, the Optimizer uses a value of the parameter specified in the input deck or in the .PARAM statement. In either case, the initial value must not be equal to zero.

In the example above, the minval, maxval and initial of the parameter wp are 2e-7, 30e-6 and 10e-6, respectively.

18.2.5 Targets

Each target specified in the .MODIF statement after the keyword TARGETS must be specified in the form:

```
measname = goalval
```

measname	Name of the measure calculated by a .MEASURE statement. It can be a circuit performance measure like delay, rise/fall time, maximum/minimum value, or a difference between a desired and a simulated curve calculated by the ERR .MEASURE statement.
goalval	Goal value. The SmartSpice Optimizer varies parameters in order to minimize the difference between simulated and desired values of the measure. The goal value must not be equal to zero.

In the example above, the goal values for the measures trip and maxcur are 1.55 and 0.0019, respectively.

18.3 Control Options

Optimizer control parameters follow the keyword OPTIONS. One or more control options can be specified to replace default options. The control options define the maximum numbers of iterations and function evaluations, convergence criteria, and control flags for interactive plotting and printing.

The following control options can be specified:

AVG	Minimum value for the average relative error. The Optimizer will terminate the optimization process when the average relative error is less than AVG. Default value is 0.01.
DELTA	Minimum value for the gradient. The Optimizer will terminate the optimization process when a norm of the gradient is less than DELTA. Default value is 1e-6.
EPS	Minimum change of the sum of squares. The Optimizer will terminate the optimization process when the relative change in the reduction of the sum of squares is less than EPS for two successive iterations. Default value is 1e-11.
FC	The FC parameter controls how the Optimizer computes derivatives. Under normal circumstances the Optimizer uses a combination of forward and backward derivatives to determine the slope at a given point in parameter space. The value of these parameters must lie between 0 and 1 (values outside this range will generate a warning during parsing). A value of 0 means that the Optimizer will use only backward derivatives, while a value of 1 means only forward derivatives are used. The default value is 0.2.
GNORM	The gradient norm is a threshold used to indicate when the optimization process is declared insensitive to a parameter. The value for GNORM is generally small. The Optimizer calculates the slope of the parameter space with respect to all parameters in the optimization set. Those parameters with partial derivatives below the GNORM threshold are considered insensitive. If all parameters fall below the threshold, the optimization terminates. The default value is 1e-18.
MARQF	Marquardt scale factor. Default value is 2.0.
MARQP	Marquardt parameter initial value. Default value is 0.1.
MARQUP	Marquardt parameter upper bound. The Optimizer will terminate the optimization process when the Marquardt parameter is greater than MARQUP. Default value is 1000.
MAXERR	Minimum value for the maximum relative error. The Optimizer will terminate the optimization process when the maximum relative error is less than MAXERR. Default value is 0.01.

MEASOFF	Control flag for printing .MEASURE statement results: • 0 - suppresses printing • 1 - prints results for the first and final iterations • 2 - prints results for all iterations • 3 - prints results for all function evaluations Default value is 2.
NUMFUNC	Maximum number of function evaluations. Each function evaluation includes such computational operations as circuit simulation, target calculations, and a new set of parameter computations. Typically, the actual number of function evaluations is greater than the number of iterations, this is because on some iterations the Optimizer calculates the Jacobian matrix.
NUMITER	Maximum number of iterations. The Optimizer will terminate the optimization process when the number of iterations reaches NUMITER. Default value is 15.
NUMJAC	Maximum number of Jacobian evaluations.
IPLOFF	Control flag for interactive plotting of output variables specified in a .IPLOT statement: • 0 - suppresses plotting • 1 - plots variables for the first and final iterations • 2 - plots variables for all iterations • 3 - plots variables for all function evaluations Default value is 2.
RMS	Minimum value for root mean square error. The Optimizer will terminate the optimization process when the root mean square error is less than RMS. Default value is 0.01.

Termination Criteria

Upon termination, the Optimizer will report a **Termination code**, which indicates which of the control options caused the optimization to finish (see Section 18.3“Control Options”). It can take one of the values shown in Table 18-1.

Table 18-1 Optimizer Termination Codes

Termination Code	Name
1	EPS criterion
2	DELTA criterion
3	BOUND criterion
4	MARQ_PAR criterion
5	NUM_FUN_EVALS criterion

Table 18-1 Optimizer Termination Codes

Termination Code	Name
6	NUM_JAC_EVALS criterion
7	RMS criterion
8	AVGerr criterion
9	MAXerr criterion
10	NUM_ITER criterion

18.4 Optimizer Examples

18.4.1 Delay Optimization Example

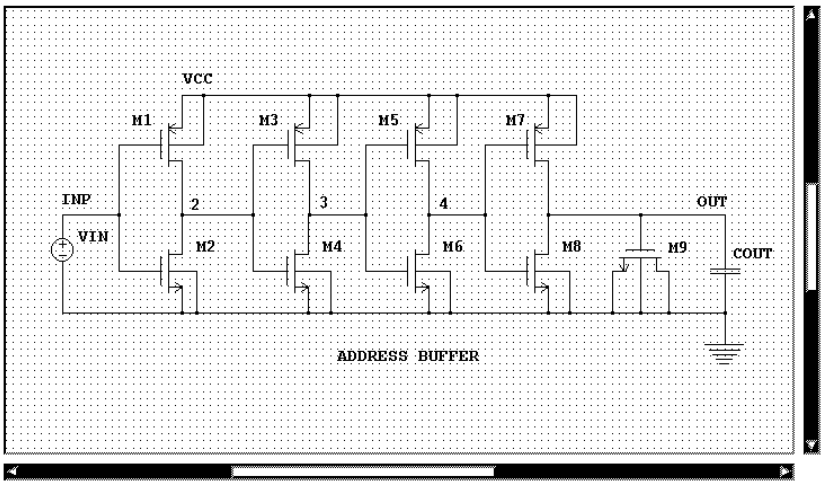


Figure 18-2 Address Buffer Circuit

For the circuit shown in Figure 18-2, the optimization targets at the temperature 75^{o}C are:

- Rise delay of 3ns
- Fall delay of 3ns

The parameters for optimization are the widths of transistors: `m3`, `m4`, `m5`, `m6`, `m7` and `m8`.

In the input file, the `.MODIF` statement sets the temperature to a value of 75^{o}C, and the power supply voltage `vccdc` to 4.5V. The `.MODIF` statement then performs the performance measure optimization. The rise time `del_r` and the fall time `del_f` are calculated using the `.MEASURE` statements. The rise time is defined as the time difference between the mid-point of the input signal's rising edge and the corresponding point of the output signal. The fall time is calculated from the mid-points of the falling edges.

Examples

```
Address Buffer
* Optimization of rise and fall delays
***** Circuit
vin inp 0 DC 1.5 PULSE(0.4 2.6 0 5N 5N 15N)
vcc vss 0 DC vccdc
m1 2   inp vss vss pm W=5.067U  L=1.6U
m2 2   inp 0   0   nm W=34.82U  L=2.0U
m3 3   2   vss vss pm W= 50u    L=1.1U
m4 3   2   0   0   nm W=70.U    L=1.0U
m5 4   3   vss vss pm W=320.U   L=1.1U
m6 4   3   0   0   nm W=440.U   L=1.1U
m7 out 4   vss vss pm W=parm7w  L=1.1U
m8 out 4   0   0   nm W=parm8w  L=1.1U
m9 0   out 0   0   nm W=380U    L=1.2U
COUT out 0 700p
***** Models
.MODEL nm NMOS (LEVEL = 3 TOX=.02e-6)
.MODEL pm PMOS (LEVEL = 3 TOX=.02e-6)
***** Analysis statement
.TRAN 0.2NS 40NS CALLV SAVEV
***** Parameter labels
.PARAM parm7w=500u parm8w=500u vccdc=4.5
***** Measure statements
.MEASURE TRAN del_R DELAY v(inp) RISE=1 VAL=1.5
+ TARG=v(out) RISE=1 VAL= ‘vccdc/2’
.MEASURE TRAN del_F DELAY v(inp) FALL=1 VAL=1.5
+ TARG=v(out) FALL=1 VAL= ’vccdc/2’
***** Optimization specifications
.MODIF   TEMP= 75    vccdc=4.5
+ OPTIMIZE parm7w=opt(10u 3000u 500U)
+ parm8w=opt(10u 3000u 500U)
+ m3(w)=opt(10u 300u 50U)
+ m4(w)=opt(10u 300u 70U)
+ m5(w)=opt(100u 1000u 320U)
+ m6(w)=opt(50u 1000u 440U)
+ TARGETS del_f=3ns del_r=3ns
+ OPTIONS AVG=0.001 MEASOFF=0
***** Interactive plot
.OPTIONS IPLOT_ONE
.IPLOT v(out) v(inp)
.END
```

The optimization results are shown in the following tables.

Parameters

Name		Final Value	Init. Value
parm7w	=	8.57060e-04	5.00000e-04
parm8w	=	1.24485e-03	5.00000e-04
m3(w)	=	3.12675e-05	5.00000e-05
m4(w	=	5.27244e-05	7.00000e-05
m5(w)	=	2.49000e-04	3.20000e-04
m6(w)	=	2.08049e-04	4.40000e-04

TARGETS

Name		Final Value	Init. Value
del_f	=	3.02174e-09	5.75936e-09
del_r	=	3.00062e-09	4.80122e-09

```
 Number of iterations          =  8

 Number of func evals          =  21

 Number of Jac evals           =  2

 Residual sum of squares       =  2.821e-04

 Norm of the gradient          =  2.165e-02

 Marquardt parameter           =  1.000e-05


 RMS relative error            =  5.126e-03

 Average relative error        =  3.727e-03

 Maximum relative error        =  7.247e-03


 Termination code              =  7 :  RMS criterion
```

A delay optimization plot is shown in Figure 18-3.

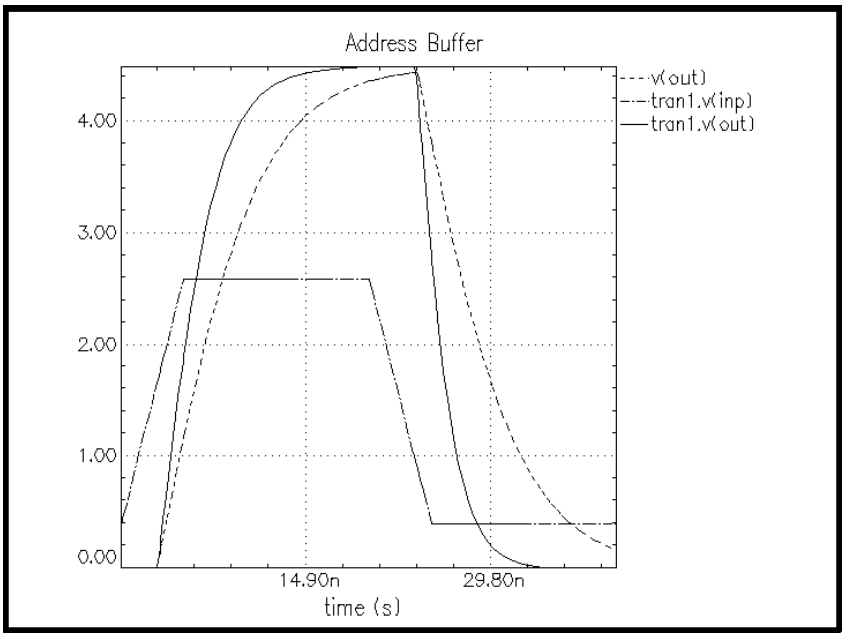


Figure 18-3 Delay Optimization Plot

18.4.2 MOSFET Optimization Example

In this example, the Optimizer adjusts MOSFET Level 3 parameters to match a simulated drain to source DC current with measured data from a physical device.

The measured values are defined in the .DATA statement. This statement contains ids versus vds curves for five vgs values from 1V to 5V in increments of 1V. For each of the five vgs values, the voltage vds is swept from 0 to 5 volts in steps of 0.2V. Therefore, there are 130 values in the .DATA statement. For brevity, only the first two and last two values are shown.

The difference differ between measured and simulated curves is calculated by an ERR .MEASURE statement. The NEST=-1 parameter causes SmartSpice to calculate the absolute relative error for all curves.

The optimization parameters are: RS, RD, LD, NSUB, VTO, UO, VMAX, XJ, ETA and THETA.

The control parameter NUMITER defines the maximum number of iterations. The control options IPLOFF and MEASOFF cause SmartSpice to print and plot results from the first and final iterations.

```
** MOSFET Level 3
* i(vds) Optimization
* Circuit
vgs  2 0
vds  1 3
vdss 0 3 DC 0
vbs  4 0 0
m1 1 2 0 4 nm W= 5E-05 L= 2E-06 NRD= 4E-02 NRS= 4E-02

***** MOSFET model definition
.MODEL nm NMOS (TOX = 5E-8
+ RS = 2 RD = 2 LD = 1E-7
+ WD = 0 NSUB = 1E16 VTO = 0.8
+ UO = 626 VMAX = 2e6 XJ = 3E-7
+ ETA = 0.02 THETA = 0.03 NFS = 3E11
+ IS = 1.78E-16 CJ = 4.550487E-5 PB = 0.6172644
```

```
+ MJ = 0.4086356 FC = 0.5 CJSW = 1.876352E-10
+ MJSW = 0.4232647 CGSO = 1E-9 CGDO = 1E-9
+ CGBO = 1E-9 LEVEL = 3)
 ***** Measured curves
 .DATA i_vds_data
 + user_vds targ
 *
 * corresponds to vgs=1.0
 *
 * 0     1e-25
 +
 +       . . .
 + 5.0   1.234e-7
 *
 * corresponds to vgs=2.0
 *
 + 0      0
 +        . . .
 + 5.0    2.345e-7
 *
 *
 * corresponds to vgs=3.0
 *
 + 0     2e-25
 +       . . .
 + 5.0   3.333e-8
 *
 * corresponds to vgs=4.0
 *
 + 0  0
 +       . . .
 + 5.0   0.123e-7

 *
 * corresponds to vgs=5.0
 *
 + 0     2e-18
 +       . . .
 + 5.0   2.2323e-7

 .DC vds 0 5.0 0.2 vgs 1.0 5.0 1.0
 ***Difference between measured and simulated curves
 .MEAS DC differr ERR2 targ i(vdss)
 + VAL=1.e-4 NEST=-1
 ***** Optimization specifications
 .MODIF
+ OPTIMIZE
+ nm(RS)   = OPT(0.01 100 2)
+ nm(RD)   = OPT(0.01 100 2)
+ nm(LD)   = OPT(1e-9 2e-6 1e-7)
+ nm(NSUB) = OPT(1e15 1e17 1e16)
+ nm(VTO)  = OPT(0.333 2.0 0.8)
+ nm(UO)   = OPT(100 800 626)
+ nm(VMAX) = OPT(1e4 1e7 2e6)
+ nm(XJ)   = OPT(2e-7 3e-6 3e-7)
+ nm(ETA)  = OPT(1e-4 5.0 0.02)
```

```
+ nm(THETA)= OPT(1.e-4 1 0.03)
+ TARGETS differr=0.01
+ OPTIONS NUMITER=25 IPLOFF=1 MEASOFF=1
 .IPLOT i(vdss) targ

 ***** Interactive plot
 .OPTIONS IPLOT_ONE
 .END
```

The optimization results are shown in the following tables.

Parameters

Name		Final Value	Init. Value
nm(rs)	=	1.81317e+00	2.00000e+00
nm(rd)	=	1.66983e+00	2.00000e+00
nm(ld)	=	9.98697e-08	1.00000e-07
nm(nsub)	=	9.06585e+15	1.00000e+16
nm(vto)	=	1.17378e+00	8.00000e-01
nm(uo)	=	6.25184e+02	6.26000e+02
nm(vmax)	=	2.18683e+06	2.00000e+06
nm(xj)	=	2.00000e-07	3.00000e-07
nm(eta)	=	1.25499e-02	2.00000e-02
nm(theta)	=	2.71976e-02	3.00000e-02

TARGETS

Name		Final Value	Init. Value
differr	=	1.00248e-02	4.10632e-01

```
 Number of iterations     =   23
 Number of func evals     =   85
 Number of Jac evals      =    6

 Residual sum of squares  =   1.859e-02
 Norm of the gradient     =   2.130e+01
```

```
Marquardt parameter      =   1.137e-04


RMS relative error       =   2.480e-03
Average relative error   =   2.480e-03
Maximum relative error   =   2.480e-03


Termination code         =   7 : RMS criterion
```

A MOSFET level 3 optimization plot is shown in Figure 18-4.

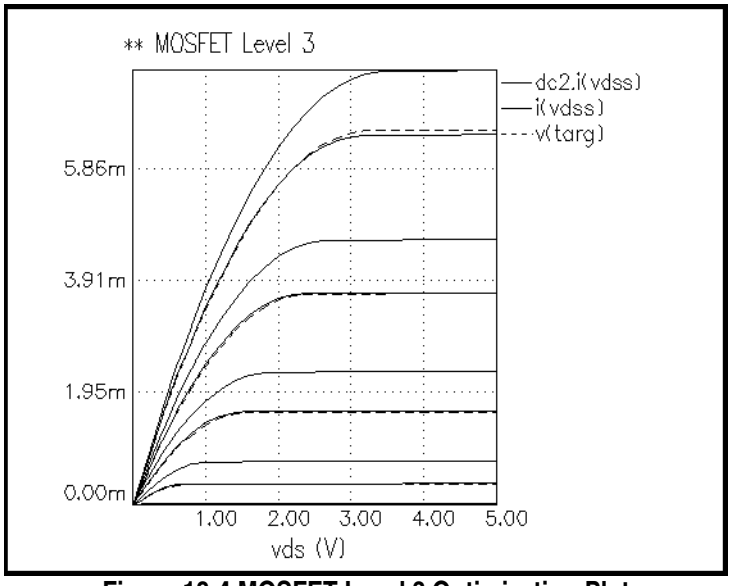


Figure 18-4 MOSFET Level 3 Optimization Plot

Device Parameter Specification For Devices In A Subcircuit

```
.MODIF OPTIMIZE
+X8.X1.m(w) = OPT(1u 20u 4u)
+X8.X1.m(l) = OPT(1u 20u 4u)
+X8.X2.m(w) = OPT(1u 20u 4u)
+X8.X2.m(l) = OPT(1u 20u 4u)
+TARGETS differr=0.01 del_F=20NS del_R=20NS
```

The above specification of the device parameters `X8.X1.m(w) , ...` will be expanded in:

```
.MODIF OPTIMIZE
+ m.x8.x1.mn3(w) = OPT(1u 20u 4u)
+ m.x8.x1.mn3(l) = OPT(1u 20u 4u)
+ m.x8.x2.mn3(w) = OPT(1u 20u 4u)
+ m.x8.x2.mn3(l) = OPT(1u 20u 4u)
+ m.x8.x1.mn2(w) = OPT(1u 20u 4u)
+ m.x8.x1.mn1(w) = OPT(1u 20u 4u)
+ m.x8.x1.mp3(w) = OPT(1u 20u 4u)
+ m.x8.x1.mp1(w) = OPT(1u 20u 4u)
+ m.x8.x1.mp2(w) = OPT(1u 20u 4u)
+ m.x8.x1.mn2(l) = OPT(1u 20u 4u)
+ m.x8.x1.mn1(l) = OPT(1u 20u 4u)
+ m.x8.x1.mp3(l) = OPT(1u 20u 4u)
+ m.x8.x1.mp1(l) = OPT(1u 20u 4u)
+ m.x8.x1.mp2(l) = OPT(1u 20u 4u)
+ m.x8.x2.mn2(w) = OPT(1u 20u 4u)
+ m.x8.x2.mn1(w) = OPT(1u 20u 4u)
+ m.x8.x2.mp3(w) = OPT(1u 20u 4u)
+ m.x8.x2.mp1(w) = OPT(1u 20u 4u)
+ m.x8.x2.mp2(w) = OPT(1u 20u 4u)
+ m.x8.x2.mn2(l) = OPT(1u 20u 4u)
+ m.x8.x2.mn1(l) = OPT(1u 20u 4u)
+ m.x8.x2.mp3(l) = OPT(1u 20u 4u)
+ m.x8.x2.mp1(l) = OPT(1u 20u 4u)
+ m.x8.x2.mp2(l) = OPT(1u 20u 4u)
+ TARGETS differr=0.01 del_F=20NS del_R=20NS
```

Warning

For duplicate optimization variables, SmartSpice will issue warnings for variables that are specified more than once in a given optimization statement. Execution of the deck will not be possible until the error has been corrected and the deck has been re-sourced. A separate error will be issued for each duplication. To illustrate, consider the following input-deck fragment:

```
.MODIF OPTIMIZE
+ wp=OPT(2.0e-7 30e-6 10e-6)
+ wp=OPT(2.0e-7 30e-6 20e-6) $ Duplicate variable
+ lp=OPT(1e-6 10e-6 5e-6)
+ lp=OPT(15e-6 20e-6 8e-6)   $ Duplicate variable
+ TARGETS  trip=1.55
```

If you source it, this optimization statement will generate the following error messages:

```
Error on lines:
  20 : .MODIF OPTIMIZE
  21 : + wp=OPT(2.0e-7 30e-6 10e-6)
  22 : + wp=OPT(2.0e-7 30e-6 20e-6) $ Duplicate variable
  23 : + lp=OPT(1e-6 10e-6 5e-6)
```

```
  24 : + lp=OPT(15e-6 20e-6 8e-6)   $ Duplicate variable
  25 : + TARGETS  trip=1.55
MODIF(OPTIMIZER) : variable 'wp' specified more than once.

MODIF(OPTIMIZER) : variable 'lp' specified more than once.
```

If you attempt to run this deck, SmartSpice will not allow it and will issue this message:

```
Fatal Error: Deck cannot be run until errors in MODIF statement are
fixed.
```

When the duplicate variables are removed and the deck re-sourced, SmartSpice will allow it to run.

18.5 Timing Optimization Using Bisection

Bisection is a method of optimization that uses a binary search to find the value of an input variable for specified value of an output variable (`goal`). The idea of this method is simple. Over some interval the function is known to pass through `goal` value because "function value - `goal`" changes sign. Evaluate the "measured function value - `goal`" at interval's midpoint and examine its sign. Use the midpoint to replace whichever limit has the same sign. After each iteration the bounds containing the root decrease by factor of two. When the size of interval (distance between bounds) is within the error tolerance and latest "measured function value" > "`goal`", bisection has succeeded, and stop.

Note: Bisection search is applied to only one parameter.

18.5.1 Bisection Optimization Syntax

.MODIF Statement

```
.MODIF  <PROFF>  <PRTBL>
+   OPTIMIZE  parname = OPT(lower  upper  <initval>)
+   TARGETS  measname = goalval
+   OPTIONS  METHOD=val   <MAXERR=val>  <NUMITER=val>
```

PROFF	This flag suppresses online printing of `.MEASURE` statement results.
PRTBL	This flag causes SmartSpice to print the final table of parameter and measure values obtained from all iterations.
OPTIMIZE	This keyword is followed by the bisection optimization parameter.
parname	Name of the bisection optimization parameter.
OPT	Mandatory keyword followed by `lower`, `upper` and `initval` in parentheses.
lower	Left bound for bisection optimization parameter.
upper	Right bound for bisection optimization parameter.

initval	Initial value for bisection optimization parameter. Default initval=(lower+upper)/2.
TARGETS	This keyword is followed by the target for optimization.
measname	Name of the measure calculated by a .MEASURE statement. It can be a circuit performance measure like delay, rise/fall time, maximum/minimum value, or difference between a desired and a simulated curve calculated by the ERR .MEASURE statement.
goalval	Goal value.
OPTIONS	This keyword is followed by the list of optimization control options.
METHOD=val	Keyword to indicate which bisection optimization method to use: • 1 is Bisection method, the measure results for lower and upper bounds of optimization parameter must be on opposite sides of goal value. • 2 is PASSFAIL method, the measure must pass for one limit and fail for the other limit.
MAXERR=val	Relative optimization parameter error tolerance. When the difference between the two latest test input values is smaller, then: $parTol = \max(interval \times MAXERR, 10^{-16})$, where: $interval = \max((initval - lower), (upper - initval))$, Bisection optimization process will be terminated. Default is 0.01.
NUMITER=val	Maximum number of iterations. The bisection optimization process will be terminated, when the number of iterations reaches: $ITERlimit = \max(((\log(1/(MAXERR))) / (\log(2)) + 10), NUMITER)$ Default is NUMITER=15.

Example

```
.PARAM vddv=5
.PARAM DelayTime=0

.MODIF
+   OPTIMIZE  DelayTime = OPT(0.0n, 5.0n, 0.0n)
+   TARGETS   MaxVout = vddv
+   OPTIONS   METHOD=1  MAXERR=1.e-3  NUMITER=20

.MEASURE TRAN MaxVout MAX 'vddv-v(D_Output)'
```

HSpice™ Compatibility Syntax

To perform Bisection optimization using HSpice™ compatibility syntax, the .PARAM, .TRAN, .MODEL and .MEASURE statements must be used.

```
.PARAM  parname = OPTxxx(initval lower upper)

.TRAN  tstep  tstop  <tstart>  <tmax>  <UIC>
+   <CALLV>  <SAVEV < =tsave>>   <TRANOP < =top>>  <STORE=num>
+   SWEEP  OPTIMIZE = OPTxxx   RESULT = measname   MODEL=modname

.MODEL   modname OPT  <METHOD = BISECT<ION> | PASSFAIL>
+   <RELIN=val> <ITROPT=val>

.MEASURE   TRAN measname ...    <GOAL < |=| > val>
```

parname	Name of the Bisection optimization parameter.
OPTxxx	Optimization parameter reference name must agree with the OPTxxx name given in the .TRAN statement associated with the keyword OPTIMIZE.
initval	Initial value for bisection optimization parameter. Default is initval=(lower+upper)/2.
lower	Left bound for bisection optimization parameter.
upper	Right bound for bisection optimization parameter.
OPTIMIZE	This keyword is followed by the Bisection optimization parameter.
RESULT	This keyword is followed by the target for optimization.
measname	Name of the measure calculated by a .MEASURE statement. It can be a circuit performance measure like delay, rise/fall time, maximum/minimum value, or difference between a desired and a simulated curve calculated by the ERR .MEASURE statement.
MODEL	This keyword is followed by the optimization model.
modname	The model name. It is used by Bisection optimization to reference a particular model.
METHOD	Keyword to indicate which bisection optimization method to use. Default is BISECTION.
BISECTION	The measure results for lower and upper bounds of optimization parameter, must be on opposite sides of goal value.
PASSFAIL	The measure must pass for one limit and fail for the other limit.

RELIN=val	Relative optimization parameter error tolerance. When the difference between the two latest test input values is smaller, then: $parTol = \max(interval \times RELIN, 10^{-16})$, where: $interval = \max(upper - lower)$, and for "=" relation case $\lvert val - goal \rvert < \max(\lvert goal \rvert \times RELIN, 10^{-16})$ with $val > goal$, Bisection optimization process will be terminated. Default is RELIN value is 0.001.
ITROPT=val	Maximum number of iterations. The Bisection optimization process will be terminated, when the number of iterations reaches: $ITERlimit = \max(((\log(1/(MAXERR)))/(\log(2)) + 10), ITROPT)$ Default is NUMITER=20.
GOAL=val	Goal value. Default is 0.

Example

```
.PARAM vddv=5
.PARAM DelayTime=Opt1(0.0n, 0.0n, 5.0n)

.TRAN .1n  8n
+   SWEEP OPTIMIZE = Opt1  RESULT = MaxVout  MODEL = OptMod

.MODEL OptMod OPT METHOD=BISECTION
.MEASURE TRAN MaxVout MAX 'vddv-v(D_Output)'  Goal = 'vddv'
```

18.5.2 Bisection Optimization Output Information

While Bisection optimization is performing, the process prints the following information: modified input parameter value, measured values at each iteration (step), and optimized parameter value.

```
...
BISECTION OPTIMIZATION : STEP #5
 MODIFIED PARAMETER VALUES :
 delaytime =  1.875e-09

                  * DFF_top Bisection Search for Setup Time
  Transient Analysis, 27 deg C,  BiSection: step #5 ,Tue Sep. 4
13:42:04 2001

maxvout        =   5.2793e-05 at=   3.6995e-09
...
BISECTION OPTIMIZATION : STEP #16
 MODIFIED PARAMETER VALUES :
 delaytime =  2.099e-09

                  * DFF_top Bisection Search for Setup Time
  Transient Analysis, 27 deg C,  BiSection: step #16 ,Wed Sep. 5
12:51:42 2001

maxvout        =   5.0001e+00 at=   4.4675e-09

          Optimization completed, the condition
          relin = 1.000000e-03 is satisfied

 *** optimized parameters
 .PARAM delaytime = 2.099304e-09
```

When the OPTLST option is set (.OPTION OPTLST=1), the process prints additional information: at first and second iterations (when the measures are performed at the lower and upper bounds)

```
bisec-opt iter = 1  meas = 7.334869e-05  goal = 5.000000e+00
bisec-opt iter = 2  meas = 5.000060e+00  goal = 5.000000e+00
```

at each next iteration:

```
 bisec-opt iter = 5  xlo = 1.250000e-09  xhi = 2.500000e-09
 x = 1.875000e-09  xnew = 2.187500e-09
 err = 6.250000e-10  tol = 5.000000e-12
 meas = 5.279327e-05  goal = 5.000000e+00
```

and at the final iteration:

```
 bisec-opt iter = 16  xlo = 2.098999e-09  xhi = 2.099609e-09
 err  = 3.051758e-13 < tol = 5.000000e-12
 meas = 5.000070e+00 > goal = 5.000000e+00
          Optimization completed, the condition
          relin = 1.000000e-03 is satisfied
 *** optimized parameters
```

18.5.3 Binary Search Algorithm (PASSFAIL Algorithm)

The syntax for using the binary-search algorithm is very similar to that of a regular Optimizer statement. To enable the algorithm the user should specify METHOD=2 keyword in .MODIF statement.

Begin with .MODIF followed by print statements, as desired. The line following the OPTIMIZE keyword indicates the parameter to be optimized (R1(res)) with boundaries Min=0, Max=1000, and initial value INIT=10 (if INIT is not specified then INIT=MIN+(MAX-MIN)/2). Another parameter can be specified (such as PARAMNONEED, as shown), but it will not be used and a warning will be printed before simulation.

The keyword TARGETS defines the targets used for the binary search. Only the target names given here will be used. Other measurements, like RESULT1 are not used. You must specify a non-zero value for the targets. If several targets are specified, then the test fails only if all targets fail. It is considered to have passed if at least one target passes. If option OFF is used on .MEASURE lines, then this algorithm does not work. However, the option PROFF on .MODIF is available.

Finally, the keyword OPTIONS defines the kind of search to be performed.

The binary search works over an interval specified by a upper and lower boundary limit. A test condition is then run at the mid point in this specified interval. If the test passes then the solution must be in the lower half of the interval and the upper limit is changed to this test point. If the test fails the solution must be in the upper half of the interval so the lower limit is changed to the test point. The test procedure is then repeated over the new defined interval. Repeating this reduction in interval size by 1/2 each time approaches closer and closer to the solution and stops when the result is within user specified limits. This is the procedure of bisectionaly optimization and assumes the interval specified covers the transition from a fail to a working test or working to failed condition.

The following shows an input deck that illustrates the use of the new binary-search algorithm:

```
PASSFAIL DEMO EXAMPLE (AFTER IMPLEMENTATION)
```

The goal is to find the value of R1 (in the interval 0-1000 that gives V(2) = 3V at 40NS.

The following demonstrates the use of the passfail method with a binary-search algorithm:

```
V1 1 0 pulse ( 0 5 0 0.1n 60n 100n )
R1 1 2 20
C1 2 0 10n

.MEASURE TRAN RESULT CROSS V(2) VAL=3
.MEASURE TRAN RESULT1 MAX V(2)
.SAVE v(2) v(1)

.TRAN 1.NS 40NS
.PARAM PARAMNONEED=2

.MODIF prtbl
+ OPTIMIZE
+ R1 (res) = OPT ( 0 1000 10 )
+ PARAMNONEED = OPT ( 0 10 1 )
+ TARGETS RESULT = 1
+ OPTIONS METHOD = 2
.END
```

18.5.4 Advanced Binary Search Algorithm

The Andvanced Binary Search algorithm is invoked by specifying `METHOD=3` in the `.MODIF` statement. The functionality is a derivation of the Binary Search Algorithm (`METHOD=2`) and behaves the same except the PASS/FAIL result is dependant on the results of the associated target `.MEASURE` statement(s).When the target measurement returns `0` (or the test fails), the current point is considered a failure (like a failed measurement). If the measurement returns a non-zero value (and, of course, the test didn't fail), the current point is considered as a valid PASS.

Note: In the normal Binary search algorithm when a `0` value is returned this can be a valid PASS result.

18.6 SmartSpice Optimization Algorithms

In addition to the local optimization algorithm, Levenberg-Marquardt, SmartSpice also has global optimization algorithms. Local optimization algorithms are generally fast, but usually will only work if the initial values of the parameters are close to the final optimized values. Global optimization algorithms use more iterations while attempting to find the optimum set of parameter values but do not require any preconditioning of those parameter values. Global optimization methods are typically slower than local methods, since they must avoid becoming trapped at local minima, and may use a stochastic search in order to locate a global minimum. Global optimizers attempt to find the minimum objection function among all local minima. Hence, this is a more complex problem than local optimization, as the method must look beyond a local minimum in searching for a global minimum. This may involve looking in the design space further away from the starting point compared to local optimizers

The `.MODIF` statement is used to specify initial guess, optimization area and target function constraints.

To use a particular optimizator, set the `OPTIMIZER` option as follows:

```
.OPTION OPTIMIZER=val
```

where `val` is the optimizer type, specified by one of the following:

- LM - Levenberg-Marquardt (default)
- HJ - Hooke-Jeeves method
- SA - Simulated annealing
- PT - Parallel tempering
- GA - Genetic algorithm
- DE - Differential evolution
- H - Hybrid optimizer

The default is LM, which corresponds to the local optimizer, Levenberg-Marquardt.

18.6.1 Optimization Algorithm Preference Settings

The optimization preferences can be set using the **Preferences** dialog under the **Edit** menu. The optimizer section is found under the **Tools** section. Each of these algorithms and their settings is described below.

18.6.2 Levenberg-Marquardt (LM)

Levenberg-Marquardt is a standard nonlinear least-squares optimizer, and works very well in practice if a reasonably good initial set of parameter values is available. It is a local minimum solver which performs a numerical differentiation at each iteration. It only accepts changes in the parameters, which improve the error metric, and so there is a danger that it will find a local minimum instead of the global minimum.

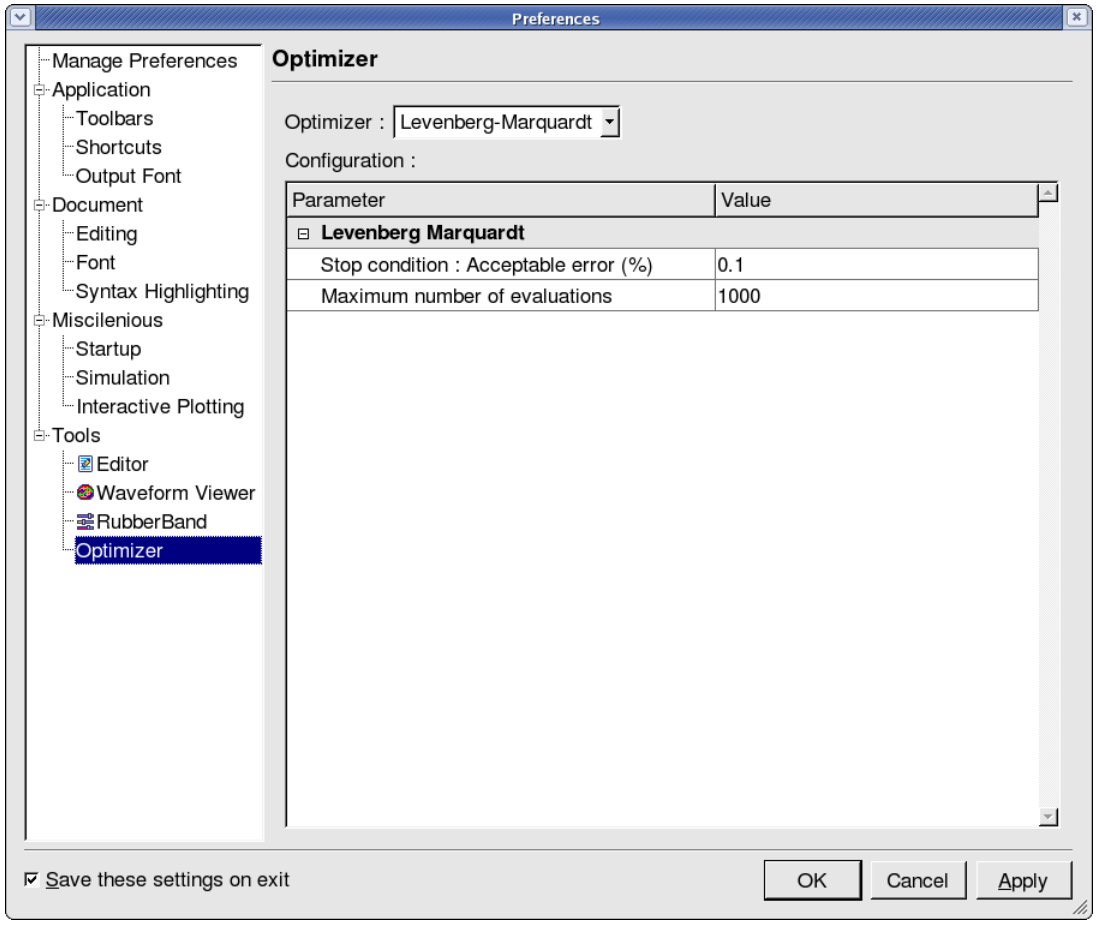


Figure 18-5 Levenberg-Marquardt Configuration

The following configuration parameters can be used for the LM optimizer.

Acceptable error (%)	The optimizer will continue if the incremental error reduction is greater than 0.1% of this value
Maximum number of evaluations	The optimizer will stop if it reaches this number of simulations

18.6.3 Hooke-Jeeves Optimization Algorithm (HJ)

The HJ optimizer is also a local optimizer. While the LM optimizer described above is also a local optimizer, there are some important differences between the two. LM calculates numerical derivatives but HJ belongs to a class of direct search algorithms which does not use derivatives. In this respect, HJ is similar to the global optimizers. As some physical models are discontinuous in the optimization parameters, and because the measured or calculated variables may introduce numerical noise, calculation of numerical derivatives is often impractical or inaccurate. In such cases, it is necessary to use a direct search optimizer, such as HJ.

Each HJ cycle consists of exploration moves followed by pattern moves. An exploration attempts to find an advantageous direction in which to move the parameters. A successful exploration is exploited by repeatedly moving the parameters in the same direction. These are the pattern moves. The amount by which each parameter is moved during an exploration is called it stepsize. At the end of each cycle every stepsize is reduced by the stepsize factor.

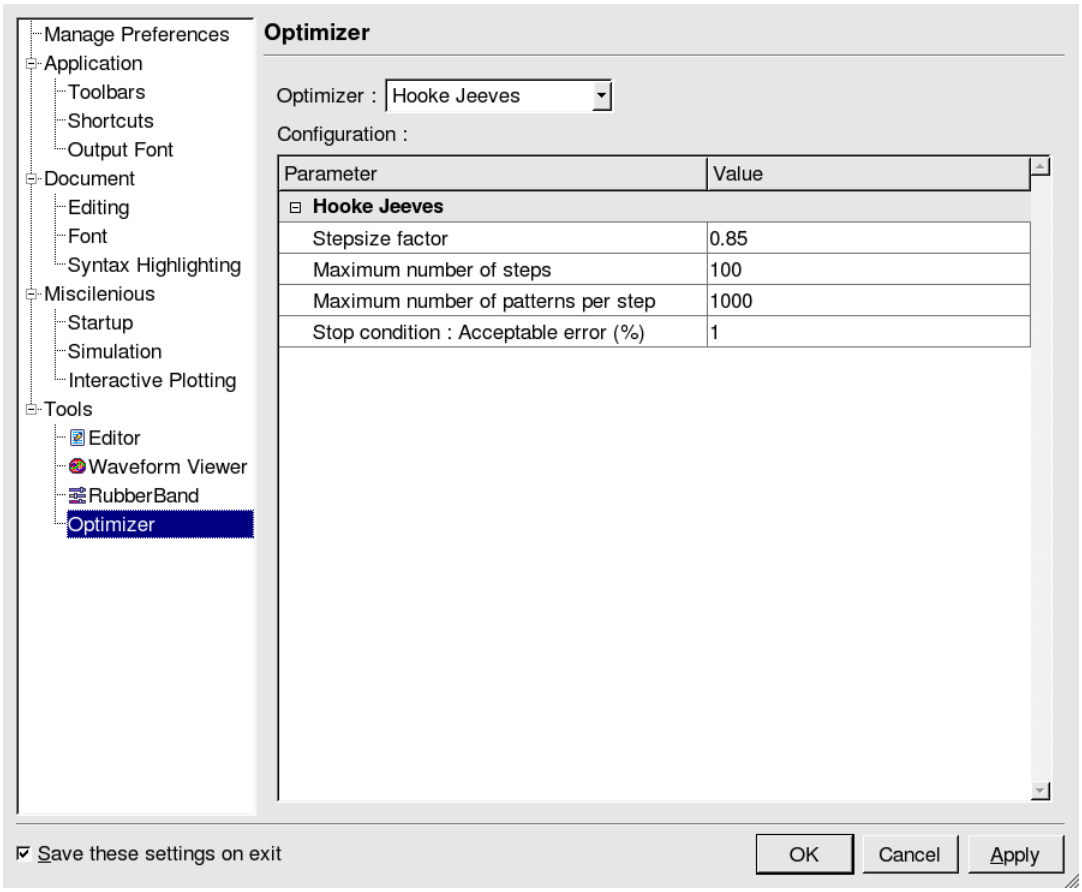


Figure 18-6 Hooke-Jeeves Configuration

The following configuration parameters can be used for the HJ optimizer.

Stepsize factor	Multiples the stepsize of every parameter at the end of each cycle
Maximum number of steps	The optimizer will stop if it reaches this number of steps
Maximum number of patterns per step	The cycle will end if it reaches this number of pattern moves
Acceptable error (%)	The optimizer will stop if the error metric is less than this value

18.6.4 Simulated Annealing Optimization Algorithm (SA)

SA is a global optimizer. The “annealing” is designed to enhance the likelihood of avoiding local minima while searching for the global minimum. Injected randomness helps prevent premature convergence to a local minimum.

The SA method mimics the formation of an actual physical solid from a liquid as the liquid cools. At high temperatures, the liquid is highly disordered, its components are free to move, and its energy is high. As the temperature is lowered, so the energy of the systems falls. If the cooling process is slow enough, an ordered solid (such as a crystal) of the lowest possible energy will form. If the temperature is lowered too quickly, then likely an amorphous solid at a higher energy than the crystal will be formed. In SA, the error metric is analogous to the energy and “cooling” is achieved by gradually decreasing a “temperature”, which is an optimization control parameter. The critical point of the SA algorithm is not to lower the temperature parameter too quickly.

SA models the physical cooling process by choosing new states at random, always accepting steps which cause the value of the error metric to fall, and sometimes accepting those which cause it to rise. Hence, there is always a chance of getting out of a local minimum and afterwards finding the global minimum. At each temperature, a number of sweeps is performed. A sweep consists of testing a (typically large) number of states. These states are chosen according to an internal algorithm.

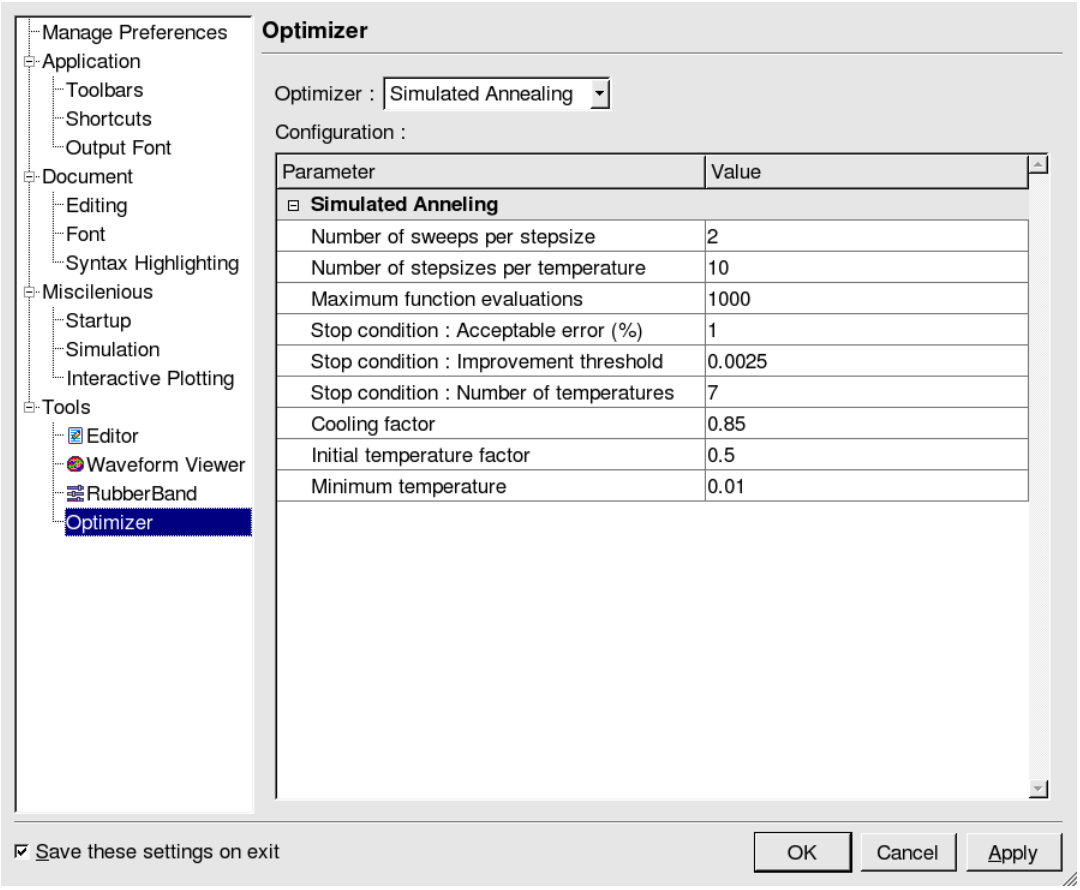


Figure 18-7 Simulated Annealing Configuration

The following configuration parameters can be used for the SA optimizer.

Number of sweeps per stepsize	The number of sweeps executed during each cycle
Number of stepsizes per temperature	The optimizer will stop if it reaches this number of steps
Maximum function evaluations	The number of iterations executed for each temperature
Acceptable error (%)	The optimizer will stop if the error metric is less than this value
Improvement threshold	The optimization will stop if, during the specified previous number of temperatures, none of them has reduced the error metric by the specified amount
Number of temperatures	Number of temperatures for which the error metric must improve without terminating
Cooling factor	Multiplies the temperature after all its sweeps have been completed
Initial temperature factor	Multiplies the initial value of the error metric to set the initial temperature
Minimum temperature	The optimizer will stop if the temperature reaches this value

18.6.5 Parallel Tempering Optimization Algorithm (PT)

Parallel Tempering (PT) is a global optimization method. It is related to SA, but instead of mimicking the gradual cooling process of a single physical system, it consists of an ensemble of systems (called replicas) at different and fixed temperatures. Each replica carries its own copy of the optimization parameters, which will in general diverge from the other replicas as the optimization proceeds.

Each cycle of the PT algorithm begins with a number of sweeps of the SA type. These are followed by exchange iterations in which replicas at adjacent temperatures exchange their configurations with a certain probability. Provided the temperatures are not too dissimilar, this probability will be reasonably high and a kind of tunneling can take place between the systems. Systems at higher temperatures can sample larger areas of parameter space, albeit more coarsely. Those at lower temperatures can sample smaller areas more finely but may become trapped in local minima and require a restart in an ordinary SA optimization.

The central idea of PT is that the exchange of replicas enables systems at higher temperatures to generate new local optimizers at lower temperatures, facilitating good sampling of the whole phase space. The disadvantage of PT is of course that simulation of replicas, rather than one, requires on the order of M times more computational effort.

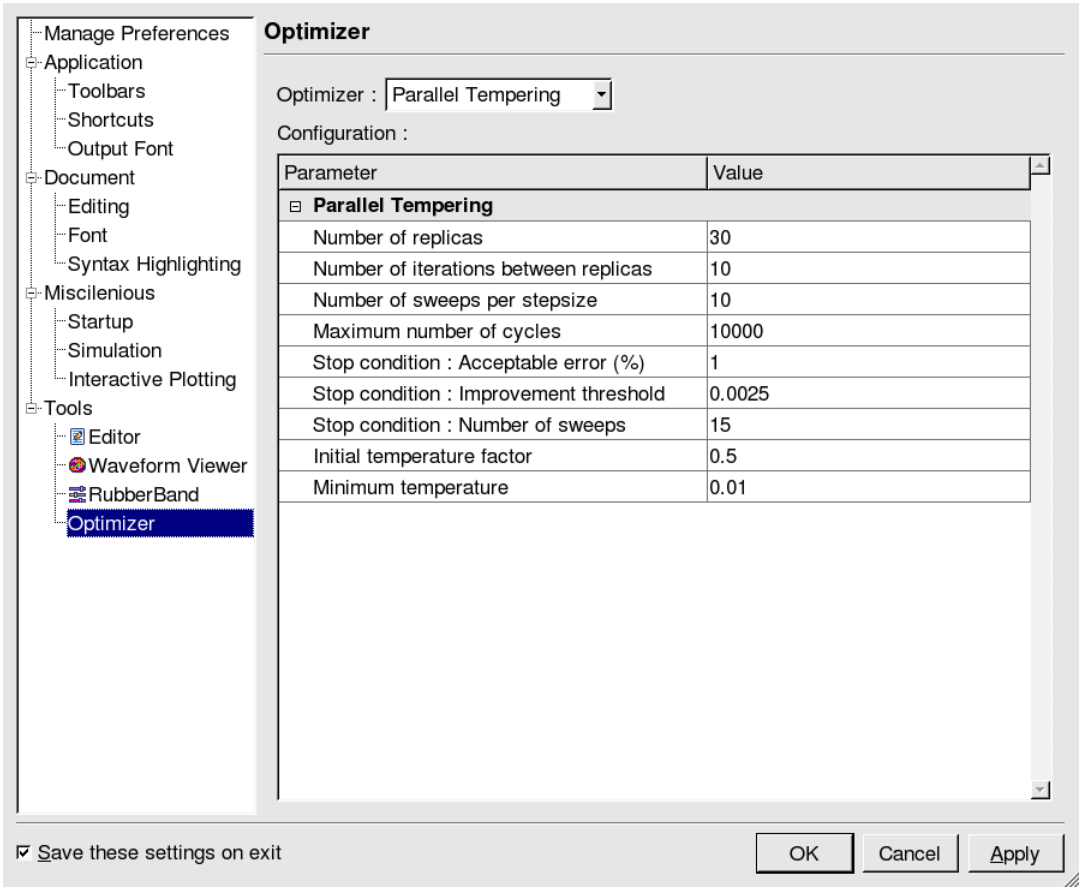


Figure 18-8 Parallel Tempering Configuration

The following configuration parameters can be used for the PT optimizer.

Number of replicas	Number of copies of the optimization parameters to be used, each at a different temperature
Number of iterations between replicas	Number of exchange iterations performed during a single cycle of the optimization
Number of sweeps per stepsize	Number of sweeps performed during a single cycle of the optimization
Maximum number of cycles	The optimizer will stop if it reaches this number of cycles
Acceptable error (%)	The optimizer will stop if the error metric is less than this value
Improvement threshold	The optimization will stop if, during the specified previous number of cycles, none of them has reduced the error metric by the specified amount
Number of sweeps	Number of cycles for which the improvement threshold must improve without terminating
Initial temperature factor	Multiplies the initial value of the error metric to set the initial temperature
Minimum temperature	The optimizer will stop if the temperature reaches this value

18.6.6 Genetic Optimization Algorithm (GA)

Genetic algorithm (GA) is a global optimizer and an evolutionary algorithm, that is, it uses techniques inspired by evolutionary biology. GA works with a population of candidate solutions to the optimization problem. At each step it generates a new population from the old. The idea is that the error metric plays the role of the evolutionary pressures found in nature and causes the evolution of an improved population.

GA uses crossover and mutation operators, analogously to biological reproduction. The crossover operator produces two children from two parents by mixing parts of both parents. The mutation operator randomly changes parts of a single child.

Individuals are chosen to contribute to the next generation by one of two means. Tournament means that pairs are compared at random and the one with the better error metric is chosen. Universal stochastic sampling chooses contributors based on their fitness relative to the whole population.

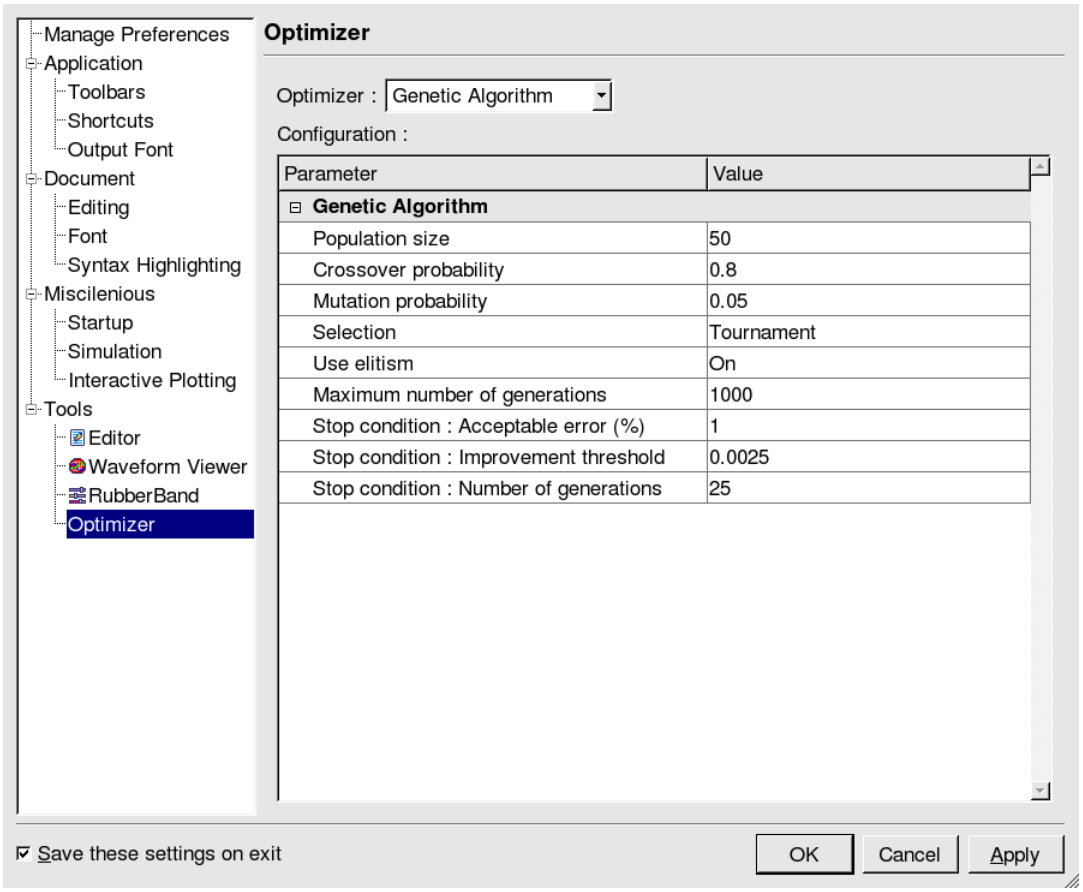


Figure 18-9 Genetic Algorithm Configuration

The following configuration parameters can be used for the GA optimizer.

Population size	The number of copies of the parameters used in each generation
Crossover probability	The probability that two parents will generate two children by exchanging their parameter values. This value should be relatively high
Mutation probability	The probability that parameter values will change spontaneously in a single child. This value should be relatively low
Selection	Choose tournament or stochastic method to select the next generation
Use elitism	If On, the fittest member of the current generation will always be passed to the next
Maximum number of generations	The optimizer will stop if it reaches this number of generations
Acceptable error (%)	The optimizer will stop if the error metric is less than this value
Improvement threshold	The optimization will stop if, during the specified previous number of generations, none of them has reduced the error metric by the specified amount
Number of generations	Number of generations for which the improvement threshold must improve without terminating

18.6.7 Differential Evolution Optimization Algorithm (DE)

Like GA, differential evolution (DE) is a global optimizer and an evolutionary algorithm, that is, it uses techniques inspired by evolutionary biology. Unlike GA, which always operates directly on the population members, DE also uses differences between members as it creates candidate solutions. It constructs a population of these candidates matching the current population. Only those which improve over their counterparts are accepted, so all the members of the next generation are always at least as good as the current one.

The amplification factor multiples the difference between existing individuals to make an intermediate. The crossover probability is used to decide which parts of an intermediate contribute to a candidate for the next generation.

Several strategies exist to combine members of the current generation. BEST means that the member with the best error metric always contributes. The numbers 1 and 2 indicate how many members contribute. EXP chooses parameters from intermediates in a block; BIN chooses parameters on a per-individual basis.

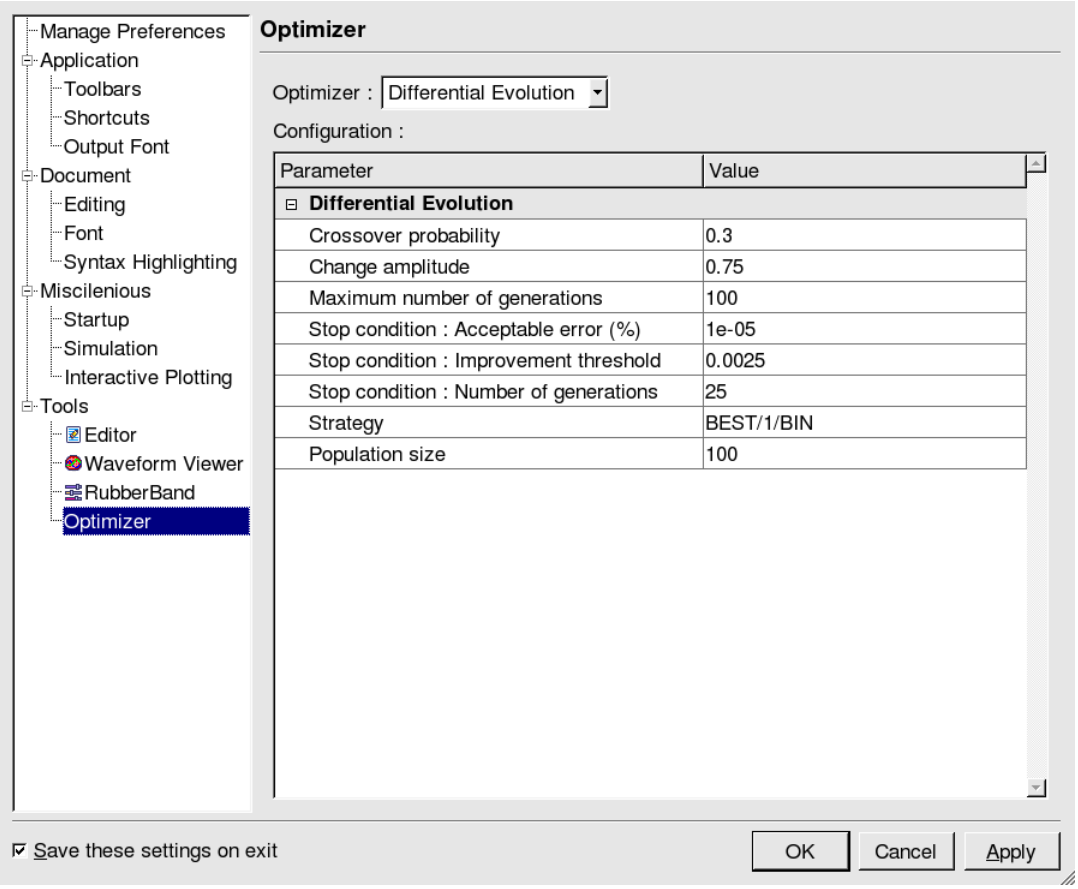


Figure 18-10 Differential Evolution Configuration

The following configuration parameters can be used for the DE optimizer.

Crossover probability	The probability that parts of an intermediate will contribute to a candidate
Change amplitude	Multiplies differences in members contributing to an intermediate

Maximum number of generations	The optimizer will stop if it reaches this number of generations
Acceptable error (%)	The optimizer will stop if the error metric is less than this value
Improvement threshold	The optimization will stop if, during the specified previous number of generations, none of them has reduced the error metric by the specified amount
Number of generations	Number of generations for which the improvement threshold must improve without terminating
Strategy	The means by which candidate members of the next generation are constructed
Population size	The number of copies of the parameters used in each generation

18.6.8 Hybrid Optimization Algorithm (H)

The hybrid optimizer allows a single selection of two sequential optimizations to take best advantage of the strengths of both global and local optimization algorithms. Typically, the first optimization is performed using a global optimizer. This optimization provides a good initial starting point for the much faster local optimizer.

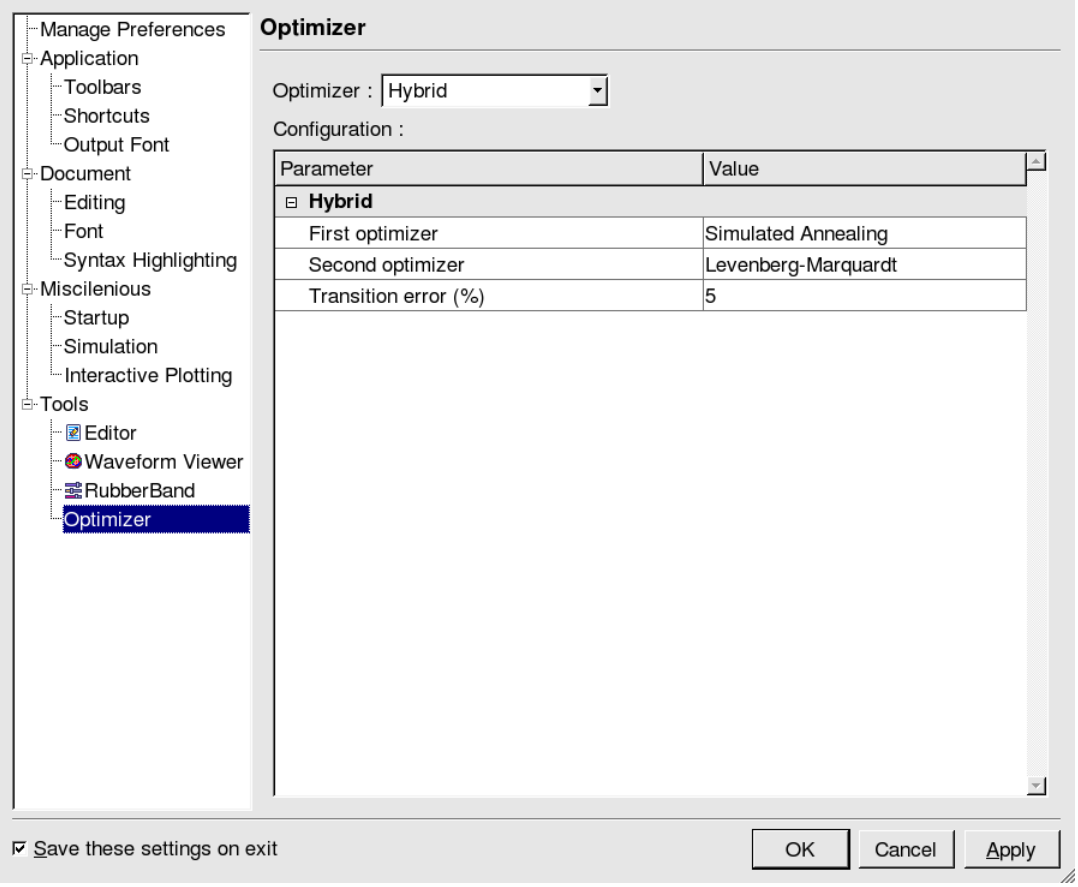


Figure 18-11 Hybrid Configuration

The following configuration parameters can be used for the Hybrid optimizer.

First optimizer	The optimization algorithm to be run first
Second optimizer	The optimization algorithm to be run second
Transition Error (%)	The first optimization will stop and the second will begin when this error metric is reached

Chapter 19
Mismatch Analyses

19.1 DCMatch Analysis

Manufacturing variations result in process and device parameter variations from lot to lot, wafer to wafer, die to die and device to device, and can be categorized as systematic or random. Lot-to-lot and wafer-to-wafer variations are common to all devices in the circuit (e.g., due to over-etching, all transistors have a shorter than nominal length). This gate length reduction introduces a systematic shift in the device characteristics and therefore circuit performance.

DCMatch analysis is calculating mismatch device characteristics using variations of the model parameters. The combined device influence on OP solution is modeled as a linear superposition of device variations. The Variability block is used to specify parameters variations specification. DCMatch analysis uses solution of undisturbed OP for calculation of DC transfer curve variation. The Output is organized in tables for each target system variable: node voltage, node voltage difference and current via independent voltage source.

The commands `.probe` and `.measure` provides postprocessing interface for `.dcmatch` analysis

19.1.1 Input Syntax

The following statement syntax performs a DCMatch analysis on the circuit:

```
.DCMATCH <outvar1 <outvar2 ...> <threshold=val> <file=filename>
+ <interval=num> <perturbation=val>
```

outvar1...	Output variables; can be node voltage, voltage difference, voltage source current.
threshold=val	Only devices with a relative variance contribution above threshold will be reported in the summary table. If `threshold=0`, SmartSpice will print summary table for all devices. If `threshold<0`, SmartSpice will not print summary table. Default is 0.01.
file=filename	DCMatch output file name.
interval=num	Summary table will be printed only for each subsequent increment of `num`. Must be a positive integer. Default is 1.
perturbation=val	Perturbations of `val` standard deviation will be used. Must be from 0.01 to 6. Default is 2.

Example 1

```
.DCMATCH V(3) V(6,8) I(VDD)
```

In this example, SmartSpice will output DCMatch variations on the voltage of node `3`, the voltage difference between nodes `6` and `8`, on the current through the `VDD` voltage source.

Example 2

```
.DC VALUE START=1 Stop=7 Step=0.5
.DCMATCH V(5) INTERVAL=5
```

In this example, SmartSpice will output DCMatch variations on the voltage of node `5`. Summary tables will be printed for `VALUE=1`, `3.5`, `6` and `7`.

19.1.2 DCMatch Table Output

```
+ 0:gnda=0.0000e+000  0:in_pos=1.2500e+000  0:vdda=2.5000e+000
+ 0:in_neg=1.2492e+000  0:out=1.2492e+000  1:net031=2.1393e+000
+ 1:net044=7.4391e-001  1:net18=2.0352e+000  1:net0148=3.6789e-001
+ 1:net058=2.0352e+000  probe_m.xi82.mp5=1.2492e+000  probe_m.xi82.mp4=2.0352e+000
+ probe_m.xi82.mp3=2.1393e+000  probe_m.xi82.mn6=1.2492e+000  probe_m.xi82.mn7=7.4391e-001
+ probe_m.xi82.mn8=3.6789e-001  probe_m.xi82.mn2=2.0352e+000  probe_m.xi82.mn1= 2.1393e+000
```

Subcircuit Names Cross Reference

Aliases Full Names Aliases Full Names

1 xi82

****** dc mismatch analysis tnom= 27.000 temp= 25.000

sweep point = operating point

==========

output = v(out) node voltage = 1.248600e+000 V threshold = 1.000e-001

perturbation = 2.00 interval = 3

Output 1-sigma due to total variations 1.325239e-002 V

DCMATCH LOCAL VARIATION

10 Devices had no Local Variability specified

Output 1-sigma due to local variations= 1.325239e-002 V

0 Devices with Local Contribution Variance larger than Threshold

---

Contribution 1-Sigma(V)	Contribution Variance (%)	Cumulative Variance (%)	Matched pair	Device Name
7.940e-004	0.36	0.36	-	m.xi82.mp5
8.829e-004	0.44	0.80	-	m.xi82.mp4
8.275e-003	38.99	39.79	-	m.xi82.mp3
1.009e-002	57.94	97.74	-	m.xi82.mn6
3.814e-004	0.08	97.82	-	m.xi82.mn7
2.223e-004	0.03	97.85	-	m.xi82.mn8
9.963e-004	0.57	98.41	-	m.xi82.mn2
1.669e-003	1.59	100.00	-	m.xi82.mn1

==========

- Operating point information
- Name and the OP value of the output variable

- DCMatch options
- Output sigma of combined global and local variations
- Number of devices that had no local variability specified
- Output sigma caused by local variations
- Number of devices with local variance contributions below the threshold (excluded from table)
- List of devices which contributes to deviation of system variable
- Rated variance of individual model parameter
- Cumulative variance for each individual parameter

19.1.3 Output Commands

The following DCMatch output commands can be used with all SmartSpice output dot statements (including the .MEASURE statement). If multiple output variables will be specified in DCMatch analysis statement, only the last one will be used in the .MEASURE statement.

DCM_Total, DCMatch_Total	Output sigma due to Global and Local variations.
DCM_Local	Output sigma due to Local variations.
DCM_Local(dev)	Contribution of device dev to output sigma due to Local variations.

Example 1

```
.DCMATCH v(3) v(4)
.PROBE DCM_TOTAL DCM_LOCAL(m.x1.m1)
```

In this example, total variations and contributions of the device m.x1.m1 will be probed. SmartSpice will probe the following vectors: dcm_total(v(3)), dcm_total(v(4)), dcm_local(v(3), m.x1.m1), and dcm_local(v(4), m.x1.m1).

Example 2

```
.DCMATCH v(3) v(4)
.MEASURE DC dcmoffset FIND DCM_LOCAL AT=2
```

In this example, dcm_local(v(4)) will be used in the .MEASURE statement.

Chapter 20
Variation Block

A Variation Block is a container for specifying variations of the model/element parameters and can be combinded with other analysis like DC Match etc..

There are 3 main inner blocks in the Variation block:

- Global Variations are defined as variations in device characteristics from lot to lot, wafer to wafer and chip to chip. Global Variations influence all devices with the same model name in the same manner.
- Local Variations are defined as variations between devices in proximity, or with common centroid layout on the same chip. Local Variations are caused by the tiny variations in material composition and geometrical definition during manufacture of the devices such that different instances of the same device will have slightly different electrical properties.
- Spatial Variations are defined as variations due to the physical arrangement of the devices on the chip. Spatial Variations are caused by gradients, from material properties, imperfections of equipments and spin processes.

A Variation Block consists of the 4 sections:

- general section
- inner block for Global Variations
- inner block for Local Variations
- inner block for Spatial Variations

The structure of the Variation Block is:

```
.Variation
Common options
Common parameters applied to all inner blocks
.Global_Variation
   Global random variables
   Additional global random variables
   Definition of global variations of model parameters
.End_Global_Variation
.Local_Variation
   Local random variables
   Additional local random variables
   Definition of local variations of model parameters
     .Element_Variation
      Definition of variations of element parameters
     .End_Element_Variation
     .End_Local_Variation
.Spatial_Variation
   Spatial random variables
   Additional spatial random variables
   Definition of spatial variation of model parameters
.End_Spatial_Variation
.End_Variation
```

20.1 General Section

In the general sections, common options can be defined. These options control behaviour of the Variation Block. The parameters defined here are applied to all inner blocks. The Variation block parameters do not interact with netlist parameters.

The following options can be defined at the top level:

`option ignore_variation_block=yes|no`	The option tells the simulator to ignore the Variation Block and executes other style variations (AGAUSS, GAUSS, AUNIF, UNIF). Default is No
`option ignore_local_variation=yes|no`	The option tells the simulator to ignore local variations in simulation. Default is No.
`option ignore_global_variation=yes|no`	The option tells the simulator to ignore global variations in simulation. Default is No.
`option ignore_spatial_variation=yes|no`	The option tells the simulator to ignore spatial variations in simulation. Default is No.
`option output_sigma_value=value`	This option helps in reporting results in terms of sigma values which are typically one, three, or six sigma based on the standards used in different companies. Default is 1; range is 1 to 10. Note that the input sigma is not affected.
`option vary_only subckts=subcktlist`	This option limits variation to the specified subcircuits. Subcircuts are separated by comma. Default not used.
`option do_not_vary subckts=subcktlist`	This option disables variation on the specified subcircuits. Subcircuts are separated by comma. Default not used.
`option sampling_method=value`	This option enables support for advanced sampling methods. Possible values are SRS (Simple Random Sampling), LHS (Latin Hypercube Sampling), OFAT (One-Factor-at-a-Time), Sobol, and Niederreiter.
`option replicates=value`	This option is used to replicate sampling of LHS
`option use_agauss_format=no|yes`	The option provides ability to combine traditional Monte Carlo Gaussian trials with Variation Block advanced sampling methods. Default not used.
`option random_generator=default|msg`	The user can specify the random number generator used in Variation Block based Monte Carlo analysis. Allowed values are MSG and Default.

`option stream=value|random|default`	The option specifies an integer stream number for random number generator (only for Variation Block). The numerical value should be between 1 and 20. If `stream=random`, the simulator chooses random stream number. Default value is equivalent to `stream=1`.
`option normal_limit=value`	The option restricts the range for the numbers generated by the random number generator for normal distributions. It is possible to specify values between 0.1 and 6.0. The default value is 4.0.
`option print_only subckts=subcktlist`	This option is used to limit output to the specified subcircuits. Subcircuts are separated by comma. Default not used.
`option do_not_print subckts=subcktlist`	This option excludes output from the specified subcircuits. Subcircuts are separated by comma. Default not used.

20.2 Inner blocks for Global, Local and Spatial Variations

The independent variables can be defined inside each of these blocks. The variables are defined as specific distributions over their own scope (Global, Local or Spatial). Additional variables can be defined through extra expressions.Variations on model parameters may be defined in these three inner blocks.

An additional inner section inside the Local Variation block defines the variations of the specified element parameters. It possible to use this subblock for defining local temperature variations or variations for elements that don't have models.

20.3 Independent Variables

Independent random variables defined in the mentioned inner block use the following three distributions:

- Uniform distribution: defined over the range from -0.5 to 0.5: U()
- Normal distribution: with mean=0 and variance=1, default range +/-4: N()
- User-defined cumulative distribution function: CDF(xyPairs)

The CDF is a sequence of pairs of values x and y. There are some rules the pairs should satisfy:

1. CDF begins with value $y_0 = 0$ and ends with $y_n=1$
2. At least 2 pairs should be specified
3. x values are monotonically increasing
4. y values are monotonically non-decreasing

The samples of the distribution definitions:

```
Parameter X=N()
Parameter Y=U()
```

The distributions N() and U() accept no arguments.

```
Parameter Z=CDF(0 0 2 0.3 4 0.4 6 0.4 8 0.5 10 1)
```

The CDF distribution coded by Z variable has the following probability density function and cumulative distribution function:

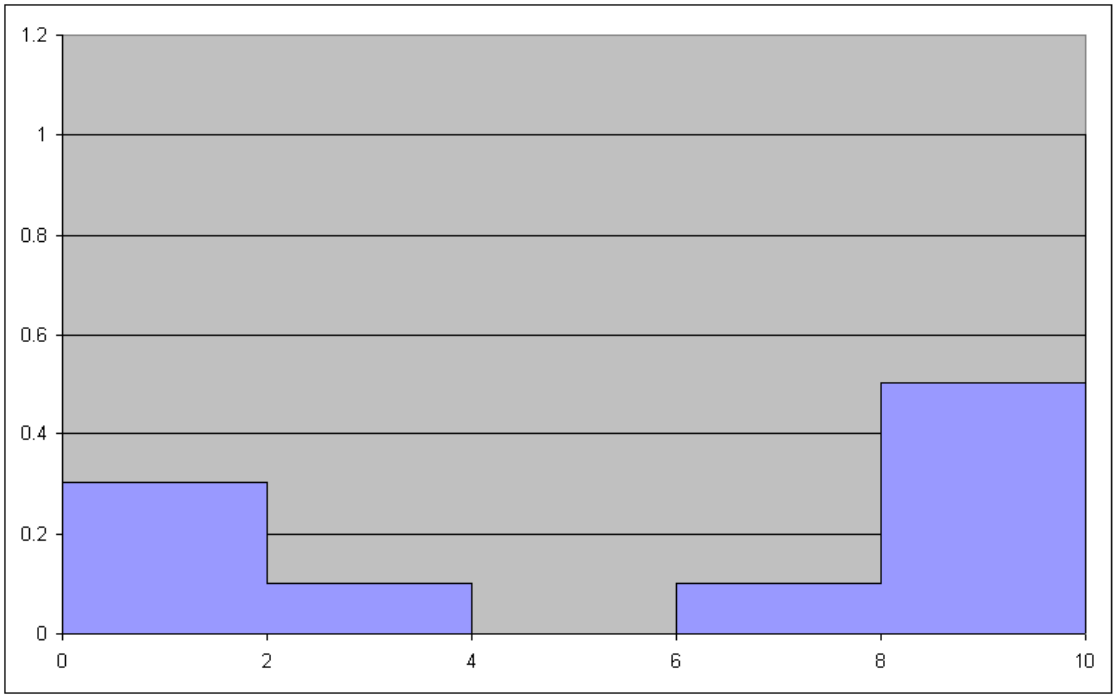


Figure 20-1 Probability Density Function

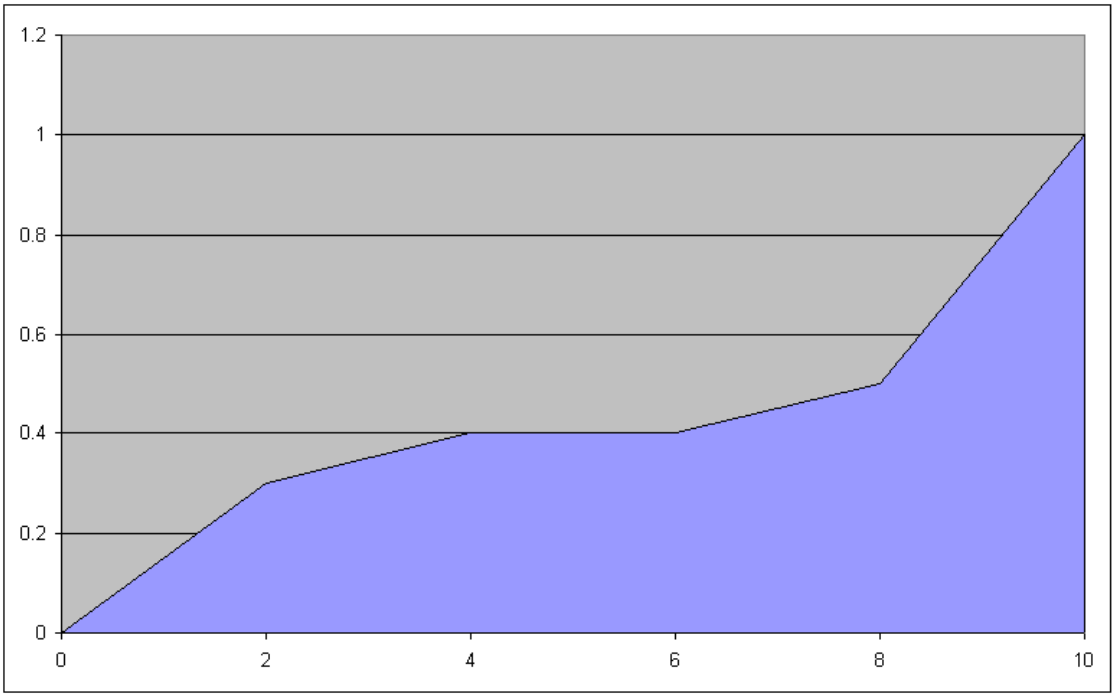


Figure 20-2 Cumulative Distribution Function

20.4 Dependent Variables

For complicated model distributions, you can define dependent variables. The dependent variables use previously defined independent variables in expressions.

Example 1

The example defines a random variable with normal distribution, with zero mean and standard deviation equals 10.

```
Parameter NormalD=N()      A='NormalD*10'
```

Example 2

The examples defines a random variable with uniform distribution, with mean equals Y and standard deviation equals W

```
Parameter X=U()      Z='Y+X*W'
```

20.5 Variations of Model Parameters

The variations of model parameters are defined in the following manner:

```
Model_Type Model_Name Model_Parameter=Expression_for_Sigma
```

If you want to use any of previously defined independent or dependent variables within the expression you should use the keyword `perturb`:

Example 1

This example shows independent variable `rnd1` with normal distribution and standard deviation. The global variation is applied to parameter par1 of all NMOS devices with specified model `nch_mac`. The applied variation has normal distribution and deviation 3 times less standard.

```
.variation
     .global_variation
          parameter rnd1=N()
          nmos nch_mac par1=perturb('rnd1/3')
     .end_global_variation
.end_variation
```

Example 2

The following examples reveal 2 subtypes of variations: absolute and relative. The variation of parameter `vth0` is specified as normal with absolute sigma 0.05V. The variation of parameter u0 is specified as normal with relative sigma 15% of original value.

```
.variation
     .global_variation
          nmos nch_mac vth0=0.05 u0=15 %
     .end_global_variation
.end_variation
```

Relative variation means that the original value of the parameter is multiplied by the specified percent and divided by 100%:

```
u0 = u0original x 15% / 100%
```

Variation block supports global variations for the following models: NMOS, PMOS, R, Q, D and C. In case of binning models, the variation is applied to all bins. It's possible to specify the variation for specific bin(s):

```
.variation
     .global_variation
          nmos nch_mac.1 vth0=0.05
          nmos nch_mac.2 vth0=0.04
          nmos nch_mac.3 vth0=0.06
     .end_global_variation
.end_variation
```

20.6 Variations of Element Parameters

Element variations are described using the following syntax:

```
Element_Type Element_Parameter = Expression_for_Sigma
```

The element variation is useful for the situation where the decision of what model to use for the passive device hasn't been made yet.

There is an ability to specify the variation of the element parameters on selected models and elements:

```
Element_Type(model_name~='modelNameA')
Element_Type(element_name~='elNameB')
Element_Type(model_name~='modelNameC' LOGICAL OPERATOR
     element_name~='elNameD') par='exp'
Element_Type(model_name~='modelNameE' LOGICAL OPERATOR
     model_name~='modelNameF') par='exp'
Element_Type(element_name~='elNameG' LOGICAL OPERATOR
     element_name~='elNameH') par='exp'
```

The variation block supports the following logical operators: AND (&&), OR (||). The operator "~=" means "equals". It is possible to use wildcard inside element and model names. The question mark (?) replaces a single character in the model/element name, and the asterisk (*) replaces multiple characters. Matching is case-insensitive.

The variation block supports the following elements: M, R, C, Q, D, L, I, and V.

Example 1

In the example, the capacity of all the capacitors is varied with a Gaussian variation with zero mean and sigma 10e-12.

```
.Element_Variation
     C C=10e-12
.End_Element_Variation
```

Example 2

The example shows how to use previously defined independent random variable X in element variation.

```
.Element_Variation
     R R=Perturb('X/5')
.End_Element_Variation
```

The Get_E() function is used to access geometry dependant parameters of instantiated elements to be used in an expression. This function is specific to the variation block and used to create a variation expression dependant on final element geometry parameters.

Example 3

The expression:

```
u0='2.345e-6/sqrt(get_E(W)*get_E(L)*get_E(M))' %
```

is a percentage variation of mobility (u0) based on the Width, Length and multiplying factor M of the transistor element.

Chapter 21
Using Solvers in SmartSpice

21.1 Matrix/Solver Options

SOLVER=val	Numerical method for matrix inversion:
• **SOLVER=SPEEDS**	is a modified version of XMS solver. Modification mainly is done in a preconditioner which is simpler and faster than XMS. Some accuracy penalty might be observed using SPEEDS during OP calculation. Recommended for digital large circuits (more than 500k active elements). LU decomposition is the same as in XMS (accuracy and performance).
• **SOLVER=XMS**	(default) is a re-engineered sparse solver. It can be much faster than the default solver on large systems, and it supports both real and complex computations.
• **SOLVER=BCG**	The iterative BCG solver implements the preconditioned version of the Bi-Conjugate Gradient Stabilized method. See Section 21.3 BCG (BiCGstab).
• **SOLVER=AMS**	The iterative AMS solver implements the preconditioned version of the Flexible Generalised Minimum Residual Method (FGMRES). The possible choices for a preconditioner are AMP and ILK. See Section 21.3 BCG (BiCGstab).
• **SOLVER=DDS**	The direct domain decomposition solver (DDS). DDS solver is recommended in HPP mode (see Section 27.3 SmartSpice HPP Mode). In non-HPP mode the solver works in the single processor mode and handles the global matrix.
ITERTOL=val	Tolerance for iterative solver. Default is 1E-12.
PIVALG=val	Pivoting algorithm selector. If PIVALG=0, then zero diagonals are ignored. If PIVALG=1, then zero diagonals are pre-ordered. Default is 1 for XMS and 0 for others. See Note 1.
PIVREL=val	Resets the relative ratio between the largest column entry and an acceptable pivot value. In the numerical pivoting algorithm, the minimum allowed pivot value is determined by: $epsrel = max(PIVREL \times maxval, PIVTOL)$ where *maxval* is the maximum element in the column where a pivot is sought (partial pivoting). Default is *1.0E-3*. See Note 1.
PIVTOL=val	Resets the absolute minimum value for matrix entry to be accepted as a pivot. Default is *1.0E-16* for XMS solver and *1.0E-13* for others. See Note 1.

PIVOT	The option PIVOT selects pivoting method. Default is PIVOT=30. • PIVOT=1: Nonpivoting algorithm. • PIVOT=2: Search for singleton to use as a pivot - if there is no singleton search the entire matrix for piviot. • PIVOT=4: Quick search the diagonal of the matrix for pivot, if no acceptable pivot is found search the entire matrix. • PIVOT=8: Detailed search of the diagonal of the matrix for piviot. If no acceptable pivot is found search the entire matrix. • PIVOT=16: Search the entire matrix for pivot. • PIVOT=30: (Default) Search for singleton - if not found, quickly search diagonal; if no acceptable pivot found, detailed search of diagonal; if no acceptable pivot is found search the entire matrix. If an acceptable pivot is found during any of the above 4 steps the chosen pivot is used and no further search is done. See Note 1.
EQNTHRESHOLD	Supports the solver switch from default to SPEEDS. SmartSpice will use solver SPEEDS if the option solver is not given, the option eqnthreshold is specified and circuit equation size is greater than eqnthresholdvalue. Default eqnthreshold value is -1. The option eqnthreshold can be set using the variable eqnthreshold in the INI-file.
MTSOLVERTHRESHOLD	This option allows solver multithreading when number of equations in the circuit is greater than the specified value val.
ORDERING=val	The option ORDERING allows you to add/exclude the different types of orderings from the default set for both direct solvers XMS and SPEEDS. val should be in form sign ordering_name where sign is '+' to include or '-' to exclude the specified ordering from the default set and ordering_name should be one of the following orderings: AMD_SUM, RCM, MD, ND, MNA, MPD, ZFD (see Section 21.5 Solver Orderings (XMS and SPEEDS solvers)). Two or more orderings should be devided by space (see the example below). Default is " "(none) which means that no orderings will be added/excluded from the default set for used direct solver. XMS solver uses the following orderings by default: MNA, AMD_SUM. SPEEDS solver uses the following orderings by default: MNA, AMD_SUM, MPD.

Example

```
.opt ORDERING= "+MPD -AMD_SUM"
```

In this case, the following orderings will be used for XMS solver: MNA, MPD.

SmartSpice will print out the following message:

```
Note: .option 'ordering' set the following orderings: 'MNA' 'MPD'
```

Note: Use the -PS SmartSpice option to specify the number of threads for the solver (see Section 21.2 Solver Multithreading).

Note 1: Pivoting algorithm and the option PIVOT applies only to XMS solver.

21.2 Solver Multithreading

Starting from 3.17.3 (3.16.5), by default, SmartSpice will request the maximum available CPU's for solver multithreading. If it's necessary to limit to N the number of requesting processors for solver multithreading, use "-PS N" command line option. The default XMS solver internally could reset this number to 1 if the matrix size is smaller than 20000 or value set from the option mtsolverthreshold. Otherwise it will use for multithreading all processors requested from SmartSpice. It is planned to use the same matrix size limit for one CPU operation for PAM solvers.

Examples

To run SmartSpice on 4 threads, and the solver on 2 threads:

```
smartspice -P 4 -PS 2
```

To run SmartSpice on the maximum number of CPU's available and the solver with 1 thread:

```
smartspice -PS 1
```

By default, SmartSpice requests the maximum CPU for the solver.

21.3 BCG (BiCGstab)

The new iterative solver BiCGstab (named BCG) has been implemented like dynamic loaded library (solver_bcg) and plugged-in to SmartSpice. The BCG solver implements the preconditioned version of the Bi-Conjugate Gradient Stabilized method.

The name of the new solver is "bcg".

To use BCG iterative solver, .OPTION SOLVER=BCG should be specified in the netlist.

There are three options related to BCG solver:

- .OPTION BCG_PRECONDITIONER = ILK | AMP: Specifies the preconditioner that needs to be used for the iterative method specified. Default value is AMP (type - string).
  - ILK preconditioner implements incomplete LU factorization with level of fill dropping. It's linked to the parameter fill level. The correspondent option to set this parameter is BCG_FILL_LEVEL.
  - AMP preconditioner implements incomplete LU factorization with dual truncation (ILUT). It's linked to the parameter fill ratio. The correspondent option to set this parameter is BCG_FILL_RATIO.
- .OPTION BCG_FILL_LEVEL = 0, 1, 2, 3,..., 20: Specifies the fill level for the preconditioner ILK. Default value for BCG_FILL_LEVEL is 5 (type - integer). The values for the BCG_FILL_LEVEL start from 0 and should end at about 20: 0, 1, 2, 3,..., 20. It is recommended to start with the default value first and if the accuracy is poor to increase the value.

- `.OPTION BCG_FILL_RATIO = 0.1, 0.01, 0.001,...`: Specifies the fill ratio for the preconditioner `AMP`. Default value for `BCG_FILL_RATIO` is 0.001 (type - real).). It is recommended to start with the default value first and if the accuracy is poor to progressively decrease it - `0.01, 0.001, 0.0001` etc.

21.4 Extended Precision Solvers

The solvers have been implemented with extended precision starting from SmartSpice version 4.5.5.C.

To run SmartSpice using a solver with the required precision (N) the following command line syntax should be used:

```
% smartspice -solverprecision <N>
```

where N is the required N-bit precision. By default, N is equal to 64.

On Linux for XMS and SPEEDS solvers:

```
N = 64, 80, 128, 160, 256, or 320
```

On Windows:

```
N = 64
```

21.5 Solver Orderings (XMS and SPEEDS solvers)

The orderings that can be chosen for direct XMS and SPEEDS solvers with option `ORDERING` (see Section 21.1 Matrix/Solver Options.) are divided in two categories:

1. Fill-reducing - reduces the number of fillins in L and U produced by the LU factorization:
   - `AMD_SUM` - approximate minum degree
   - `RCM` - reverse Cuthill MacKee
   - `MD` - minimum degree
   - `ND` - nested dissection
2. Diagonal filling - orders the matrix in such a way so that there are no zeros on diagonal:
   - `MNA` - modified node admittance,
   - `MPD` - ordering from Duff and Koster,
   - `ZFD` - ordering based on bi-partite graph maximal matching.

The default orderings for direct solver XMS are: `MNA`, `AMD_SUM`.

The default orderings for direct solver SPEEDS are: `MNA`, `AMD_SUM`, `MPD`.

Chapter 22
Rubberband

22.1 RubberBand GUI Control Elements

The SmartSpice Rubberband feature allows you to select and modify model, instance and `.PARAM` parameters, and interactively watch in SmartView how the simulation changes in real time.

SmartSpice Rubberband is invoked from the **Tools→Rubberband** menu or from the **Rubberband** button in the Toolbar. The Rubberband feature becomes available after the circuit has been sourced.

Rubberband mode of SmartSpice supports multiple analyses: FFT, FOUR, NOISE, NET, PZ, OP, AC, DC, TRAN, S-parameters, TRAN/NOISE, TRAN/jitter, parametrization of devices, entire functionality of `.measure` statement. Netlist may contain multiple analyses, expressions and `.let` statements.

A Rubberband license is required for Rubberband operation. The License is checked out when the Rubberband dialog is invoked, and released when the dialog is closed. However, in SmartSpice200 the Rubberband feature does not require a license.

How to Invoke Rubberband

1. Start SmartSpice in GUI mode, or type the command `smartspice`.
2. Source the input deck.
3. Press the **Rubberband** button in the Toolbar.

Figure 22-1 SmartSpice Main Window With the Rubberband Button

4. Initial run will be started and SmartView will be invoked.
5. Nominal waveforms correspondent to the original netlist settings will be displayed in green.
6. Select model, device or global parameters for Rubberbanding.
7. Press the **Add Selected** button in the Rubberband dialog.
8. Move the Rubberband slider from the Sliders tab. The Rubberbanded waveforms will be displayed in red.

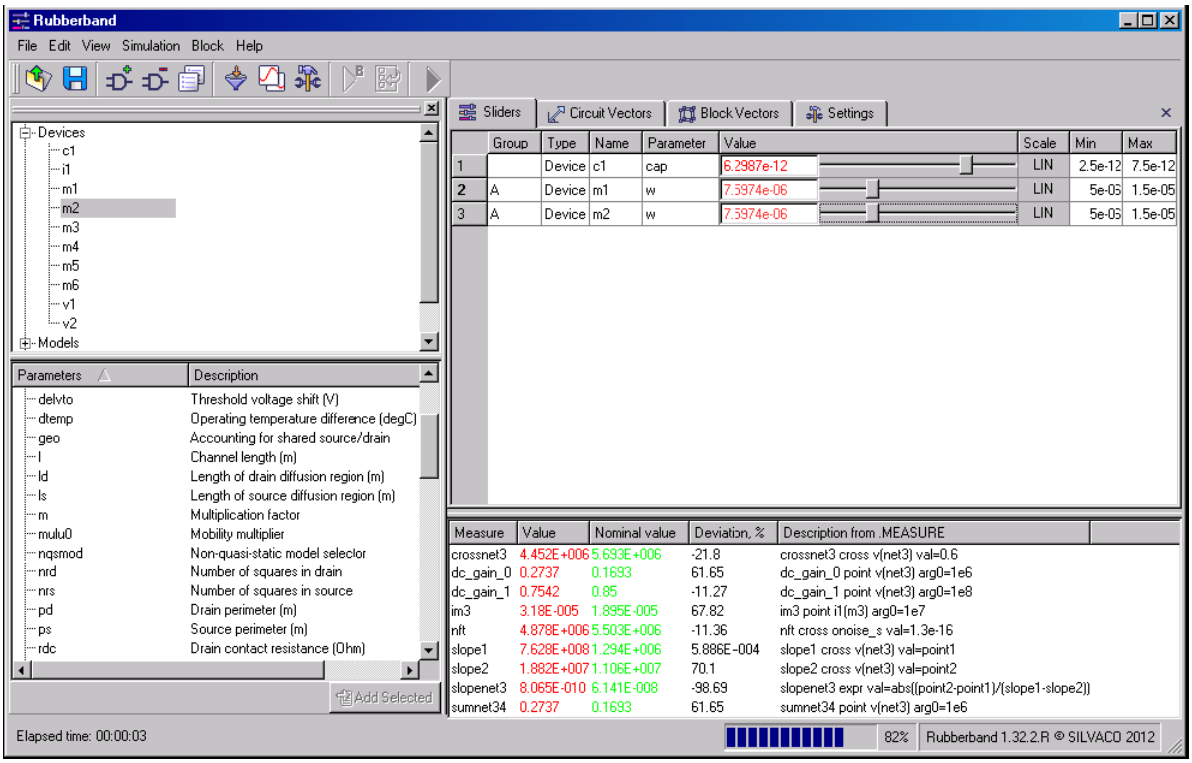


Figure 22-2 SmartSpice Rubberband Feature Dialog with Sliders Selected Tab

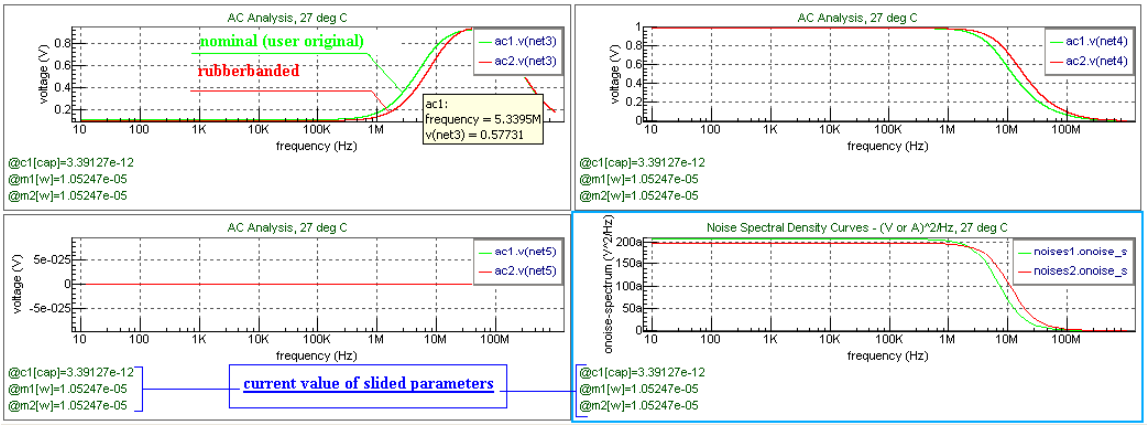


Figure 22-3 SmartView Application With Rubberbanded Waveforms

Rubberband Menu

Provides access to all Rubberband functionality. Enabled functionality may vary according to the Rubberband state. As each menu item is selected, a description of the menu action will be displayed in the status bar.

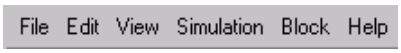


Figure 22-4 SmartSpice Rubberband Menu

Rubberband Toolbar

The toolbar action icons are shortcuts to operations contained within the menus described above. A tool-tip phrase will be displayed when the mouse pointer is held over each toolbar icon to explain its purpose.

Figure 22-5 SmartSpice Rubberband Toolbar

Rubberband Menu Items

- **Open Session:** Loads a Rubberband session from the user specified file.
- **Save Session:** Saves a Rubberband session into the user specified file. SmartSpice saves in the user specified file:
  - Sliders configuration from tab "Sliders"(min/max/current value/scale)
  - Status of "Filter Elements" dialog
  - Layout of Viewer from tab "Settings"
  - Vectors from tab "Block vectors"
- **Exit:** Closes the Rubberband dialog and releases the licence.
- **Add Device:** Adds a new device to the circuit.

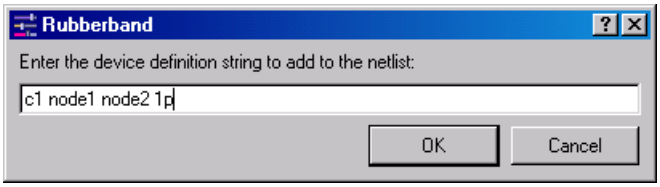


Figure 22-6 SmartSpice Rubberband add device dialog

- **Remove Device:** Removes selected device from the circuit.
- **Preferences:** Opens SmartSpice Rubberband preferences dialog.
- **Filter Elements:** Displays the Rubberband Filter Elements dialog, which allows you to display only R, C, L and K parasitic devices on device tree control. You can modify parasitic thresholds from the Rubberband Preferences Dialog (see Section 22.6“SmartSpice Rubberband Preferences Dialog” for more details). Also, see Chapter 3 Statements, “RubberBand”.
- **Devices/Models:** Shows the models, devices and parameters tree control
- **Sliders:** Shows the sliders tab

- **Circuit vectors:** Shows the circuit vectors tab.
- **Block vectors:** Shows the block vectors tab.
- **History:** Shows Rubberband history dialog.
- **Settings:** Shows the settings tab.
- **Toggle Measures:** Shows/hides circuit measures table.
- **Run simulation:** Run a new simulation.
- **Disable slider simulation:** When selected, a simulation will not start while a slider is moving. To perform a simulation press the **Run simulation** button.
- **Simulate block:** Allows you to Rubberband only this selected cell (see Section 22.8“Rubberband Block Simulation Feature” for more details).
- **Back to the full circuit:** Go back to the full circuit (see Section 22.8“Rubberband Block Simulation Feature” for more details).
- **About:** Displays Rubberband about dialog.
- **Models, devices, and parameters tree control:** Displays all available input deck model, device and parameter type names. You can select them for Rubberbanding.
- **Simulate block context menu:** Can be invoked from devices tree control by right clicking the mouse button on the cell name. Allows you to Rubberband only this cell (see Section 22.8“Rubberband Block Simulation Feature” for more details).

Figure 22-7 SmartSpice Rubberband Simulate Block Context Menu

- **Parameters list box:** Displays all model, device or input deck parameter names you can select for Rubberbanding.
- **Add Selected:** Moves selected parameters to the Rubberband sliders control for future Rubberbanding.
- **Elapsed time:** Displays Rubberband simulation elapsed time.
- **Progress bar:** Displays Rubberband simulation progress.
- **Sliders tab:** Displays Rubberband table control, which allow you to perform Rubberbanding.
- **Circuit vectors tab:** Displays available circuit vectors to plot.
- **Block vectors tab:** Displays available block vectors to plot (see Section 22.8“Rubberband Block Simulation Feature” for more details).
- **Rubberband settings tab:** Displays Rubberband settings.

Rubberband Filter Elements Dialog

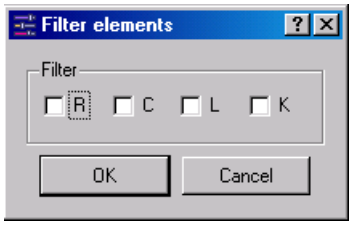


Figure 22-8 SmartSpice Rubberband Filter Elements Dialog

- **R:** Leave only R parasitic devices.

- **C:** Leave only C parasitic devices.
- **L:** Leave only L parasitic devices.
- **K:** Leave only K parasitic devices.
- **OK:** Apply parasitic filter to the devices tree control.
- **Cancel:** Do not apply parasitic filter.

Circuit Measures Table

Measure	Value	Nominal value	Deviation, %	Description from .MEASURE
crossnet3	6.416E+006	5.736E+006	11.86	crossnet3 cross v(net3) val=0.6
dc_gain_0	0.1581	0.1693	-6.591	dc_gain_0 point v(net3) arg0=1e6
dc_gain_1	0.8514	0.85	0.1643	dc_gain_1 point v(net3) arg0=1e8
im3	1.975E-005	1.895E-005	4.267	im3 point i1(m3) arg0=1e7
nft	6.217E+006	5.503E+006	12.97	nft cross onoise_s val=1.3e-16
slope1	1.457E+006	1.293E+006	12.66	slope1 cross v(net3) val=point1
slope2	1.24E+007	1.122E+007	10.52	slope2 cross v(net3) val=point2
slopenet3	5.481E-008	6.043E-008	-9.286	slopenet3 expr val=abs((point2-point1)/(slope1-slope2))
sumnet34	0.1581	0.1693	-6.591	sumnet34 point v(net3) arg0=1e6

Figure 22-9 SmartSpice Rubberband Circuit Measures Table

- **Measure:** Measure name.
- **Value:** Current measure value.
- **Nominal value:** Nominal measure value.
- **Deviation, %:** Deviation between current and nominal measure values.
- **Description from .MEASURE:** Measure line from the input deck.

Sliders Tab

- **Column with numbers:** Displays raw number. Also allows you to reset a parameter value or delete a parameter from the list for Rubberbanding.
- **Sliders tab context menu:** Can be invoked from the column with numbers control by right clicking the mouse button on it.
- **Group:** Displays group title.
- **Type:** Displays parameters types (**Device**, **Model** or **Parameter**).
- **Name:** Displays device, model or parameter type names.
- **Parameter:** Displays parameter names.
- **Value:** Displays current parameter values, which may be modified and will be displayed in red color. You may edit that GUI element.
- **Rubberband slider:** Allows you to modify the parameter value and interactively watch in SmartView how the simulation changes in real time.
- **Scale:** Allows you to change Rubberband sliders scale (linear or logarithmic).
- **Min:** Displays minimum parameter values, which can be modified. You may edit that GUI element.
- **Max:** Displays maximum parameter values, which can be modified. You may edit that GUI element.

Sliders Tab Context Menu

Figure 22-10 SmartSpice Rubberband Sliders Tab Context Menu

- **Reset selection:** Resets all selected Rubberbanding parameters to their initial values since input deck sourcing.
- **Reset all:** Resets all Rubberbanding parameters to their initial values since input deck sourcing.
- **Delete selection:** Delete all selected Rubberbanding parameters from the Sliders tab.
- **Delete all:** Delete all Rubberbanding parameters from the Sliders tab.
- **Ungroup selection:** Allows you to ungroup selected sliders. The group title will be removed form the “Group” column. Sliders will behave as normal Rubberband sliders.
- **Ungroup all:** Allows you to ungroup all sliders.
- **Join group:** Allows you to add selected sliders to the existing group. The “New group” dialog will appeared to select group title.
- **Group selection:** Allows you to group selected sliders. The New Group dialog (Figure 22-11) will appear to select a group title. The group title will be shown in the “Group” column of the sliders tab. All sliders in a group change their values at the same time. To create a group of selected sliders press **Ctrl** and click on the slider number with the left mouse button, (another way is to hold left mouse button go via a number of sliders) then right click mouse button and select “Group selection” from the context menu.

New Group Dialog

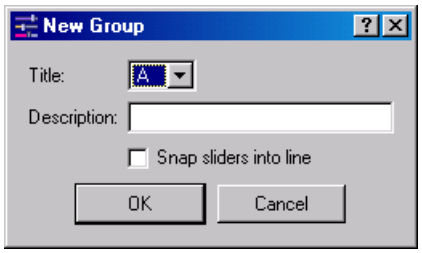


Figure 22-11 SmartSpice New Group Dialog

- **Title:** Allows you to select group title.
- **Description:** Allows you to enter group description.
- **Snap sliders into line:** When selected, all sliders in the group will have the same value.

Circuit Vectors Tab

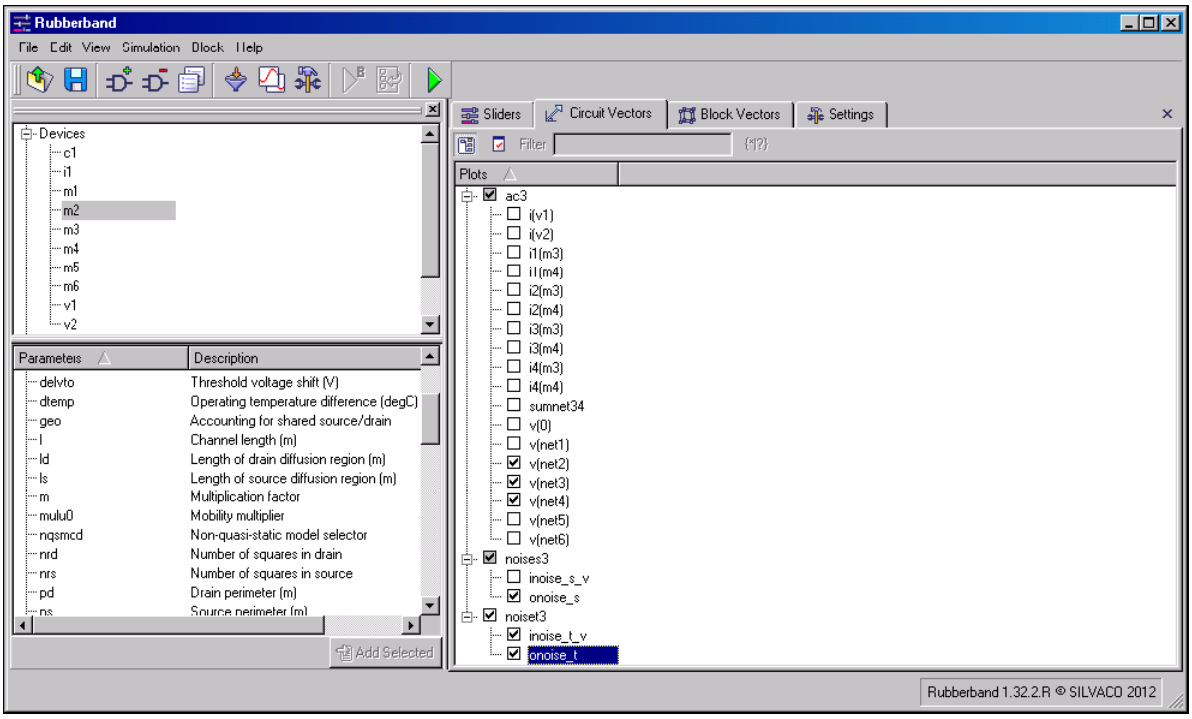


Figure 22-12 SmartSpice Rubberband Feature Dialog with Circuit Vectors Selected Tab

- **Plots:** Displays available circuit vectors to plot.
- **View mode:** Tree view or list view.
- **Filter:** You can apply regular expressions on vectors in list view mode. It is recommended to keep a few vectors in circuit vectors tab.

Rubberband Settings Tab

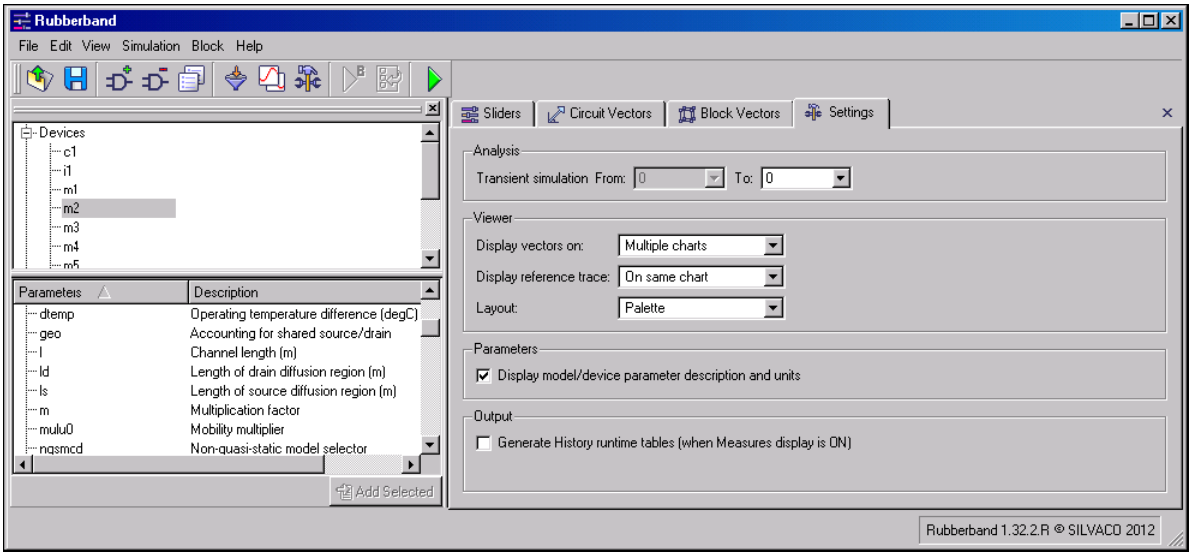


Figure 22-13 SmartSpice Rubberband Feature Dialog with Rubberband Settings Selected Tab

- **Transient From/To:** Modify from and to values for the Rubberband Save/Recovery feature (see Section 22.7“Rubberband Save/Recovery Feature” for more details).
- **Display vectors on:** This combo box has two options: **Multiple charts** and **Single chart**. By default, **Multiple charts** is active and forces SmartSpice to plot one waveform in one chart. If **Single chart** is selected, SmartSpice plots all waveforms in one chart.
- **Display reference trace:** This combo box has three options: **On same chart**, **On separate chart** and **None**. By default, **On same chart** is active. SmartSpice plot reference traces the waveform in the same chart together with Rubberband waveform. If **On separate chart** is selected, SmartSpice creates new chart for reference trace waveform. If **None** is selected, SmartSpice does not plot reference trace waveform.
- **Layout:** Selects SmartView charts layout. You can select Page, Horizontal, Vertical, Palette or Tile layout.
- **Display model/device parameter description and units:** If selected, SmartSpice will display on parameters list box description and units for all model/device parameters.
- **Generate History runtime tables (when measures display is ON):** If selected and measures display is ON, SmartSpice will generate history runtime tables.

Rubberband History Dialog

History

Sliders Configuration 0

	SLIDER(Device):c1/scale	.MEAS:crossnet3	.MEAS:dc_gain_0	.MEAS:dc_gain_1	.MEAS:im3	.MEAS:nft	.MEAS:slope1	.MEAS:slope2	.MEAS:slopenet3	.MEAS:sumnet34
1	0.996063	5.757E+006	0.1689	0.85	1.897E-005	5.526E+006	1.298E+006	1.126E+007	6.021E-008	0.1689
2	0.972441	5.886E+006	0.1666	0.8503	1.913E-005	5.662E+006	1.329E+006	1.151E+007	5.896E-008	0.1666
3	0.948819	6.018E+006	0.1643	0.8506	1.929E-005	5.803E+006	1.361E+006	1.175E+007	5.776E-008	0.1643
4	0.912073	6.229E+006	0.1607	0.851	1.955E-005	6.035E+006	1.414E+006	1.212E+007	5.602E-008	0.1607
5	0.875328	6.492E+006	0.1572	0.8515	1.983E-005	6.285E+006	1.473E+006	1.25E+007	5.44E-008	0.1572
6	0.841207	6.765E+006	0.154	0.852	2.01E-005	6.559E+006	1.532E+006	1.302E+007	5.223E-008	0.154
7	0.762467	7.427E+006	0.1469	0.8532	2.076E-005	7.261E+006	1.686E+006	1.432E+007	4.749E-008	0.1469
8	0.686352	8.186E+006	0.1403	0.8547	2.146E-005	8.076E+006	1.868E+006	1.556E+007	4.381E-008	0.1403
9	0.620735	9.035E+006	0.1349	0.8561	2.21E-005	8.98E+006	2.061E+006	1.713E+007	3.981E-008	0.1349
10	0.557743	9.926E+006	0.13	0.8577	2.275E-005	1E+007	2.285E+006	1.878E+007	3.638E-008	0.13
11	0.520997	1.063E+007	0.1273	0.8587	2.315E-005	1.078E+007	2.442E+006	1.974E+007	3.468E-008	0.1273
12	0.5	1.106E+007	0.1258	0.8594	2.337E-005	1.125E+007	2.542E+006	2.051E+007	3.339E-008	0.1258

Figure 22-14 SmartSpice Rubberband History Dialog

- **Plot measures:** Allows you to plot selected measures from the history table. X-axis will be the value of the slider; Y-axis will be the values of the measures.

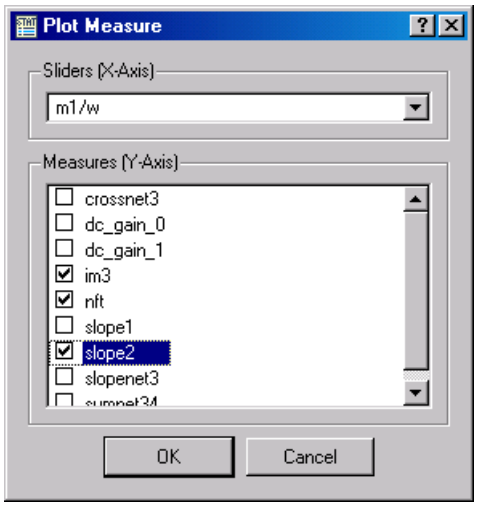


Figure 22-15 SmartSpice Plot Measure Dialog

History tables accumulate information received from continuous simulation runs during Rubberbanding by logging the sliders values and measure results. The new history table will be generated automatically for each unique slider set at the slider releasing event, or appended to an existing table if the slider configuration has not been changed. The tables are automatically named as “Sliders Configuration <id>”, where <id> is a number identifier. In order to keep track of which table has been last updated, the table is shown immediately.

Double-click or press **Enter** at the row on a table to recover a particular sliders configuration and their values which being logged on a row, and switch to the “Sliders” tab. This will trigger the re-simulation event and all measures are recalculated according to the new sliders positions.

Turn on History table generation from the Rubberband “Settings” tab by checking the **Generate History runtime tables** option. The History tables will not be generated if the current deck has no measures or measures display is OFF.

22.2 Rubberband Flow Description

When invoking a Rubberband menu item, the Rubberband license has been taken (in SmartSpice200 the Rubberband feature does not require a license) and the initial run has been performed. After the initial run has been completed, SmartView with nominal run waveforms has been invoked and the Rubberband dialog is displayed.

Select the parameters for Rubberbanding. You can choose model, device or input deck parameters. Then select the model, device or parameter type name by using the tree control. Finally, select the parameter names from parameters list box and press the **Add Selected** button. All selected parameters will appear in the Sliders tab.

To start Rubberbanding, move the Rubberband slider or change the parameter value manually in the **Value** edit box. When a parameter is changed, a new simulation is performed and waveforms are updated in SmartView. Also, you can change the minimum and maximum parameter values.

When the Rubberband dialog is closed, the Rubberband license is released.

22.3 Changes in SmartSpice Behavior Under Rubberband

To speedup Rubberband simulations and avoid malfunctions, SmartSpice behavior has been changed under Rubberband:

1. `.IPLOT` command has been blocked.
2. `.OPTION RAWPTS` has been reset to "0".
3. `.measure` files are not created.
4. SmartSpice calculates measures only when the circuit measures table has been displayed in the Rubberband dialog.
5. `.ST`, `.MODIF` and `.TEMP` statements take only the first loop.

22.4 Rules for Manipulating Sliders

22.4.1 Manipulating Sliders With Zero Initial Value

By default, you cannot Rubberband parameters with zero initial values. Before Rubberbanding, you must specify boundaries (change minimum or maximum value). After that, you can Rubberband this zero parameter.

22.4.2 Manipulating VSRC/ISRC PWL Time Parameters

Example

```
V1 1 2 PWL (0 0 5n 0 10n 5 60n 5 70n 0)
```

In piecewise linear pairs the PWL time parameter must be positive and must be increasing. SmartSpice controls this rules under Rubberband and does not allow you to set a wrong value. If you, for example, try to change `t3` parameter from `10n` to `4n`, SmartSpice will not pass this value to the `V1` instance and Rubberband waveforms will not change.

22.5 Rubberband Restrictions

- Current SmartSpice Rubberband supports only PWL, PULSE and SIN vector parameters for VSRC/ISRC devices. All other vector parameters are not supported yet.

22.6 SmartSpice Rubberband Preferences Dialog

SmartSpice Rubberband preferences can be accessed using the Preferences dialog by selecting **Preferences** from the Rubberband dialog **Edit** menu:

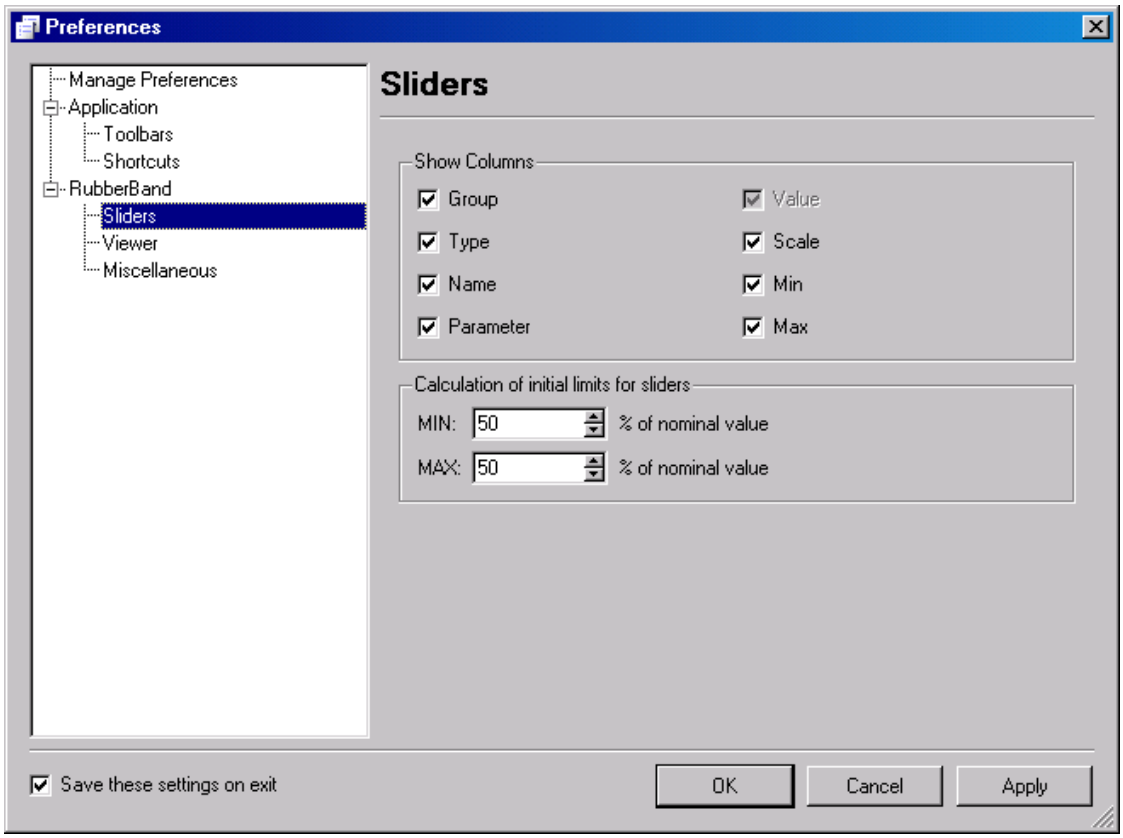


Figure 22-16 SmartSpice Rubberband Preferences Dialog with Sliders Panel

- **Group:** Show the group column on Sliders tab.
- **Type:** Show the type column on Sliders tab.
- **Name:** Show the name column on Sliders tab.
- **Parameter:** Show the parameter column on Sliders tab.
- **Scale:** Show the scale column on Sliders tab.
- **Min:** Show the min column on Sliders tab.
- **Max:** Show the max column on Sliders tab.
- **MIN multiplier coefficient:** Controls the minimum Rubberband value. MinValue = NominalValue - MinCoeff / 100 * NominalValue
- **MAX multiplier coefficient:** Controls the maximum Rubberband value. MaxValue = NominalValue + MaxCoeff / 100 * NominalValue

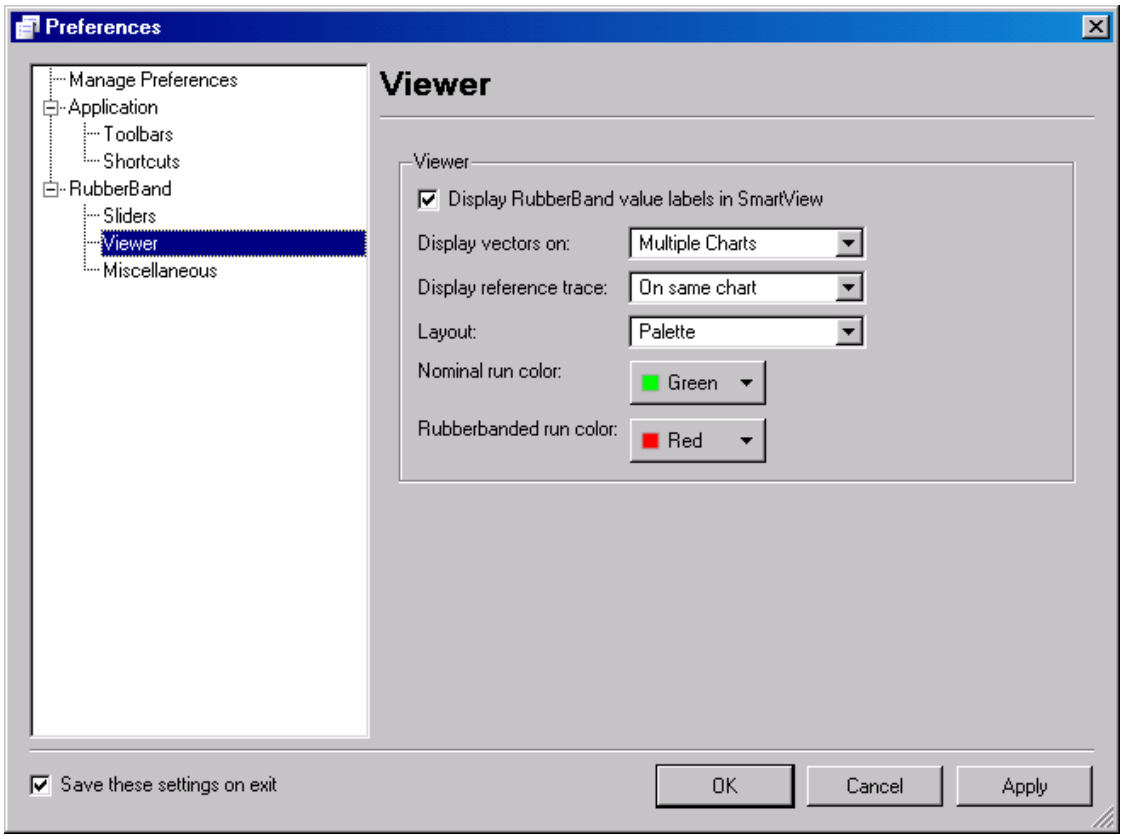


Figure 22-17 SmartSpice Rubberband Preferences Dialog with Viewer Panel

- **Display RubberBand value labels in SmartView:** When checked, parameter information will be sent to SmartView and the parameter label with the current value will appear on the left footer of the chart. Only modified parameters are displayed.
- **Display vectors on:** Default settings for displaying vectors.
- **Display reference trace:** Default settings for displaying reference trace.
- **Layout:** Default SmartView charts layout. You can select Page, Horizontal, Vertical, Palette or Tile layout.
- **Nominal run color:** Color to display nominal run in SmartView.
- **Rubberbanded run color:** Color to display Rubberbanded run in SmartView.

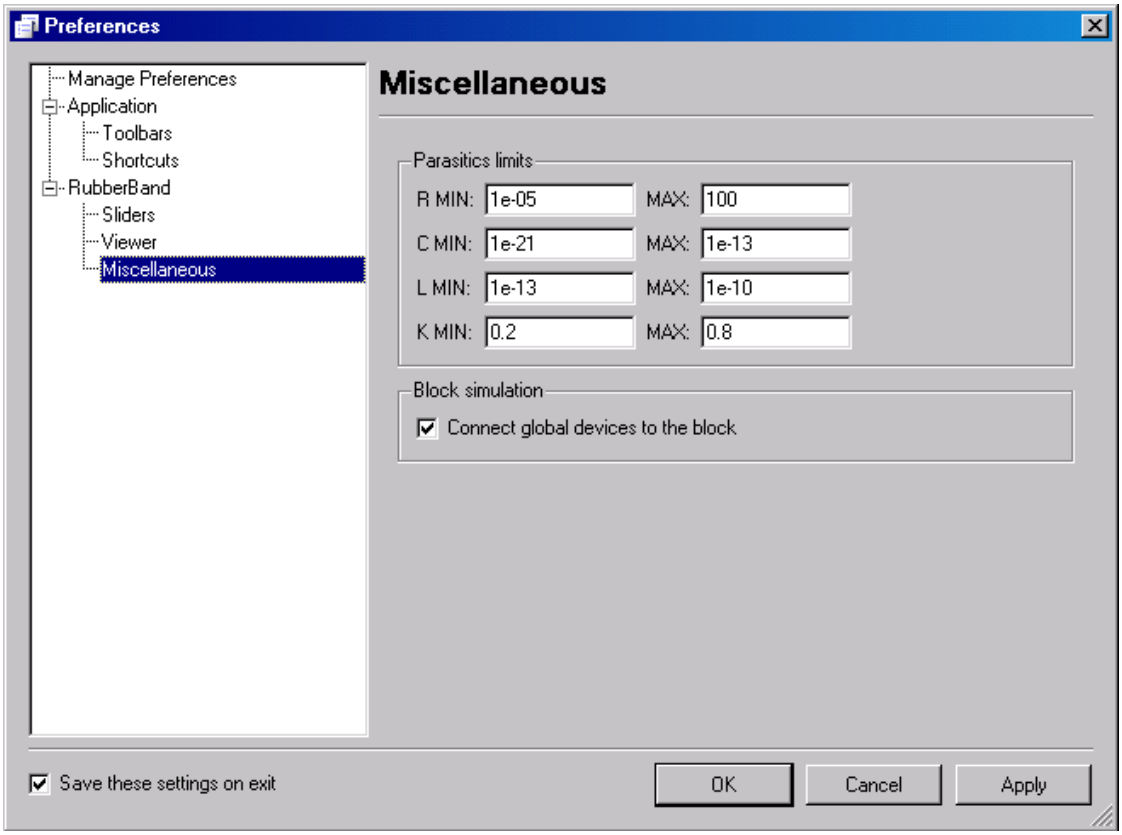


Figure 22-18 SmartSpice Rubberband Preferences Dialog with Miscellaneous Panel

- **Parasitics limits:** Minimum and maximum values for parasitic filter for R, C, L or K-devices.
- **Connect global devices to the block:** If a block has global nodes, SmartSpice connects all top level devices, which are connected those global nodes.

22.7 Rubberband Save/Recovery Feature

Rubberband save/recovery feature saves simulation time and is designed for transient window simulation. An initial run is required before the use of save/recovery. SmartSpice does an initial run automatically when the **Rubberband** button is clicked, or when **Tools→Rubberband** is selected from the menu.

SmartSpice preserves simulation data for recovering point (`FROM` parameter in the transient statement) and then uses the preserved data on sequential Rubberbanded runs.

Before using save/recovery you must setup transient analysis from and to parameters. The `FROM` parameter specifies left boundary of transient window; the `TO` parameter specifies right boundary of simulation window. The `TO` parameter might be bigger than `TSTOP`. If you want to change `FROM` the value, restart the Rubberband initial run.

The `FROM` and `TO` parameters are active only in the Rubberband mode, and does not affect regular simulation.

Example

```
.param  b=2n
.tran 5n  500n  from=270n  to="420n+b"
```

SmartSpice will reuse simulation data at point 270n and end up at 422n in the Rubberband mode.

Multipoint save/recovery feature is activated using new transient parameters FROM_LIST and FROM_STEP in .TRAN statement.

Syntax

```
.TRAN ...
+ <FROM = val TO=val > |
+ <FROM_LIST = num_values val1 ... valN TO=val > |
+ <FROM_STEP = start stop step TO=val >
```

When the parameters FROM_LIST and FROM_STEP are specified in a .TRAN statement the GUI element FROM in the Rubberband dialog becomes enabled, and you may enter any recovering timepoint or select a predefined timepoint from the droplist. SmartSpice searches the nearest predefined recovering timepoint and restarts the Rubberband simulation from it. A single recovering point is specified using the FROM parameter in a .TRAN statement; the GUI element FROM is disabled. A simulation will restart from the same timepoint. The transient parameters num_values and step are designed for flexibility and a more accurate simulation for the restarted simulation. Large values of the mentioned parameters may cause memory and performance overhead. Keep those values to a reasonable level.

Example

```
.TRAN 1ns 100ns FROM_LIST=3 10ns 15ns 17ns
.TRAN 1ns 100ns FROM_STEP=10ns 16ns 3ns
```

In the first example, three save/recovery points at 10ns, 15ns and 17ns will be generated and then will be available for restarted simulation.

In the second example, three timepoints at 10ns, 13ns and 16ns will be used in Rubberband save/recovery.

22.8 Rubberband Block Simulation Feature

Rubberband block simulation feature allows you to Rubberband a part of the circuit. To select a block for Rubberband, right click the mouse button on the cell name in the device tree control, and then select **Simulate block** from the context menu. After that a new block circuit will be created and available for Rubberbanding. You can go back to the full circuit by pressing the **Back to the full circuit** button.

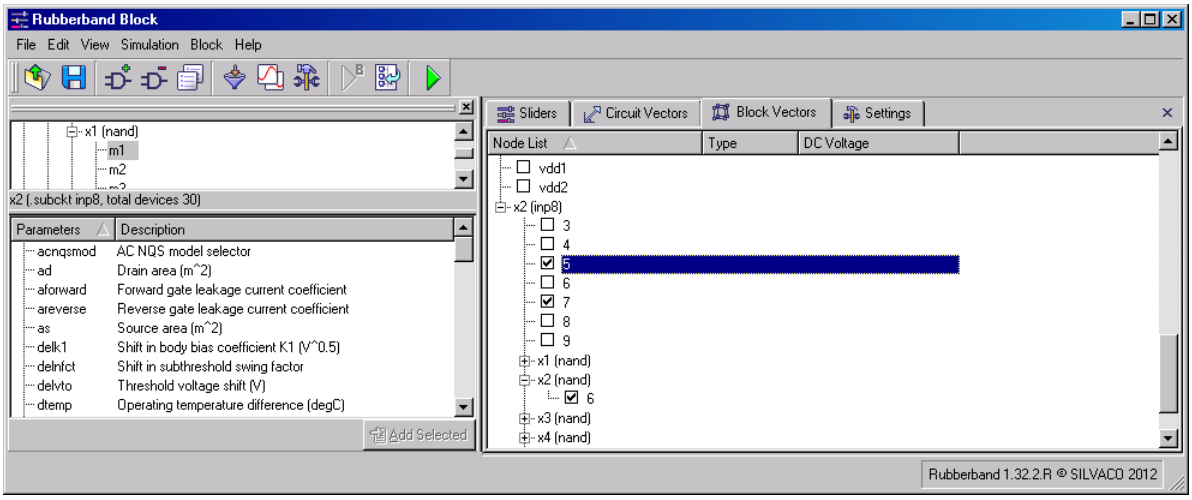


Figure 22-19 SmartSpice Rubberband Feature Dialog for Block Simulation with Block Vectors Selected Tab

- **Node list tree control:** Displays available block nodes to plot.
- **Type:** Displays “Terminal” for block terminal nodes (see Section 22.8“Rubberband Block Simulation Feature” for more details).
- **DC Voltage:** Displays DC voltage for block terminal nodes (see Section 22.8“Rubberband Block Simulation Feature” for more details).

For viewing waveforms that characterize block internal nodes, use the **Block Vectors** tab. The column **Type** marks up terminal nodes of the block, and **DC voltage** displays the DC value of that terminal. DC value is taken from a nominal run of the entire circuit.

Block simulation does a nominal run and the Rubberbanded run.

The Rubberband dialog changes the caption on **Rubberband Block** and after block selection displays the name of the block and the number of devices that form that block.

Chapter 23
Expression Accelerator (EAC) library

Expression Accelerator Library can be used by SmartSpice to accelerate expressions calculation. Expressions include those defined in .MODEL parameters, .PARAM parameters and instances.

23.1 When to Use Expression Accelerator

The Expression Accelerator Library has an advantage at the input deck sourcing stage and at the simulation stage. The following are a list of conditions that suggest using EAC. Numbers are a guideline, and the benefit can even be seen in lower values.

- Deck contains a high amount of expressions in .MODEL card parameters, as well as a high amount of these .MODEL cards used (>100 expressions; multiple used model bins)
- Deck has a low amount of expressions but actual expressions are large (>10 operators within expression, func calls, ternary operators, etc.)
- Deck contains a considerable amount of macro-models that have very deep parametric dependency of expressions
- Deck contains a parametric analysis or any another kind of feature that results in numerous runs (sweep, modif, alter, etc.). Expressions are reevaluated before each step/run and benefit from acceleration (>5 steps/runs)
- Deck contains RCLEGF devices with run-time expressions
- Deck contains high amount of RCLEGF devices with value set as non-runtime expression (>100 devices)
- Deck has a high amount of .PARAM expressions with deep parametric dependency passed to .SUBCKT cards, .MODEL cards, etc. (>100 .PARAM expressions)

Any and all of the above may suggest usage of Expression Acceleration. It can be used all the time as well - there is no downside of using it.

23.2 Expressions Accelerator (EAC) Converter Script

Expressions Accelerator Converter Script is a support utility to automate expression accelerator library generation. Script converts input file(s), replacing expressions surrounded by single quotes with function calls. At the same time it creates required sources and project files to build EAC library.

Script also modifies main input files with required .option statements, and converts include and library file references to point to the newly created modified files.

For more information see the sdk/EACscript package.

23.3 Library Structure

Library is consisted of three source files: `EAC.h`, `EAC.c` and `EACexpressions.c`.

EAC.h

The file `EAC.h` contains generic type definitions, and accelerated expression functions definitions. When a new function is added, its definition should be added in this file under `/* Expression accelerator functions */` comment.

EAC.c

The file `EAC.c` is the library main code file. It contains expression accelerator functions list and library initialization function `EACinit()`. The initialization function is not to be modified; SmartSpice relies on it and initializes library through it. When new functions are added, their info should be added into functions list. This list is a static array `ExpressionFuncs` which contains function name, pointer to function and number of expected arguments.

Example

```
{"EACexample", &EACexample, 2}
```

Note: Even if the function takes no arguments, set 1 as the number of arguments; when calling it from SmartSpice, put 0 as single argument. This is required because SmartSpice doesn't allow user defined functions with zero () arguments and ignores them.

Note: The last entry in `ExpressionFuncs` should always be `{"", NULL, 0}` because SmartSpice uses this syntax to determine end of function list.

EACexpressions.c

The file `EACexpressions.c` contains bodies of actual expression accelerator functions. All functions should comply to the following rules:

- they all should return double
- they all should get (`double* Args`) as arguments, where `pArgs` - array of doubles
- function names should be lowercased

Input arguments will be auto constructed by SmartSpice from calls like `func(x, y, ...)`.

Example

Input deck card:

```
.model ... lmin='sqrt(9e-7*par1+par2)' ...
```

should be changed to:

```
.model ... lmin='lmin_func(par1, par2)' ...
```

In the library, `lmin_func` should be declared and defined:

```
double lmin_func(double* Args);
```

In the function body you'll have access to `par1` through `Args[0]` and to `par2` through `Args[1]`.

The function body should be:

```
double lmin_func(double* Args)
{
  double par1 = Args[0];
  double par2 = Args[1];
  double result = sqrt(9e-7*par1+par2);
  return (result);
}
```

Note: The function name can be any you like, there is no constraint on that; just make sure its unique.

The following example is the sample of a circuit which contains a resistor model with the parameter cox defined as expression.

Example

```
vin in 0 dc 5 ac 1
r1  in out ronly 1k l=2m w=1m
c1  out 0 1p
.param pcox = 1e-8
.model ronly r (
+ cox='sqrt(pcox*pcox)' )
.tran 0.1n 5n uic
.print v(out)
.option brief nomod capdc=1
.end
```

The following example is the modified circuit which contains a resistor model with the parameter cox defined as the expression accelerator library function call eac_____ronly_cox(). The .OPTIONS eacflag and eacpath have to be added to activate expression accelerator flow.

Example

```
vin in 0 dc 5 ac 1
r1  in out ronly 1k l=2m w=1m
c1  out 0 1p
.param pcox = 1e-8
.model ronly r (
+ cox='eac_____ronly_cox(pcox)' )
.tran 0.1n 5n uic
.print v(out)
.option brief nomod capdc=1
.option       eacflag=1       eacpath='libEAC/build/lib/optimize/
res_expression_1_0_0_R'
.end
```

To see EAC.h, EAC.c and EACexpressions.c use the sdk/EAC package.

23.4 Directory Structure

Library Unix and Windows projects are located in `./`:

- `./EAC.sln` - Windows solution
- `./EAC.vcproj` - Windows project
- `./Makefile` - Unix makefile

Source files are located in `./src`:

- `./src/EAC.c`
- `./src/EAC.h`
- `./src/EACexpressions.c`

Build files will be located in `./build`:

- `./build/objects/<platform>/<optimize|debug>` - compiled object files
- `./build/lib/<optimize|debug>/<libname>_<libversion>/<platform>` - final library

Notes

You can have several `EACexpressions.c` files, just name them the way you like, and modify `Makefile/EAC.vcproj` files accordingly to reflect for new sources.

To select build mode on Unix, add `CONFIG=optimize|debug` commandline switch; default is optimize build.

Supported platforms are:

- Red Hat Linux 4, 5, 6 (64 bit)
- Windows XP (32 bit), Vista (32 bit or 64 bit), 7 (32 bit or 64 bit)

`<libname>` and `<libversion>` can be controlled through `EAC.vcproj` post build event on Windows and through Makefile on Unix platforms.

Actual library file is and should always be named `libEAC.so` or `libEAC.dll`; there is no control on that.

23.5 Connecting the Library to SmartSpice

To connect built expression accelerator library to SmartSpice, the following option statement should be added into the input deck:

```
.OPTION EACFLAG=1 EACPATH='path_to_eac_library_directory'
```

You should not give a full path to the actual `.so` or `.dll` library. Instead, in the supply path up to and including `<libname>_<libversion>`, SmartSpice autoselects a platform from there and loads the library according to the platform.

Example

If you build Windows libraries in:

```
/home/user/sdk/EAC/1.0.0.R/build/lib/optimize/my_1_6_3_R/x86-NT
```

then just set:

```
.OPTION EACFLAG=1
+          eacpath='/home/user/sdk/EAC/1.0.0.R/build/lib/optimize/
myEAC_1_6_3_R'
```

SmartSpice will select the library automatically.

23.6 Expression Accelerator (EAC) Automatic Flow

SmartSpice provides automatic EAC parsing flow, and automatically extracts derivatives for runtime expressions, runs perl scrips which generates new netlist files and EAC library source files, makes an EAC library, and loads modified files. SmartSpice also creates an `EACflow.log` file, where it saves EAC flow statistics information. SmartSpice supports EAC script version 1.0.23.A.

To activate this feature:

1. Specify the full path to the `EACconvert.pl` script using the environment variable `EACSCRIPT_PATH`.

Example

```
setenv EACSCRIPT_PATH ~/sdk/EACscript/1.0.23.A/EACconvert.pl
```

2. Run SmartSpice with the `-eacflow` command key.

Example

```
smartspice -eacflow
```

SmartSpice is in GUI mode and can source the input deck as normal, and will use the new functionality during simulation.

Note: EAC automatic flow mode is not supported on the Windows platform.

Chapter 24
SmartSpice Distributive Capabilities for Monte Carlo and Alter

24.1 Command Option -mp

Syntax

```
smartspice -mp <n>
```

The command line option -mp allows you to run jobs that contain multiple iterations, in parallel on multiple CPUs. The number of CPUs used can be limited by specifying the number n. If the number n is not specified, SmartSpice will use all the available CPUs during the simulation.

SmartSpice will generate a new circuit for every iteration step, and write out each circuit in the same directory. Then SmartSpice will create extra child processes. Each child process will handle a separate circuit. The parent process waits for the children to finish. If we specify `-mp 4`, then 4 extra processes will be created. The total amount of SmartSpice processes in memory is 5, composed of 4 child processes and 1 parent. SmartSpice will print the number of extra child processes in the output file. After finishing the simulation, the child processes will remove their own circuit input files. Finally, SmartSpice will combine output, raw and measure files from the child processes.

Note: Only transient analysis with sweep data and .dc lin analysis is supported. This feature is not available on the Windows platform.

Example

```
.param val=1

v1 1 0 sin ( 0 val 50e6 )
r1 1 0 1

.probe v(1)
.measure max_v_1 max v(1)
.measure min_v_1 min v(1)

.option nomod nodeck post=2 probe

.tran 1e-9 2e-8 sweep data=val_data

.data val_data val
+               1
+               2
+               3
+               4
.enddata

.end
```

SmartSpice will generate the following temporary files for this example after executing the command:

```
>smartspice -b -mp 4 example.in -o example.out -r example.raw
Input decks: example-1.in, example-2.in, example-3.in, example-
4.in.
Raw files: example-1.raw, example-2.raw,
                 example-3.raw, example-4.raw.
```

These files will be deleted after simulation is complete.

24.1.1 .MODIF Behavior

The .MODIF statement has different behavior under the command option -mp. .MODIF will be activated after each analysis iteration step. By default, SmartSpice activates the .MODIF statement when all analyses of the input deck are completed.

24.1.2 DC Analysis Behavior

The DC analysis has different behavior under the command option -mp. Instead of generating a new circuit for every iteration step, SmartSpice will divide the DC analysis on intervals. the number of intervals corresponds with the number n from the -mp command option.

Example

```
.dc lin 1 10 1
```

If -mp 4 is specified, the DC analysis will be divided on 4 intervals.

```
.dc lin 1 3 1
.dc lin 4 6 1
.dc lin 7 8 1
.dc lin 9 10 1
```

24.1.3 Option SUPPRESSOUTRAW

Option SUPPRESSOUTRAW is used to suppress output and raw files for the child processes.

Example

```
.option SUPPRESSOUTRAW
```

24.2 Parametric Parallel Analyses

24.2.1 Command Option -mps

Syntax

```
smartspice -mps <n>
```

The command option `-mps` allows you to run parallel simulations on multiple CPUs. The number of CPUs can be limited by specifying the number `n`, where `n` is less than the total number of CPUs available. If the number `n` is not specified, SmartSpice will use all available CPUs during the simulation.

Note: This feature is available on all platforms in batch and GUI modes for transient analysis only. All types of sweep are supported. The `.option RAWPTS` is not allowed under `-mps`.

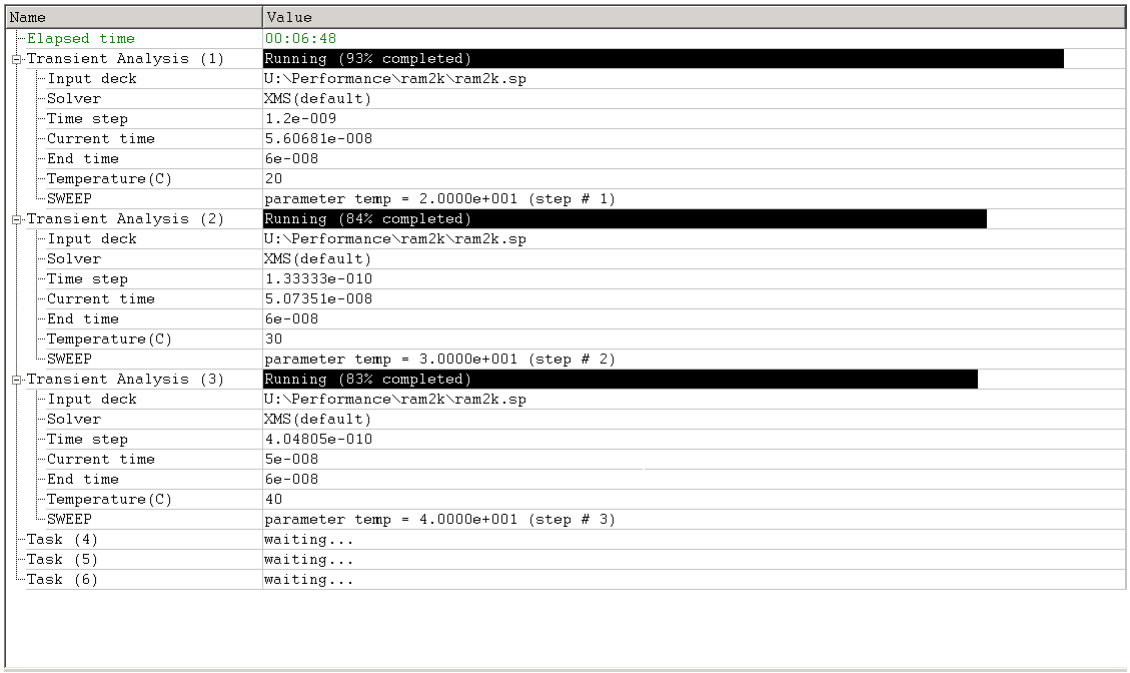


Figure 24-1 Simulation Window

Example

```
.param fre=50e6
.param mult=1
vin my1 0 sin(0 1 'fre')
r1 my1 0 1000
v1 my2 0 5
r2 my2 0 'mult*1000 + 200*v(my1)'
.probe v(my1) i(r2)
.tran 1e-09 50e-08 sweep mult 1 3 1 sweep fre 50e6 53e6 1e6
```

In this example the number of parametric (sweep) steps is 12. On an 8-CPU computer, where the option `'-mps 4'` is specified, SmartSpice creates four circuits, sets swept parameters for each circuit, and runs the simulation in separate threads. When the simulation is done SmartSpice resets swept parameters and performs the next simulation.

24.3 Remote Distribution for Monte Carlo, Sweep and Alter

SmartSpice, by means of the SmartSpiceServer application, is capable of splitting the netlist into separate simulation branches which can be processed in parallel on multiple CPUs on a local machine or/and remote hosts. This will speedup the entire simulation process significantly for most cases. The current implementation supports alters and Monte Carlo statements only.

This feature can work in batch mode or GUI mode to allow you to monitor the process or manipulate it. In this mode SmartSpice launches the child SmartSpiceServer process which is responsible for all the distributive communication with the remote agents and, delegates the data processing tasks to its parent.

Limitations

The current implementation supports:

- Implemented on Windows and Linux only.
- .ALTERs, Sweeps and Monte Carlo statements only;
- Single analysis per netlist is allowed only;
- Due to security scheme implemented in SmartSpice, the local and remote hosts which are to be used must have the system time synchronized within +/- 5 minutes deviation. Otherwise the remote agents will fail to start with security check error.

Password Authentication

To access the remote host and be able to run jobs on it you will need to enter a password athentication. You will be prompted for your password with every host you try to use. If the password is entered correctly and the host is accessable, the athentication information will be saved and retrieved the next time without prompting you. Otherwise, you will be prompted again.

IMPORTANT: In batch mode, in order to be able to enter a password, you must launch SmartSpice in such a way so that the input terminal is properly attached to the program. On Windows, a new command prompt will be created by program, but on UNIX you must start SmartSpice attached to a terminal (without the appended '&' in command line or from shortcut).

After you have all your athentication information for accessing remote hosts entered correctly and stored, you can launch SmartSpice in preffered way.

Batch Mode

To run remote processing in batch mode, start SmartSpice with the `-mpr` command-line switch. In this mode the additional information must be provided in order to make a simulation run without errors. These are:

- Remote agents configuration;
- Input deck;

Remote agent's configuration must be supplied with help of SmartSpice startup file (`.INI`).

```
> smartspice -mpr -startupfile <file name> <input deck>
```

You can setup some attributes in the `.INI` file's control block: the hosts list, amount of agents for each host, core CPU number, solver CPU number, output/raw locations and agents version.

The variables and it's format dedicated to this purpose are:

- remote_hosts = (host1[:amount][:core CPUs] [:solver CPUs] [host2] …)
- remote_version = X.X.X.X
- remote_outpath = <directory name>
- remote_rawpath = <directory name>
- remote_timeout = <0 ... 9999>

Example

```
.control
set remote_hosts = ( localhost=2:2:3 spice1=3:2 dent=8 )
set remote_version = "4.1.37.R"
set remote_outpath = "/home/user/outpath"
.endc
```

where number of instances follows the hostnames right after '=' in the list for remote_hosts variable. It may be followed by the core CPUs number or by the solver CPUs number. No spaces are allowed. If remote_version is omitted, the default one will be used as a rule.

Batch Mode Examples

When the above mentioned information is supplied, the SmartSpice remote processing can be initiated in batch mode as this:

```
>smartspice -mpr <input deck>
```

In this case, SmartSpice will try to produce the number of simulation branches by splitting input deck according to .ALTER and Monte Carlo statements which are found inside.

The snippet of input deck, containing .ALTER statements, which is subject to such processing:

```
…
.param vd=1
.alter
.param vd=2
.alter
.param vd=3
…
```

Example of Monte Carlo statement in input deck:

```
.TRAN STEP=0.1p STOP=100n START=0 SWEEP MONTE=10 PRMC
```

Sweeps statements can be of any type (LIST, LIN, DEC, OCT or .DATA) and contain nested levels in any combinations according to syntax specification for supported analyses types.

Examples

```
.DC data=Data
.DATA Data Parameter1 Parameter2
1 2
2 4
.enddata

.TRAN 1ns 10ns sweep a LIST 4 1000 1100 1200 1300 sweep
data=scale_b
.TRAN 0.1n 50n SWEEP c(cap) OCT 10 500p 1n
```

The output produced by SmartSpiceServer can be found in a .log file in the current directory.

Output

If SmartSpice is started with '-o <path>' command-line switch (the request for stdout to be redirected to the file), SmartSpiceServer will produce it's own log file by appending '.sssvr' part to the base name of supplied path with the same extension (usually '.out'). It is supposed to contain the dump of jobs distribution task alongside with error/warning messages whatever could be useful to monitor the flow of distributed simulation or identify the problems.

The output produced by remote agents can be found at location specified by 'remote_outpath' variable in .INI file or suppressed if no such variable is used. In terms of final representation of remote agent's output several possibilities exist:

- Per agent output (default): Each agent (including the master) will generate its own output file named after it's own internal name (e.g., '@Spice0(1).out'). The disadvantage is that it may contain the output from jobs not in consecutive order - the way they were submitted.
- Per job output: Each job will produce its own output file. The name will be based on the input deck's base name and the job's hierarchical identifier in a simulation tree (e.g.: 'pchip.1.out' - the alter '1' of 'pchip' input deck). This mode is intended for advanced use and may prove to be helpful in certain cases. To enable this mode, start SmartSpice with '-pjo' command line switch:

  ```
  >smartspice -mpr -pjo
  ```

- Per job output combined: In this mode all the output snippets from all the jobs will be orderly combined (possibly in a course of distributed simulation) and the single out file produced. The result file will be the one specified with '-o <path> option'. To enable this mode, start SmartSpice with '-pjoc' and -o command line switches:

  ```
  >smartspice -mpr -pjoc -o combined.out
  ```

All output modes require the 'remote_outpath' variable to be properly set up in .INI file.

GUI Mode

In this mode the SmartSpiceServer's GUI environment is displayed giving you interactive control. The Batch mode description is true for GUI mode operation, except there should be used an additional command line switch: -mprg (not -mpr):

```
>smartspice -mprg [-mprfile <batch script file>] [input deck],
```

where entries in brackets [] are optional.

If startup file and input deck are supplied, SmartSpiceServer will start processing right after initialization. If those are omitted, you can do the things interactively by launching additional agents on specified hosts with customized properties and loading any decks for processing through the program's menu.

For details on how to operate SmartSpiceServer in GUI mode please refer to the SmartSpiceServer User’s Manual.

Chapter 25
Job Queuing Systems

25.1 Using SmartSpice on a Job Queuing system

While Silvaco does not support any particular queuing system many customers have used a variety of queuing systems with SmartSpice Including:

- Oracle Grid (http://www.oracle.com/technetwork/indexes/documentation/index.html)
- Open Grid Scheduler (http://gridscheduler.sourceforge.net/)
- Platform LSF (http://www.platform.com/workload-management/high-performance-computing)

25.2 Running SmartSpice on a Grid

Once you have a grid system installed and functioning the exact method of running SmartSpice will differ between systems but these general rules apply:

a) Write a starter script to launch your simulation making sure SmartSpice will run in silent batch mode

b) Submit your script to the batch system.

For example using Open Grid Engine:

a) Create a script with your SmartSpice command as follows:

```
#!/bin/sh
smartspice -sb -P 1 -PS 1 inputdeck.in -o outfile.out -r
waveforms.raw
```

Where:

- `-sb` forces SmartSpice into silent batch mode
- `inputdeck.in` is your input deck name
- `outfile.out` will contain the run information
- `waveforms.raw` will contain the waveforms

If you want to run with more CPUs increase `-P` and `-PS` from 1 to how many CPU's you want SmartSpice to use.

b) Submit the job to a queue

```
qsub -cwd scriptname
```

Where `scriptname` is the name of the script from a) above.

c) With Open Grid Schedular and Oracle Grid Engine the `qmon` command can be used to interactivly see jobs on the grid or `qstat -u "*"` can be used from the command line.

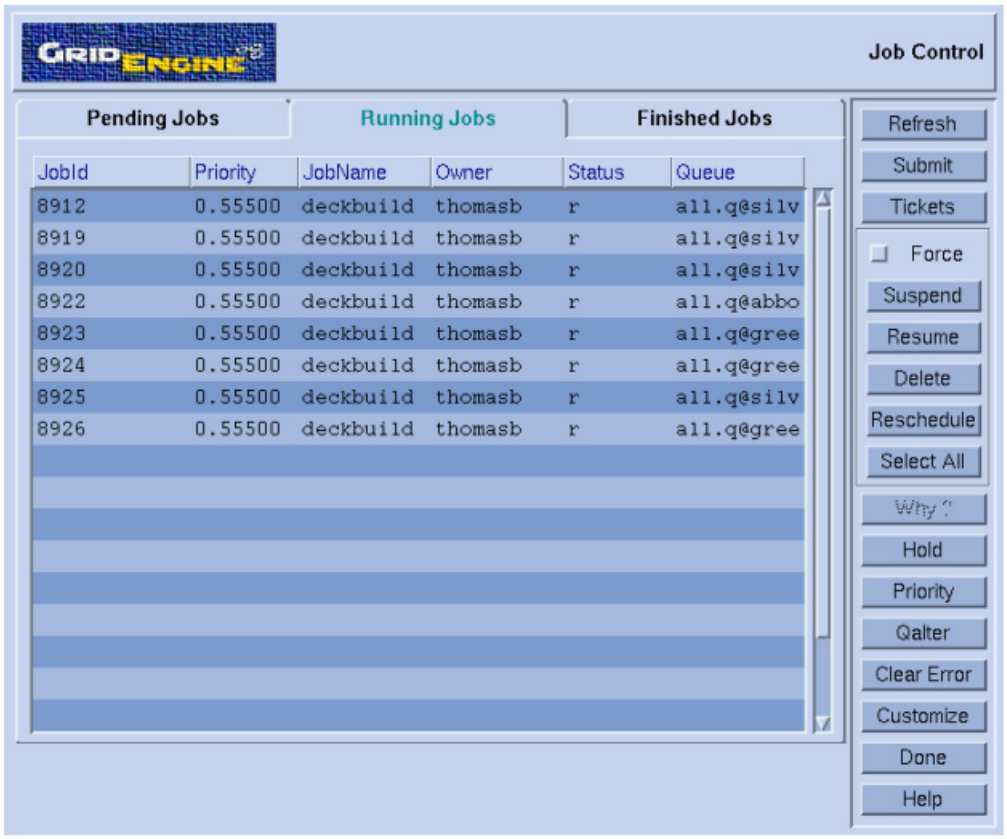


Figure 25-1 Grid-Engine Job Control Panel

Chapter 26
Installing and Using SmartSpice in the Analog Artist Environment

26.1 Introduction

SmartSpice can be run in a spectre compatible mode both as a stand-alone program or in a batch mode under the Cadence Analog Artist environment. Running under the Analog Artist environment a SmartSpice process is spawned in batch mode using the standard Analog Artist GUI. The introduction of SmartSpice into the Cadence environment takes nothing away from the current Analog Artist functionality still allowing full use of Spectre but also adds the capability to use SmartSpice as the spice simulation engine. The binary simulation data is then returned in a format compatible with the Cadence viewer.

Setup

Required user environment settings for the cadence initialization file (`.cdsenv`)

```
spectre.envOpts controlMode string "batch"
spectre.envOpts simOutputFormat string "psfbin"
```

Optional settings in the x-windows initialization file (`.cshrc`)

```
setenv PSF_WRITE_CHUNK_MODE_ON true[1]
setenv SMARTSPICE_V 4.6.5.R[2]
```

Note: 1. For large output files greater than 2GB.
2. For running a specific version of SmartSpice.

Optional settings for SmartSpice initialization (`smartspice.ini`)

```
set viewer = smartview
```

Note: For writing a `.raw` file to view in SmartView rather than a PSF directory for simulation data.

26.2 Installation

Installing SmartSpice to run with Cadence environment requires a functioning SmartSpice installation and a separately functional Cadence environment.

Overview

Installing SmartSpice to run under the Cadence environment involves installing an intelligent starter program into the Cadence installation that runs either the original Spectre program or SmartSpice.

Requirements

1. Working Cadence Environment (supported versions IC5.1, IC6.1., MMSIM70 or later).
2. Working Silvaco SmartSpice installation (2010.00 baseline or later).

Installing SmartSpice as a Spectre replacement requires that Spectre is installed but a Spectre license is not required. This ensures all the Cadence hooks are in place to run Spectre (or SmartSpice in Spectre mode). If the Spectre program is not installed you should first install Spectre into your Cadence environment even if you do not have a License to run it.

You will need to know the path to the Spectre starter program which may be in the Cadence Analog Artist tree or in a separate MMSIMxx tree.

Installation Procedure

Logged in as a user that can write to the cadence installation folder run the command:

```
<silvaco_install_dir>/bin/smartspice -install -spectre
```

This command will search the current PATH to locate the Spectre starter. Once located the binary it launches will be renamed and an intelligent wrapper program will be inserted in its place.

If spectre is not located on the PATH the script can be directed to look in an additional folder use the command:

```
<silvaco_install_dir>/bin/smartspice -install -spectre
<cadence_install_dir>/tools/bin
```

where `<cadence_install_dir>` is the path to your Cadence installation containing Spectre.

Again this command will rename the existing spectre binary and an intelligent wrapper program will be inserted in its place.

Note: The `smartspice -install -spectre` command renames the original Spectre binary and inserts an inteligent starter program in its place. The starter program will call either the renamed Spectre or SmartSpice as directed by the user. To revert your Cadence installation back to its original state the starter program can be removed and Spectre renamed back to its original name.

26.3 Simulation Files

Netlist

Netlist files are located in the `/simulation/cellview/spectre/schematic/netlist` directory just as they are for Spectre. The `input.scs` and `input.scs.in` files are located here. Verilog files will be compiled in a subdirectory called `SilvacoVLG` immediately underneath the `/netlist` directory. Verilog models only need to be compiled once unless the source file is changed.

Plot Files

Plot files are located in the `/simulation/cellview/spectre/schematic/psf` directory as in a Spectre simulation. Transient, DC and AC output is supported in binary format.

If set `viewer=smartview` is added to the `smartspice.ini` file, then the `input.scs.raw` file will be written in the `/simulation/cellview/spectre/schematic/netlist` directory where the `input.scs.in` file is located. However, no Cadence output will be generated.

26.4 Running Under Analog Artist

It is assumed the user is familiar with running a spectre simulation of his design and only the differences to run SmartSpice in a spectre compatible mode will be described.

Once the circuit design has been setup to run a normal spectre simulation the following change can be made:

In **Virtuoso Analog Design Environment** window select menu item **Setup - Environment** and userCmdLineOptions dialog. You can enter the following values:

- `-smart`: Runs SmartSpice in spectre compatible mode.If not set spectre will be run in the normal way.
- `-format psfbin`: Enables Cadence binary PSF data for plotting, back annotation etc.
- `-f psfbin`: Equivalent to the previous value.
- `-format psfascii`:Enables Cadence ascii PSF data for plotting, back annotation etc.
- `-f psfascii`: Equivalent to the previous value.
- `-turbo`: Turns on a faster mode of SmartSpice to get a quicker simulation.
- `+mt=n`: Specifies number of processors (n) for multi-threading.

For typical entry in Analog Artist window, see Figure 26-1.

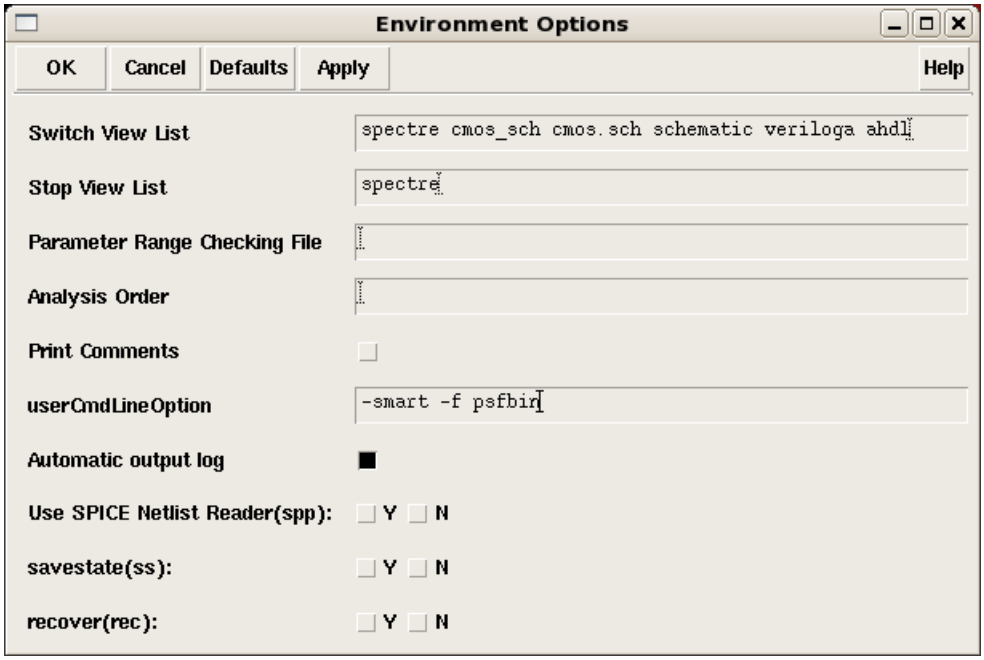


Figure 26-1 Settings to run SmartSpice instead of Spectre

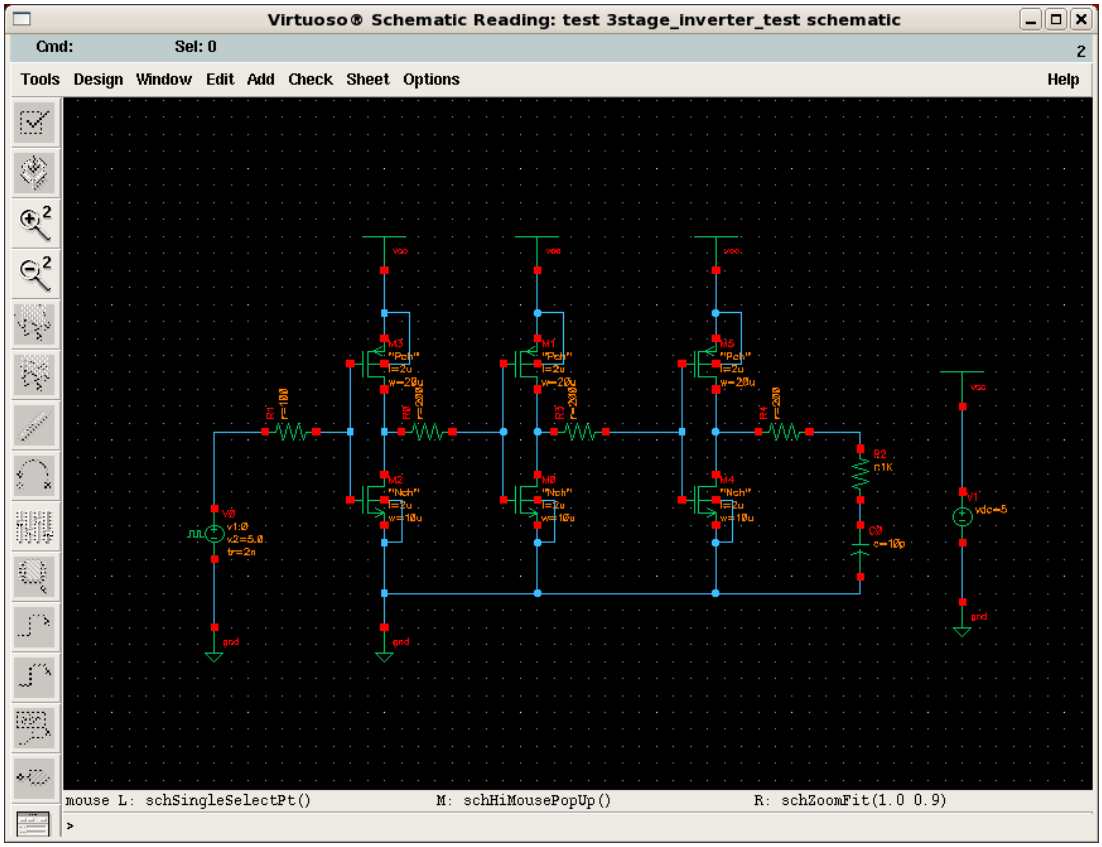


Figure 26-2 Example Test Circuit

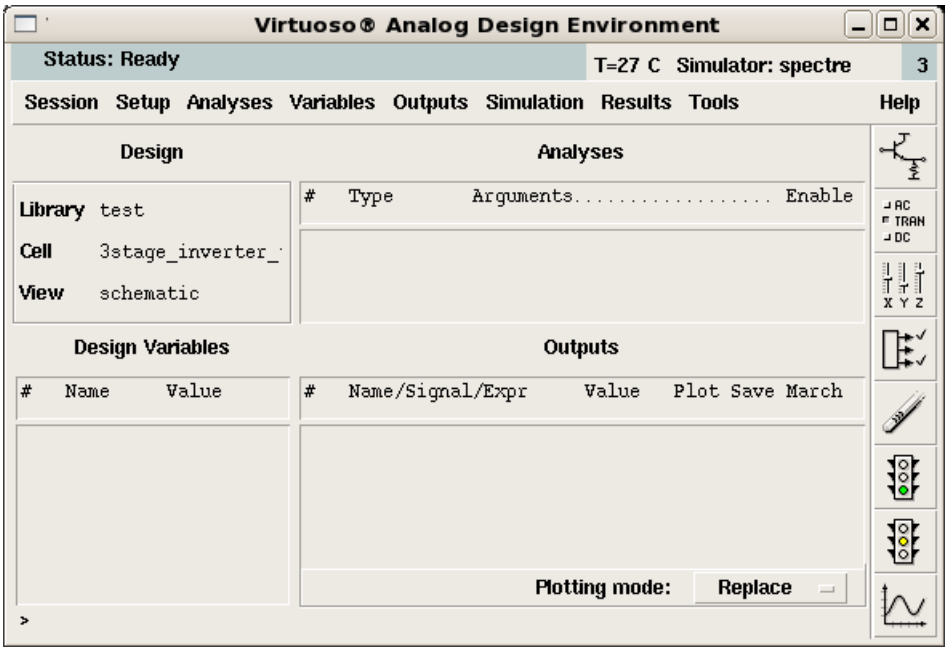


Figure 26-3 Main Simulation Control Window

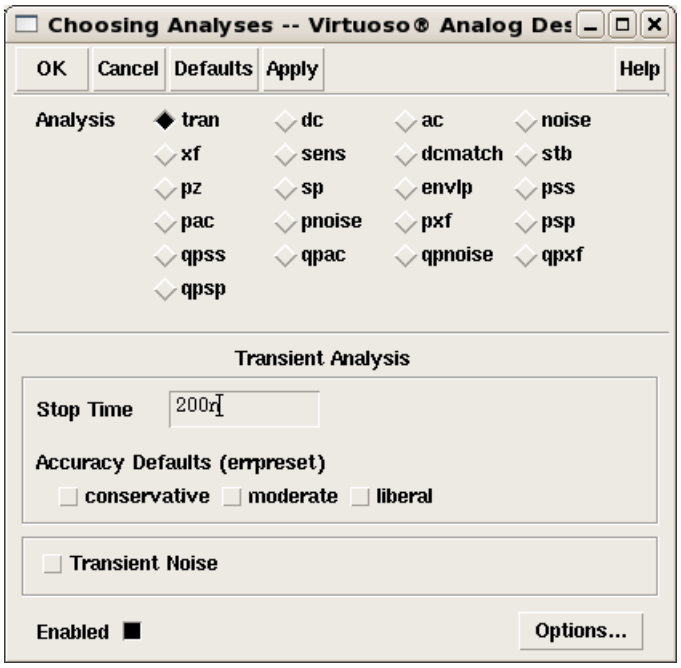


Figure 26-4 Analysis Selection and Setup

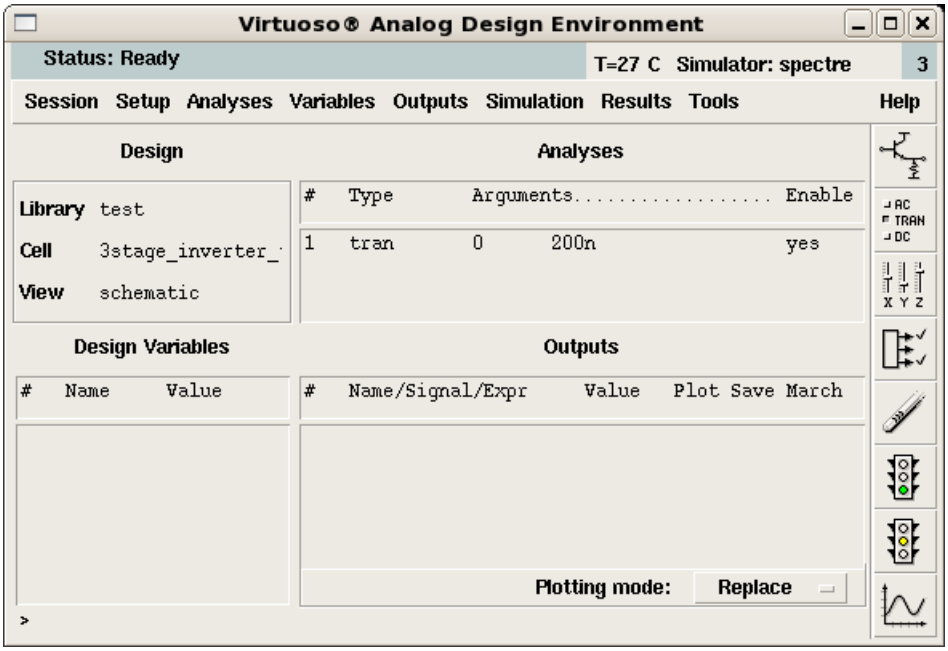


Figure 26-5 Main Control Window Showing Setup

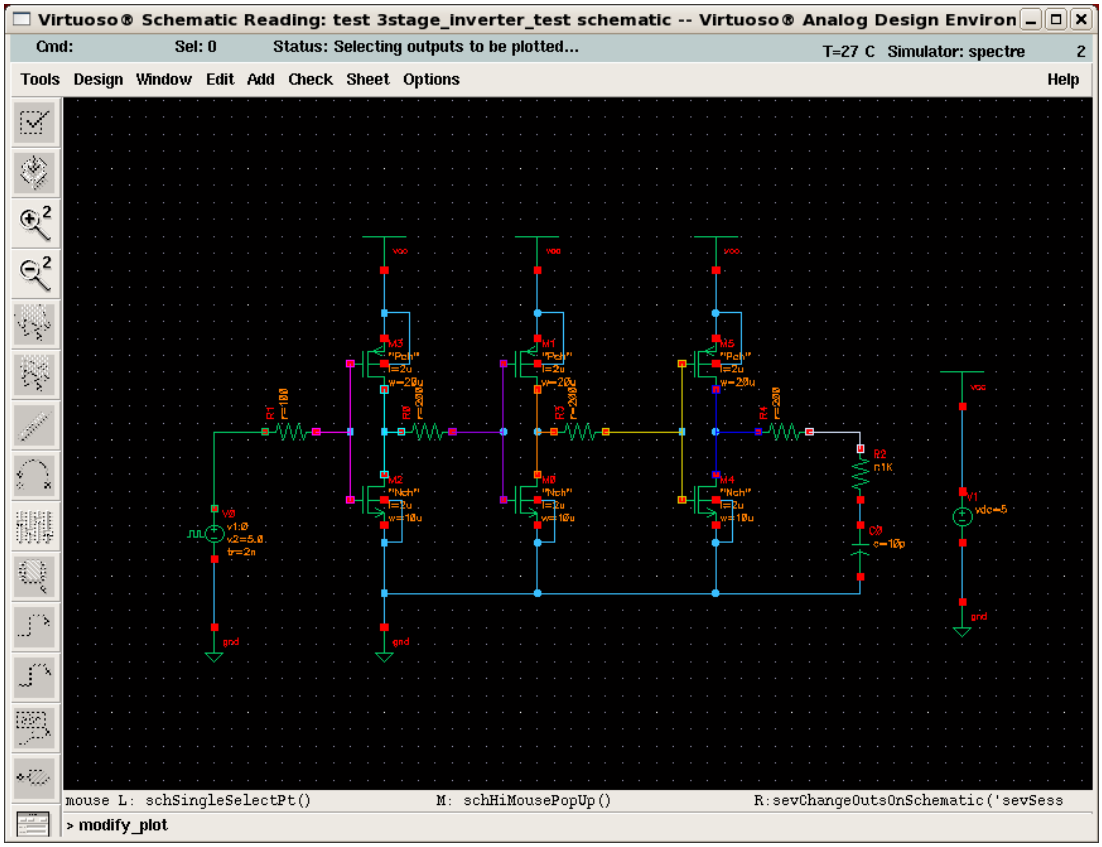


Figure 26-6 Select Nodes on Circuit Schematic to plot

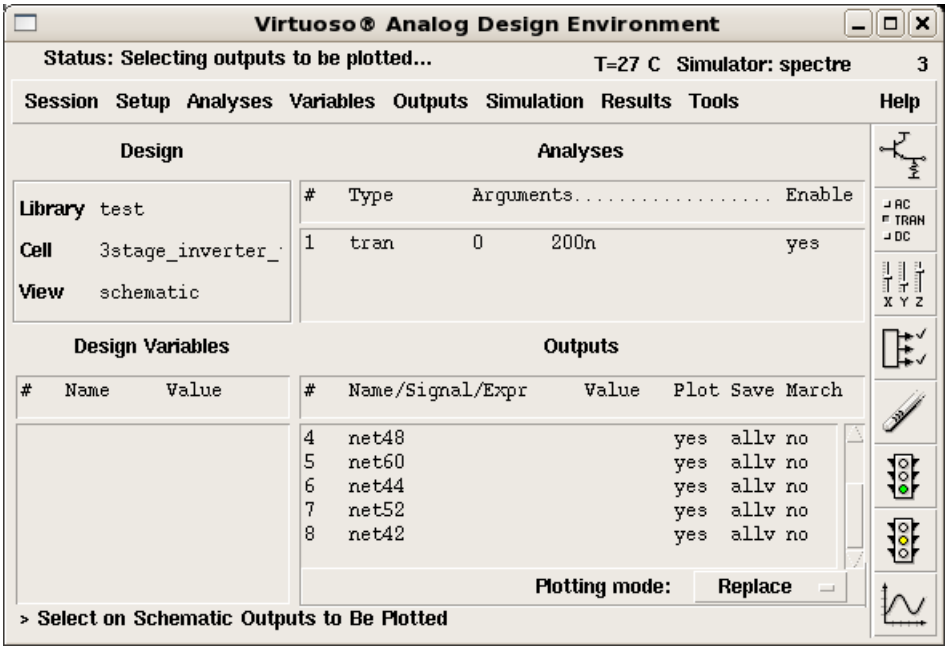


Figure 26-7 Main Simulation Control Window Showing Vectors to be Plotted Automatically

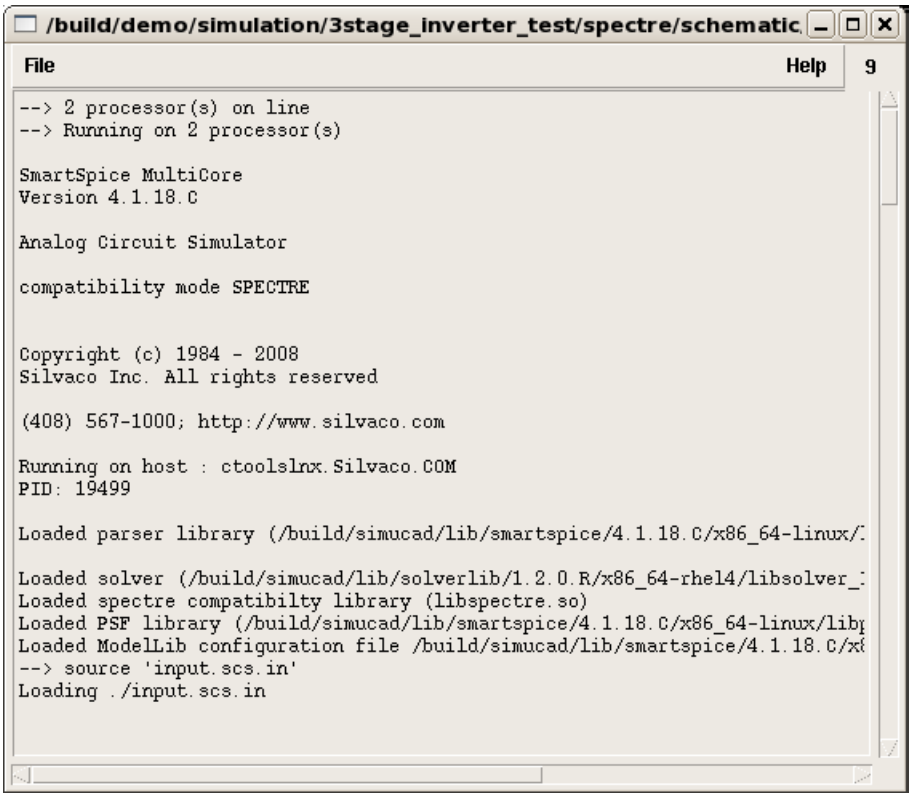


Figure 26-8 Example Deck Running in SmartSpice under Analog Artist

Figure 26-9 Automatically Plotted Waveform from SmartSpice

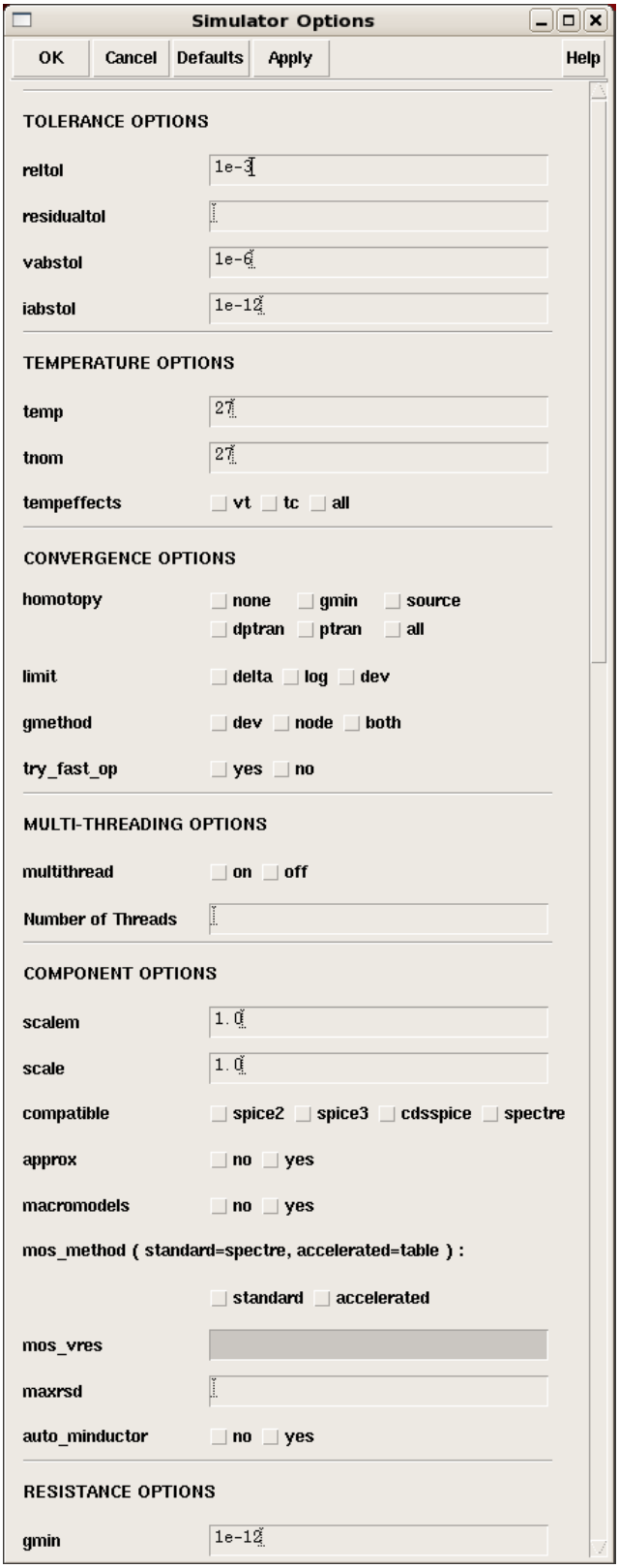


Figure 26-10 Simulation Options Setup

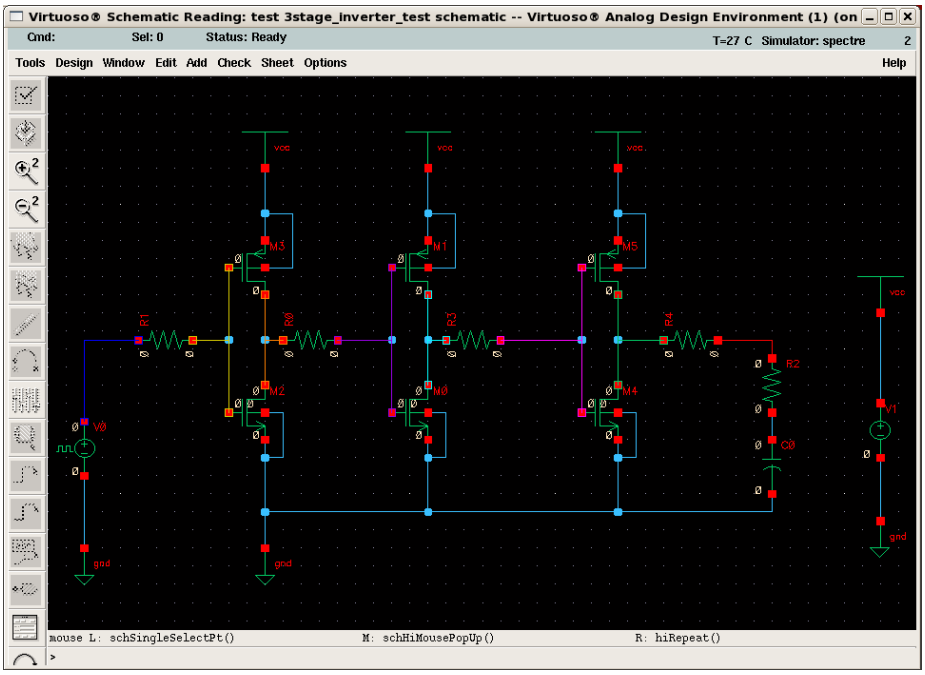
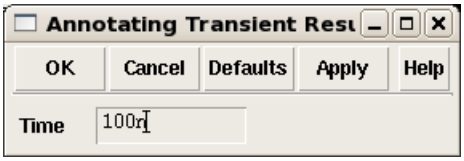


Figure 26-11 Node Voltages at transient time = 0 Backannotation



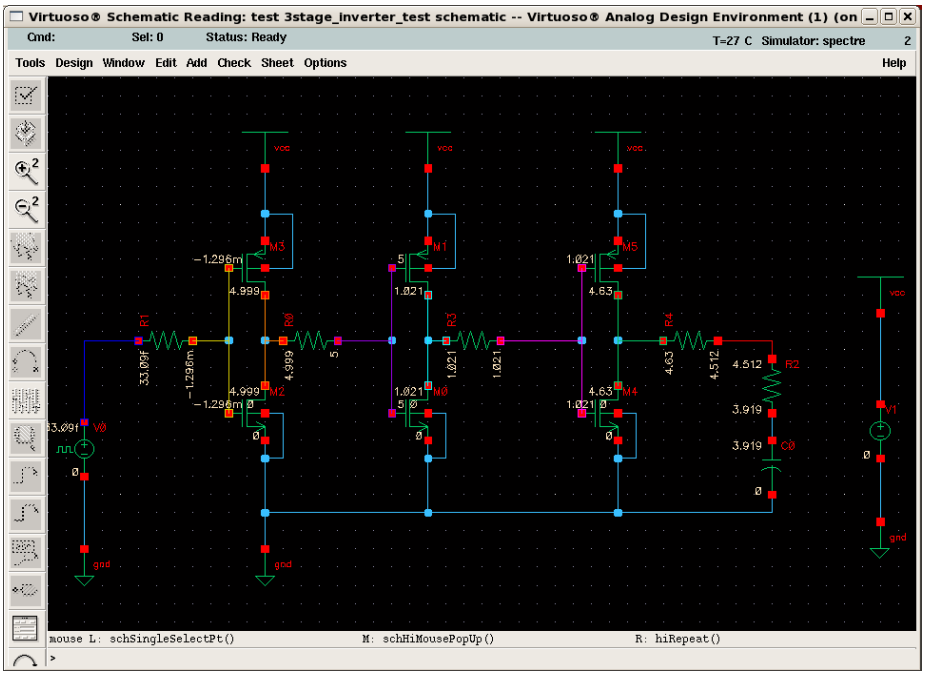


Figure 26-12 Node Voltages at transient time = 100ns Backannotation

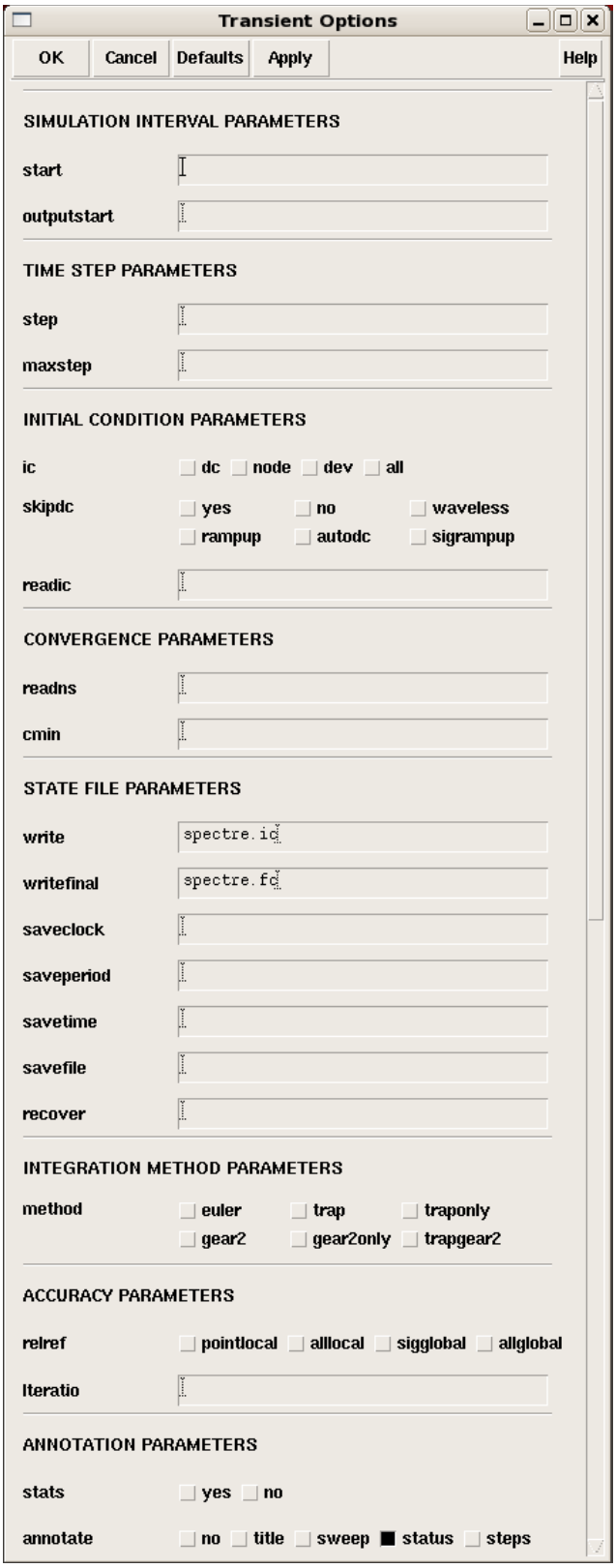


Figure 26-13 Transient Analysis Options Setup

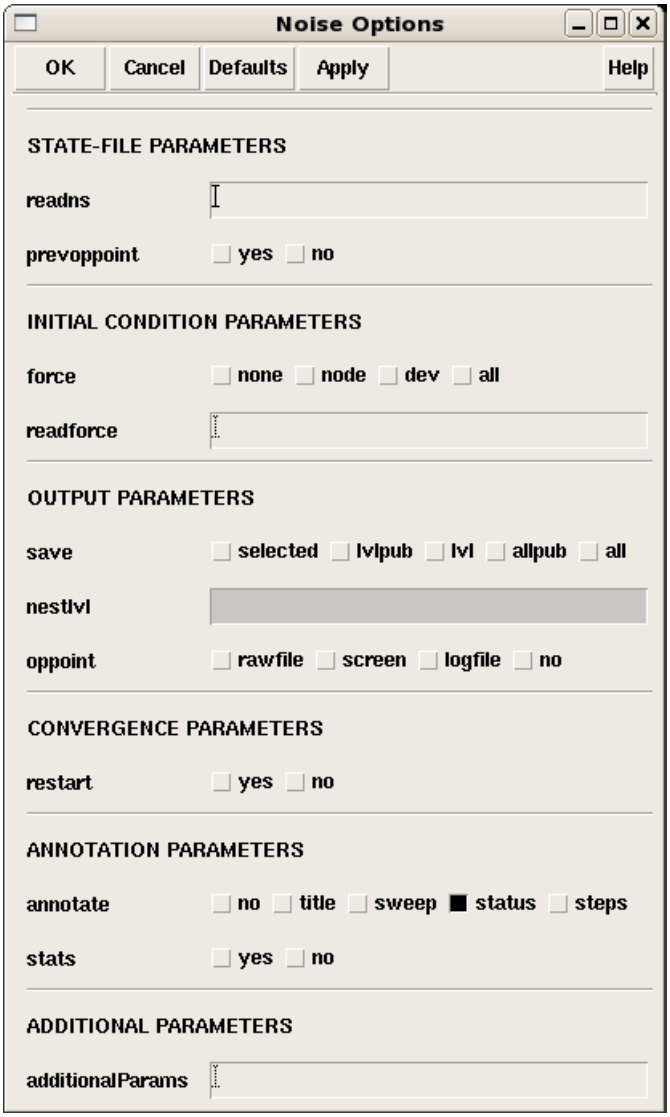


Figure 26-14 Noise Analysis Setup

Note: All resistors in SmartSpice have a noise contribution by default, so if in Spectre `ISNOISY=true` is present it is effectively ignored. If the setting for a resistor is `ISNOISY=false` then in SmartSpice the resistor is simulated as a noise-less device.

26.4.1 Simulation Netlist Debug

If a simulation does not start. and the `CDS.log` and `spectre.out` file does not provide enough information, it is useful to source the input netlist from the SmartSpice GUI in interactive Spectre compatible mode (`smartspice -spectre`). The input netlist can be found at:

```
/simulation/cellview/spectre/schematic/netlist/input.scs.in
```

`input.scs.in` is the SmartSpice version of the `input.scs` and has some extra SmartSpice syntax appended at the end of psf plotting. This file can be run in SmartSpice stand alone mode.

Sourcing the deck from the **File** menu will provide error messages in the **Output** pane or **Messages** window. The translation of the Spectre syntax to SmartSpice syntax can also be seen by selecting the appropriate deck in the **Decks** tab and then right-clicking to select **View Listing→Physical**. This will bring up the SmartSpice physical deck which contains the SmartSpice syntax converted from the Spectre syntax.

An environment variable `SPECTRE_V` is used to show the specific version of Spectre being used. In debug this version can specifaclly be set by the user.

Chapter 27
Speed/Performance Control Settings

27.1 Performance Control Using Command Line Mode '-turbo'

The command line option `-turbo` activates a fast simulation mode and sets a group of related options (suboptions). Accuracy-performance balance is controlled by advanced algorithms. Suboptions can be used separately and will override the default value under `-turbo`.

The command line option `-turbo` sets the following set of options:

```
bypass=2, runlvl=2, solver=SPEEDS, hsimspeed=3, pivrel=1e-17,
cflflag=1, rmax=25, expbypass=1e-03
```

See the following table for `-turbo` and the corresponding option values:

Option	-turbo mode
`.option bypass`	2
`.option runlvl`	2
`.option solver`	SPEEDS
`.option hsimspeed`	3
`.option pivrel`	1.00E-017
`.option cflflag`	1
`.option rmax`	25
`.option expbypass`	1e-03

The option bypass activates latency check in the transistor model. It improves performance for the model evaluation. Always check simulation results for accuracy. If results are not satisfied, reset the option to zero:

```
.option bypass=0
```

Currently the following models support option `bypass=2`:

Models, supporing option bypass=2
DIO
BJT
BSIM3v3
BSIM4
RPI a-Si TFT (RPIaTFT)
RPI p-Si TFT (RPIpTFT)

Setting `runlvl=2` activates the relaxation algorithm for waveform slope control.

Solver SPEEDS provides the best balanced multithreaded accuracy-performance algorithm for large circuits (500K nodes). Recommended for slow converged circuits (uses `CONV=1, 2, 4, -1`).

Option `hsimspeed=3` activates model RsRd collapsing scheme. If it is activated, RS and RD related internal nodes will be collapsed.

Currently the following models support option `hsimspeed=3`:

Models, supporing option hsimspeed=3
DIO
BSIM3v3
BSIM4
RPI a-Si TFT (RPIaTFT)
RPI p-Si TFT (RPIpTFT)

Option `pivrel=1e-17` sets the relative ratio between the largest column entry and an acceptable pivot value to 1e-17. Default is 1e-3.

Option `cflflag=1` activates the precompiled binaries for the instance runtime expressions.

During the simulation under option `-turbo` SmartSpice will print out the values of all dependent options.

Example

```
smartspice -turbo
```

Output:

```
Mode '-turbo' settings:
BYPASS: 2
HSIMSPEED: 3
RUNLVL: 2
PIVREL: 1e-17
CFLFLAG: 1
SOLVER: SPEEDS 64-bit
```

27.2 Transistor Terminal RS/RD Reduction Technique

New advanced algorithms under the option `hsimspeed` provides reduction of extra drain/source nodes created by RS/RD resistors. Two subalgorithm: `.option hsimspeed=5` and `hsimspeed=3` reduces circuit matrix and provides performance increase for solver. 5 might be used for analog/digital circuits, and 3 for analog circuits (charge pumps, vco, pll).

Example

```
.option hsimspeed=5|3
```

27.3 SmartSpice HPP Mode

In HPP (High Performance Parallel) mode the global circuit is decomposed into a number of subcircuits (blocks) which are calculated separately, then the final solution adjusted considering the interconnection between all subcircuits. Decomposition can be done based on the user defined hierarchy, or by graph partitioning applied to the hierarchical or flattened circuit. A new DDS solver (Domain Decomposition Solver) is recommended for running in HPP mode. Additionally the circuit decomposition DDS solver will cause each subcircuit to have its own small matrix instead of one global matrix. This leads to a smaller factoring time and overall speedup.

```
-hpp startup key
```

Initializes HPP mode. `-hpp` mode sets following default options:

- `hsimspeed=3`
- `bypass=2`
- `expbypass=1e-3`
- `cflflag=1`

```
-hppmultirate startup key
```

Uses different time steps for blocks and top level circuit variables. Turns on HPP mode. Overrides the specified `.option lvltim` to the `lvltim=4`. Use `.option reltol` to adjust accuracy.

Example

```
smartspice -hpp
smartspice -hppmultirate
```

27.3.1 User Defined Hierarchy Partitioning Options

Here is the list of options for user defined hierarchy partitioning algorithm:

`.option separator_level=number`	All cells starting from this level of hierarchy will form blocks for independent calculation. All elements starting from top level to separator level exclusively will form interconnect network. If not defined auto detection algorithm will calculate separator level to minimize interconnect network and maximize number of separate blocks.

.option hpp_block_size=number	Size of block sets number of internal nodes for each block. After defining separator level all cells there will be reordered and merged into composite blocks to reduce number of small blocks on matrix diagonal. This option specifies desired size of block. Default value = 100.

27.3.2 Graph Partitioning Options

```
-partition startup key
```

Initializes graph partitioning for HPP mode. In this mode circuit graph partitioned to form blocks. This key should be used with `-hpp` startup key. By default number of `partition=4`, but it can be redefined with `.option hpp_partition_count`. If some partitions too small they will be merged into one. Related options:

.option hpp_partition_count=number	Set desired partitions count for graph partitioning. Default value = 4.

Example

```
smartspice -hpp -partition
smartspice -hppmultirate -partition
```

27.3.3 Common Options

.option hpp_nestdiss_intercon=number	Specifies number of levels of nested dissection which will be used to decompose interconnection matrix to provide more parallelism in calculation. Can speedup calculation on multiple CPUs. Default value = 0.

27.3.4 Domain Decomposition Solver

As mentioned above DDS solver finds solution for set of small matrices instead of one global matrix. This gives higher parallelism and less factorization time. DDS solver can be engaged by:

```
.option solver=dds
startup key -forcesolver dds
```

27.3.5 Isomorphism

Under DDS solver and user defined hierarchy, the partitioning block isomorphism feature available. During simulation topologically, isomorphic blocks share their results. So instead of simulating all blocks only one of them simulated and others use calculated results. Always check simulation results for accuracy.

.option hpp_block_isomorphism	Turns on isomorphic blocks detection.
.option hpp_block_isomorphism_reltol	Isomorphism relative voltage error tolerance for blocks. Default value 1e-3.

.option hpp_block_isomorphism_abstol	Isomorphism absolute current error tolerance for blocks. Default value 1e-9.
.option hpp_block_isomorphism_vntol	Isomorphism absolute voltage error tolerance for blocks. Default value 50.e-6.

Chapter 28
Convergence Problems - Troubleshooting and Solving

28.1 Overview

Convergence is the objective of the iterative process of solving circuit equations within specified tolerances and a specified number of iterations. That is, if consecutive iterations achieve results within the specified accuracy tolerances, the circuit simulation has converged.

Starting with an initial guess, these iterative processes in OP and Transient analyses continue until all of the node voltages settle to values that are within specific tolerance limits. These limits can be altered using various `.OPTIONS` parameters.

If the node voltages do not converge within a certain number of iterations, the OP analysis issues error messages, such as "`No convergence for conv=0`", "`Matrix is Singular`", or "`Newton Method is Nonconverged`". The simulation will be terminated because both the OP and Transient analyses require an initial stable operating point in order to proceed.

During the Transient analysis, this iterative process is repeated for each individual time point. If the solution in a particular time point does not converge, the time step is reduced and the simulator tries to calculate the node voltages again. If and when such a reduction makes the timestep smaller than predefined minimum, the error message "`Time step too small`" appears and the analysis will be halted.

Among the reasons for convergence failures are: unrealistic circuit impedance, wrong initial voltage estimates (OP analysis), model discontinuities, unrealistic circuit or parasitic modeling (Transient analysis).

28.2 Convergence Algorithms

If the standard Newton-Raphson method fails to converge, or if convergence requires a large number of iterations, SmartSpice performs the auto convergence algorithm in four steps. Steps 1, 2, 3 and 4 are advanced stepping algorithms, based on the continuation method, when the solution of each step provides a good initial guess for the next.

0. Standard Newton-Raphson Iterations: The circuit is simulated using the default values of the convergency control parameters. The limit of iterations is equal to `ITL1` (`100`, by default). If convergence is not achieved, the simulator goes to step 1.
1. DCGMIN Stepping: SmartSpice multiplies the `DCGMIN` (`1.0e-12`, by default) value by $10^{DCGMSTEPS}$, where `DCGMSTEPS` is the number of continuation method steps (`10`, by default), and simulates the circuit. If it converges, then `DCGMIN` is divided by `10` and the circuit is simulated again. This iteration process is continued until `DCGMIN` reaches the original value. At this point, the simulator attempts to perform Newton-Raphson method again, and if convergence is not achieved, SmartSpice goes to the step 2.
2. DiagGMIN Stepping: The simulator multiplies the `GMIN` (`1.0e-12`, by default) value by $10^{GMINSTEPS}$, where `GMINSTEPS` is the number of continuation method steps (`10`, by default), it changes the diagonal elements of the Jacobi Matrix by the value of parameter `GMIN`, and simulates the circuit. If it converges, then `GMIN` is divided by `10` and the circuit is simulated again. This iteration process is continued until `GMIN` reaches the original `GMIN` value. If convergence is not achieved, SmartSpice goes to the step 3.
3. Combination of steps 1 and 2 using the previous results as new initial conditions.
4. Source Stepping: Sets all voltage/current sources to `0` and simulates the circuit. If it converges, then the value of each voltage/current source is increased by an `SRCvalue`/`SRCSTEPS` increment, where `SRCvalue` is the original voltage/current source value, and `SRCSTEPS` is the number of continuation method steps (`10`, by default), and the circuit is simulated again. This iterative process is continued until the voltage/current source values reach the original level. To select algorithm we have to define `.OPTIONS CONV=0...4`.Default is `5` - auto convergence algorithm. If the first convergence method fails, it will proceed to the next.

28.3 General Remedies

Many convergence problems can be solved with the introduction of the .OPTIONS parameters GMIN and GNODE.

GMIN is the conductance placed in parallel with all p-n junctions of all models, and all drain-to-source nodes of the MOSFET models, to improve Newton iterations convergence properties at each time point. The conductance is used to change the diagonal elements of the Jacobi Matrix by the value of the parameter GMIN. Its default value is 1e-12[mhos]. Setting GMIN to a value between 1n and 10n often solves convergence problems.

GNODE causes an insertion of a resistor from every node in the circuit to ground. GNODE value between 1.e-14 and 1.e-12 typically helps. Bigger GNODE value may cause accuracy problems.

The default numerical integration method is the TRAP(TRAPEZOIDAL) method. It has good accuracy but can cause computational oscillation. The time step becomes too small and simulation slows down. Some circuits converge better when the GEAR integration method is used. The GEAR method smooths out the oscillations found in the TRAP(TRAPEZOIDAL). To invoke the GEAR integration, use the .OPTIONS METHOD=GEAR statement. The GEAR method works well for most power electronics simulations.

Most of the time, however, the reason for convergence problems are minor mistakes in netlist syntax and the circuit construction. Therefore, it is a good idea to check each line of the netlist before the simulation.

The following is a collection of tips and hints that might be useful in solving convergence problems.

28.3.1 OP Convergence Aids

1. Check the input file carefully for:
   - correct connectivity
   - components polarity
   - floating and undefined nodes; every node has to have DC path to ground
   - loops of inductors or voltage sources
   - series capacitors or current sources
   - realistic parameters values - parameters of conductances, capacitances and inductances should not differ more than 14-15 orders of magnitude
   - syntax mistakes
2. Define .OPTIONS EXPERT=777. If the RELTOL value has a strong tendency to become <1, but needs some more iterations to do it, increase ITL1 and ITL2 up to 500 in the .OPTIONS statement. In all but the most complex circuits, increases over 500 won't typically aid convergence.
3. Relax ABSTOL and RELTOL parameters. ABSTOL should not be more than 9 orders of magnitude smaller than the largest current of nonlinear device.
4. Add .NODESET values for the top level circuit nodes. Use a .NODESET value of 0V if you do not have a better estimation of the proper DC voltage.
5. Add or delete certain physical effects of the device model. Do not use complex models without necessity, but keep them consistent enough for accuracy and stability of the solution. Make sure you know what you are doing.

6. Use the parameter tranop in the .TRAN statement, which invokes transient ramping procedure similar to source stepping. This method is more reliable than the operating point calculation. However, it is usually more time consuming.

DC Accuracy Considerations

- Good accuracy assumes that models are consistent, accurate and correct.
- The circuit is free from any topological defects: floating nodes, less than 2 connection nodes, each node has a dcpath to ground.
- The circuit may have multiple stable states. Use the .NODESET statement to force the simulator to find the desired solution.
- Tighten .OPTION RELTOL=val; set reasonable .OPTION ABSTOL=val and VNTOL=val.
- If .OPTION GMIN/DCGMIN=val affects the solution, set one to zero

28.3.2 Transient Convergence Aids

1. Check circuit topology and connectivity (as in the OP analysis).
2. Use realistically modeled devices; add parasitics, especially stray/junction capacitance, to smooth any strong nonlinearities or discontinuities. This may be accomplished by the addition of capacitance to various nodes and verifying that all semiconductor junctions have capacitance.
3. Use realistic rise/fall times of the PULSE sources to smooth strong nonlinearities. If no rise or fall time values are given, or if 0 is specified, the rise and fall times will be set to the TSTEP value in the .TRAN statement.
4. Increase number of iterations, ITL4 up to 40+.
5. Change the integration method to GEAR. This option causes GEAR integration to solve the transient equations, as opposed to the default method of TRAP(TRAPEZOIDAL) integration. Use the GEAR integration method with a reduced RELTOL value. This will produce more stable numerical solution. Trapezoidal integration tends to produce a less stable solution because of spurious oscillations. GEAR integration often produces superior results for power circuitry simulations, due to the fact that high frequency ringing and long simulation periods are often encountered.
6. Set TRTOL to a large value. The time step in this situation will be controlled by the iteration count.

The recommendations were collected from various sources, including:

1. Kielkowski, Ron. “Inside SPICE”, McGraw-Hill, Inc., 1994
2. Vladimirescu, Andrei. “The SPICE Book”, John Wiley & Sons Inc., 1994

Transient Accuracy Considerations

- Circuit topology is free from defects.
- Operating point for each device is correct.
- Tighten RELTOL.
- Set reasonable ABSTOL, VNTOL, CHGTOL.
- Decrease TRTOL.
- Check if GMIN is affecting the solution; set DCGMIN/GMIN to zero.
- Use Gear's second-order integration method if the solution is “ringing”.
- For low-loss resonator circuits, use trapezoidal integration method.
- To startup oscillator circuits, set the maximum timestep to at most one tenth the size of the expected period of oscillation.

- Set `.OPTION INTEGR=0` to suppress automatic switch of integrations methods.

28.3.3 Generating Initial Conditions using savebias

Specify in the input deck:

- `.savebias 'time.bias' tran time=100n all` - to save tranop info as future IC's
- `.tran 10p 200n tranop=100n` - to perform analysis with OP info at 100n
- run input deck
- open generated file `time.bias` in the editor and change `.NODESET` to `.IC`
- `.include 'time.bias'`
- remove `.savebias`
- change transient settings to `.tran 10p 700n UIC`
- run modified input deck

28.3.4 Checking Circuit Topology Defects

Input deck Example:

```
CHECKING TOPOLOGY

R1 3 0 1k
C1 3 99 1p
C2 99  2 1p
R3 2    22 1k
V1  44 0  dc 1
.op
.END
```

After sourcing the previous netlist, SmartSpice prints the following messages:

```
Circuit:     CHECKING TOPOLOGY

Warning: There are nodes with less than 2 connections.
The table of nodes with less than 2 connections is generated after
sourcing...
Warning: There are floating nodes.

The table of floating nodes is generated after sourcing...
***warning***: the following singular supplies were terminated to 1
meg resistor

supply       node1            node2
v1           44                  0

The following nodes have less than 2 connections:
-------------------------------------------------------------------
----------
|  22                                                          |  44
|
-------------------------------------------------------------------
----------

The floating nodes are terminated by DCPATH=1e-011:
-------------------------------------------------------------------
----------
|                                                               99
```

```
|
-----------------------------------------------------------------
----------
```

Nodes with less then 2 Connections

Nodes 22 and 44 have only 1 connection. Usually these defect types are not serious and will not cause non-convergence issue. SmartSpice warns about possible forgotten connections.

Floating Nodes

Floating nodes can not be created by the user. Such nodes appear during the calculation of OP/DC by disconnecting capacitors. Node 99 is not connected to any device during OP/DC and will cause a “Matrix singular” issue.

SmartSpice detects such nodes and connects them to ground using a `DCPATH` conductance. If influence of `DCPACTH` is significant, use `.OPTION DCPATH=val` to reduce the value. The `.OPTION DCGNODE=val` may also be used to fix floating node issues. The `.OPTION DCSTEP=val` can also be used to overcome such issues. SmartSpice will also detect floating gates of transistors.

Singular Voltage Supplies

Singular voltage supplies are those which are not connected to the rest of the circuit. They will cause matrix singularity during OP/DC and transient analysis calculations. SmartSpice provides a warning and automatically connects 1M ohm resistor to these nodes.

Bistable Nodes

The `.OPTION PRINTBISTABLENODES` will show nodes with oscillating voltages during the OP calculation. Such nodes are created by inverters and by feedback loops.

SmartSpice detects such nodes automatically for `CONV=5` and aborts the regular newton iterations and then restarts `CONV=5` with the predefined `.nodeset` voltage `((val1+val2)/2)` on such nodes.

Bistable nodes cause time overhead by repeating the same calculations for `ITL1` number of newton iterations.

Subcircuits Terminals with No Connection to the Upper Hierarchy Level

The `.OPTION CHECK_TERMINAL_CONNECT` forces SmartSpice to check the topology of the subcircuits and determine if there are subcircuit terminals with no connection to the upper levels of the subcircuit hierarchy. If such terminals are detected, an error message and table with these terminals is displayed. This is a fatal error and a simulation will not be performed.

Recursive Inclusion of Subcircuits

SmartSpice warns of endless recursivity for nested hierarchy

Example

```
X2 A
.subckt A
x1 B
.ends

.subckt B
x1 A
.ends
```

Shell Variable hspicetopologychecker

```
.control
set  hspicetopologychecker=true
.endc
```

This command is for the topology rule checker (TRC).

Default is false in the regular SmartSpice mode. In the Hspice mode of SmartSpice, this variable is true. TRC mode supports checking of MOSFET devices. If floating gates, shortened source-substrate and drain-substrate connections are detected, a message “Error: no dc path to ground from node x1.in” is printed when `hspicetopologychecker=true` or in Hspice mode, then SmartSpice will abort the simulation with the message “job aborted”. In regular SmartSpice mode, when `hspicetopologychecker=false` (default), SmartSpice will continue automatic shunting of floating nodes to ground using the conductance value `dcpath`.

Also in the TRC “Warning: no connection on node 0:0 defined in subckt 0 ignored” will be issued. SmartSpice will continue the simulation.

Chapter 29
Debugging Features

SmartSpice has an interactive GUI interface that can be used to investigate the functionality and topology, etc., of a circuit design to aid simulation.

Most users run a transient simulation which consists of 2 stages:

1. Operating point to establish a stable bias condition for all nodes of the circuit
2. Calculation of the frequency components as time is incremented

29.1 Read in Circuit Input Deck and Components

When the circuit input deck is sourced, all the paths and components on that path are brought together and checked for consistency.

The Parser will tell you in the log window if it cannot understand any syntax, or if some of the included components are not found.

You will need to make sure there are no errors, or if there are errors that all of them are understood, such as dummy circuits that have all terminals shorted together, or Bus elements that should be open circuit and will have a conductance added to the floating node to allow all node voltages to be calculated.

29.2 Transient Timepoint Stepping Analysis

The options `BREAKPOINTDISPLAY` and `ITERSDISPLAY` save information about break points and iterations at each transient time point.

Step 1

1. Add a line `.OPTIONS BREAKPOINTDISPLAY ITERSDISPLAY` in a deck to generate vectors `v(breaks)`, `v(iters)` and `v(4)`.
2. Run the simulation and open the **Vectors** dialog.
3. Select the vectors and click the **PLOT** button to show them in SmartView.

Step 2

1. In SmartView right click the mouse button on a chart and choose **Split All/Selected** to split waveforms `v(breaks)`, `v(iters)` and `v(4)`.

For waveforms `v(breaks)` and `v(iters)` turn ON **Line Markers** and **Bars** (**Chart→Line Markers** and **Chart→Bars**), and turn OFF **Lines** (**Chart→Lines**), see Figure 29-1.

Bars on `v(breaks)` waveform show transient breaks at a transient time. For example, the green label indicates that there is a break at transient time 2.1e-8 sec.

Bars on `v(iters)` waveforms shows how much iterations was proceed to achieve solution at a transient point. For example, the green label at `v(iters)` shows four iterations at transient time 1.96e-8 sec.

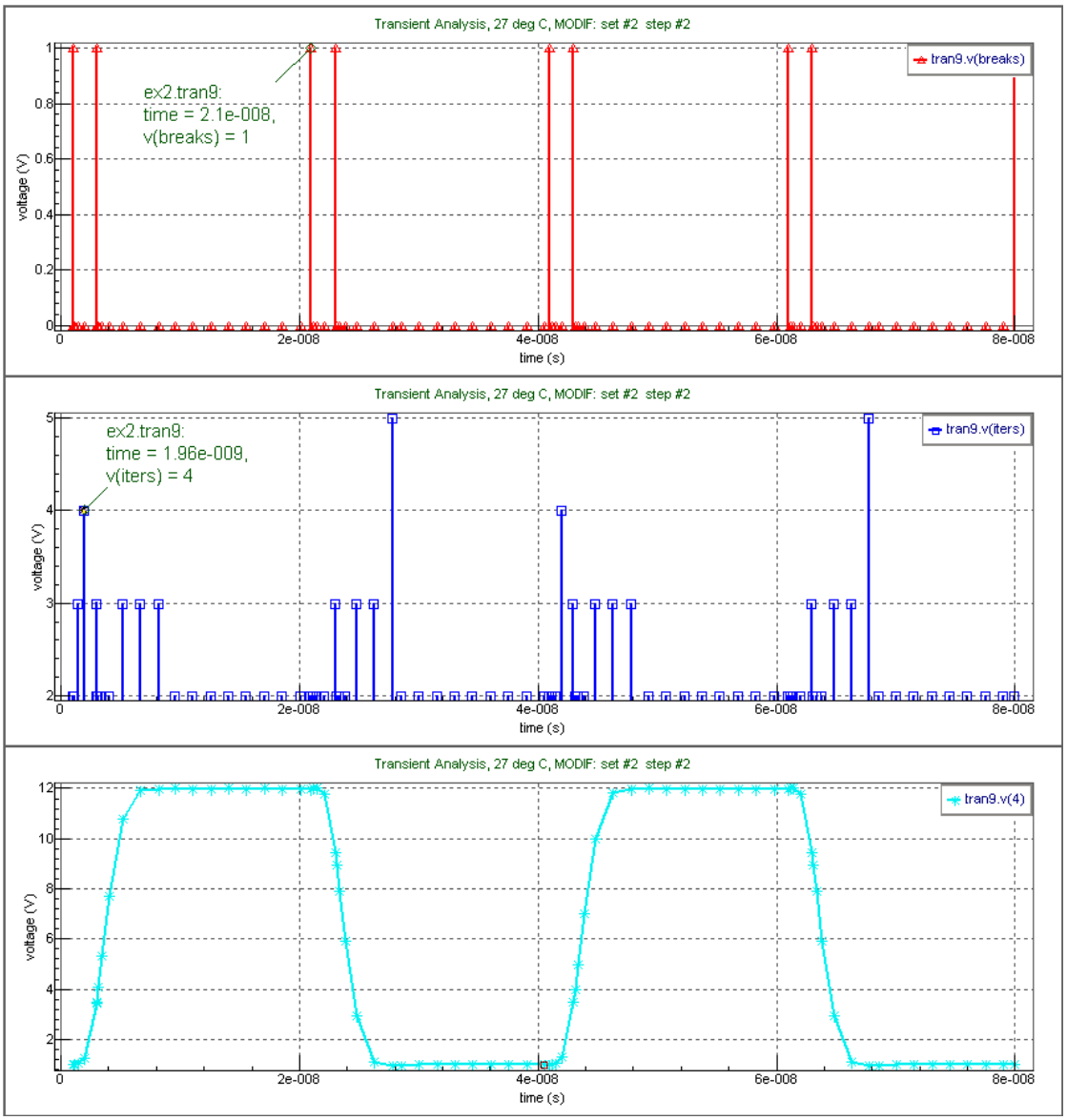


Figure 29-1 Waveforms v(breaks) and v(iters)

29.3 Calculate Simulation Waveform Difference

Syntax

```
calcdiff new_rawfilename reference_rawfilename <sigma_threshold
<sigma_interval <from <to>>>>
```

new_rawfilename	New file name
reference_rawfilename	Reference file name
sigma_threshold	Sigma threshold in %. Default is 0%.
sigma_interval	Sigma interval delta in %. Default is 10%.
from/to	[from, to] interval in %. Interval, where differences and sigmas will be calculated. Default is [0%, 100%].

The command CALCDIFF calculates:

- Differences for every vector in a raw file for the [from, to] interval and places them into new plots diff_error*.

  Differences are calculated by using the following formula:

  ```
  diff = (new_value - ref_value) /
  (ref_value_max - ref_value_min)
  ```

- Sigmas for the specified simulation interval (standard deviations) for every vector in a raw file and places them into new plots sigma_error*. Interval delta is assigned by the sigma_interval parameter. Default is 10%.

  Sigmas are calculated by using the following formula:

  ```
  sigma = sqrt ( sum ( diff * diff ) / num_points )
  ```

- Sigmas for all simulation for every vector in raw file. They will be printed in output file. Sigmas, which are lower than sigma_threshold, will not be printed.

Examples

```
calcdiff new_file.raw ref_file.raw
calcdiff new_file.raw ref_file.raw 0.1
calcdiff new_file.raw ref_file.raw 0.1 5
calcdiff new_file.raw ref_file.raw 0.1 5 25 75
```

In the first example, SmartSpice calculates differences and sigmas by using default parameter values (sigma_threshold=0, sigma_interval=10, from=0, to=100) .

The second example uses sigma_threshold=0.1 parameter value.

The third example uses sigma_threshold=0.1 and sigma_interval=5 parameter values.

The fourth example uses sigma_threshold=0.1, sigma_interval=5, from =25 and to =75 parameter values.

Step 1

1. Load SmartSpice in GUI mode.
2. Type the following command in the **Command History** tab:

```
calcdiff new_rawfilename reference_rawfilename
```

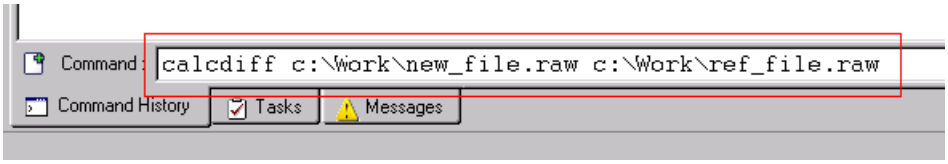


Figure 29-2 SmartSpice Main Window with Command History tab

Step 2

1. Select the `diff_error` and `sigma_error` plots, select the vectors, and click the **Plot** button to show them in SmartView.

In SmartView load your reference and new raw files (**File→Open**). Also, plot your reference and new waveforms (select plots and vectors and click the **Plot** button). Waveforms `diff_error*` will show you the differences between reference and new waveforms. Waveforms `sigma_error*` will show you sigmas for 10% simulation interval.

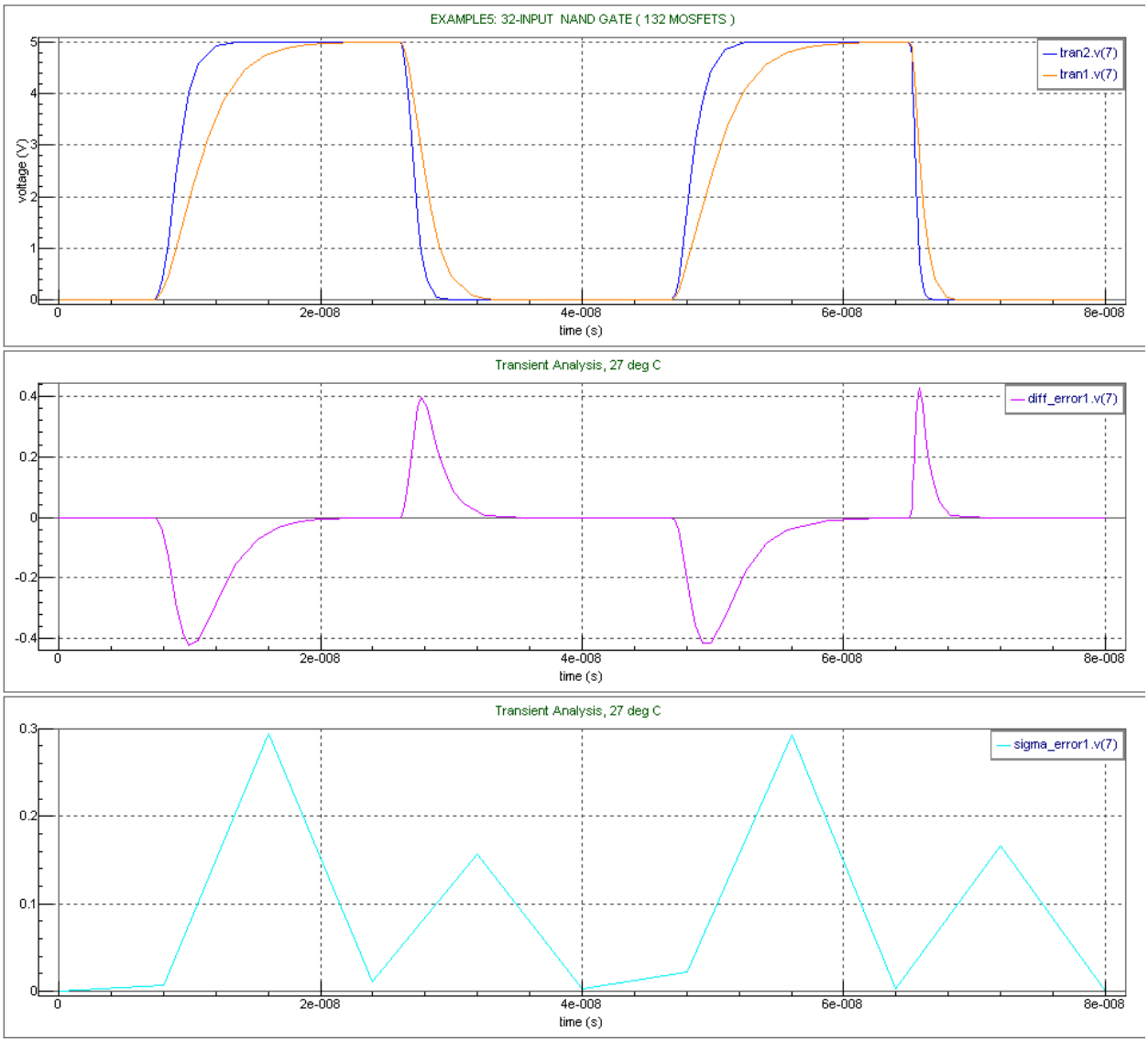


Figure 29-3 tran1.v(7), tran2.v(7), sigma_error1.v(7) and diff_error1.v(7) Waveforms in SmartView

Step 3

1. Look in the SmartSpice **Output** window for display of sigmas for all simulations and every vector, average sigma and the number of vectors, for which sigma was calculated. For example:

```
Transient Analysis, 27 deg C
v(1) sigma = 8.3257%
v(2) sigma = 3.1472%
```

```
v(3) sigma = 5.7912%

average sigma = 5.7547%
number of vectors = 3
```

29.4 SHOWTIMESTEP

The command SHOWTIMESTEP allows you to see how a simulation time step changed during simulation.

Step 1

1. Load SmartSpice in GUI mode.
2. Type the following command in the SmartSpice **Command History** tab:

   ```
   showtimestep rawfilename
   ```

Figure 29-4 SmartSpice Main Window with Command History tab

Step 2

1. Select the `timestep` and `numtimestep` vectors from the first plot and click the **Plot** button to show them in SmartView.
2. Select the `timestep_sorted` vector from the second plot and click the **Plot** button to show them in SmartView.

Vector `timestep` shows you how the time step changed during simulation; `numtimestep` shows you the quantity of the definite time step; `timestep_sorted` is the time step, which is sorted from the largest to the smallest value.

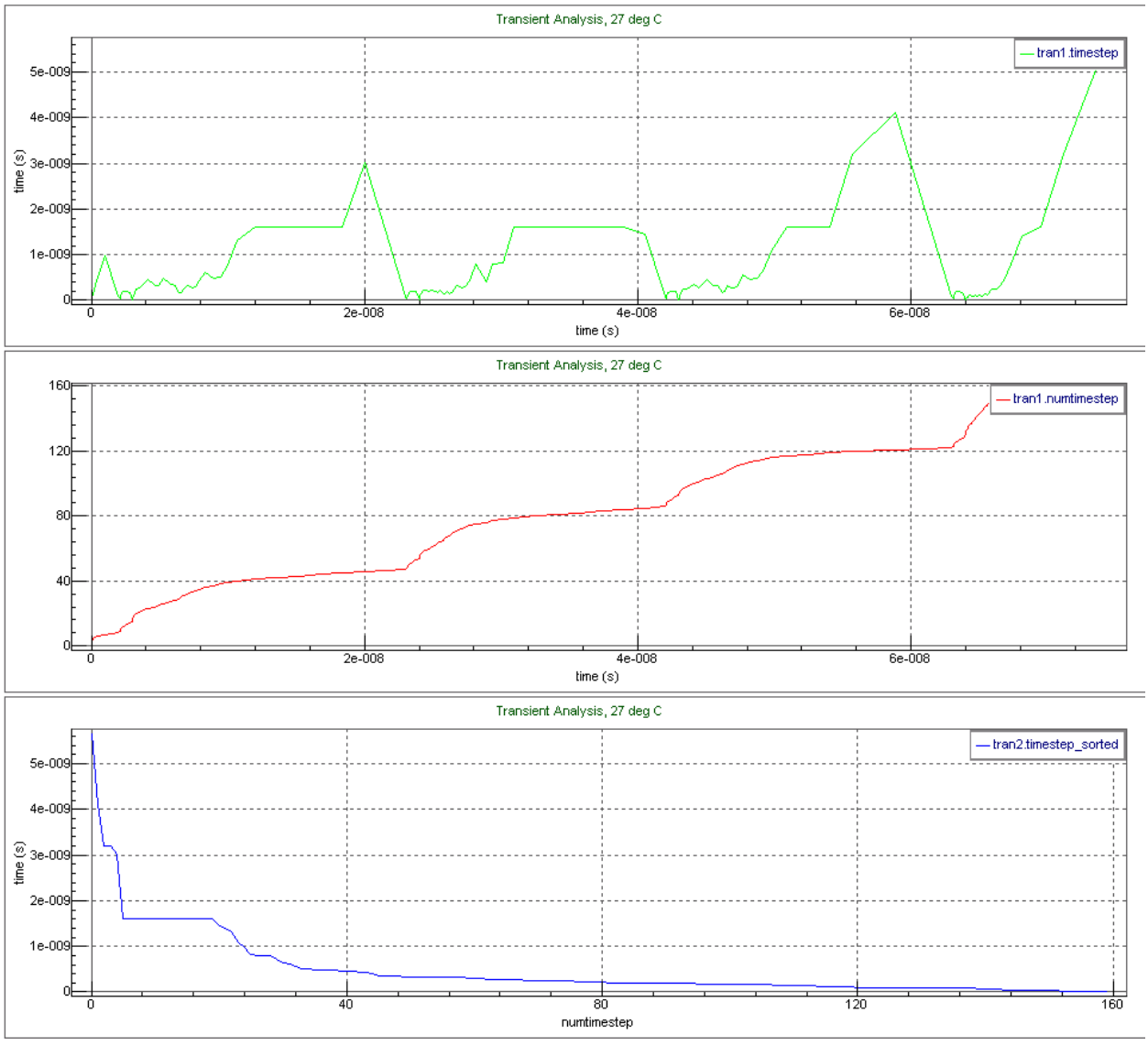


Figure 29-5 timestep, numtimestep and timestep_sorted Waveforms in SmartView

Step 3

1. Look in the **Output** window. SmartSpice will print the quantities and the time step values in it. For example:

```
Transient Analysis, 27 deg C
Quantity:       Timestep:
15              1.6e-009
4               2e-010
4               1e-010
3               4e-011
3               8e-011
3               2e-011
2               4.4444444e-011
2               8.8888889e-011
2               1.7777778e-010
2               1.6e-010
2               1.6e-011
2               2.0000103e-010
2               1.3218916e-010
2               8.0132004e-010
2               2.5384827e-010
2               2.6530156e-010
2               3.2e-009
2               2.0000691e-010
2               8.877557e-011
```

Step 4

To reduce the number of lines in the **Output** window, use the epsilon optional parameter. SmartSpice will group time steps whose value is less than epsilon. Default value is 1e-15.

1. Type the following command in the **Command History** dialog:

```
showtimestep rawfilename 1e-9
```

You will see the following output in the **Output** window:

```
Transient Analysis, 27 deg C
Quantity:       Timestep:
136             1.6e-011
19              1.3106324e-009
3               3.0178408e-009
```

Step 5

To see how many simulation steps were performed at a specific time, use the `end_time` optional parameter.

1. Type the following command in the **Command History** dialog:

```
showtimestep rawfilename 0 4e-8
```

You will see the following output in the **Output** window:

```
Transient Analysis, 27 deg C
End time: 4.000000e-008 Number steps: 85
```

29.5 Acquiring Data on MOS Operating Regions

To acquire data on MOS device operating regions during TRAN analysis, use the following statement:

```
.cutofftab mosdebug
```

If this statement is given, then for each MOS device a vector will be created in the plot named `mosregion_instancename_modelname`. Such a vector can contain only 3 distinct values at a timepoint:

`0`	means device was in Cut-off mode at timepoint
`1`	Linear mode
`2`	Saturation mode

29.6 Real Time vs Simulation Time Analysis

The option `REALTIMEDISPLAY` saves information about how fast or how slow a simulation performed during a transient analysis. On each time step of transient simulation the vector `v(realtimedisplay)` shows how much time passed since simulation has started.

A simulation 10u seconds of a transient analysis passed faster from 10u to 20u (see Figure 29-6) than from 20u to 30u (see Figure 29-7).

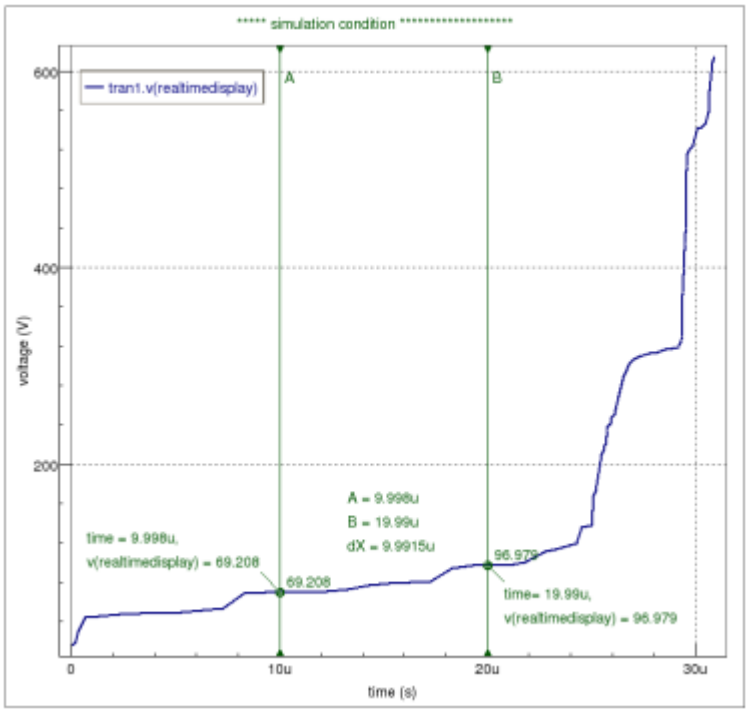


Figure 29-6 10u simulation of a transient passed for B(96.979s) - A(69.208s) = 27.771s

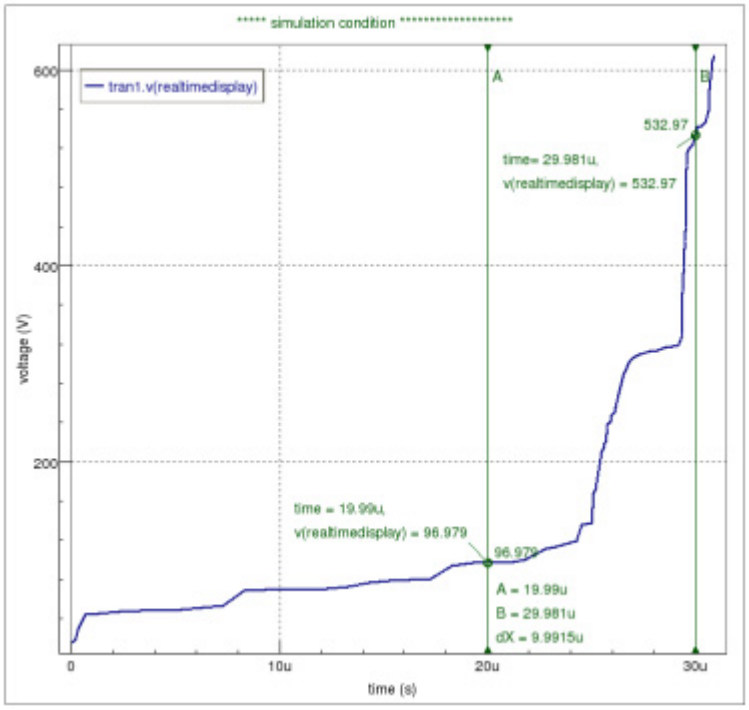


Figure 29-7 10u simulation of a transient passed for B(532.97s) - A(96.979s) = 435.991s

29.7 PRINTPAR

The command PRINTPAR allows you to print instance and model parameters.

Syntax

```
printpar
+ <rlclinear <resmin=value> <indmin=value> <capmin=value>
+ | rlcnonlinear> <device_name | model_mame>
```

Step 1

1. Load SmartSpice in GUI mode.
2. Source input deck by pressing the **Source Deck** button.
3. Type the following command in the **Command History** tab:

   ```
   printpar
   ```

Take a look in the **Output** window. SmartSpice will print in them all instance and model parameters.

Step 2

To print model and instance parameters of the particular device, use the following command:

```
printpar device_name
```

Step 3

SmartSpice allows the use of wildcards for the device name in the PRINTPAR command. For example:

```
printpar m.x1.m*
```

For this example, SmartSpice will print model and instance parameters for all MOSFETs in x-call X1.

Step 4

To print only model parameters of the particular model, use the following command:

```
printpar model_name
```

29.8 BINS

The command BINS prints a list of models and associated devices.

Syntax

```
bins <argument>
```

argument	If the argument equals 1, SmartSpice also prints the cell names, which contains models. If the argument equals 2, only model information will be printed. If the argument equals 3, SmartSpice also prints the total number of the binning models, and all binning model names, which is satisfies the binning criteria.

For MOS type models this command also prints values of binned parameters LMIN, LMAX, WMIN, WMAX, TMIN and TMAX.

For example, advanced1.in the command BINS outputs following information:

```
--> bins

model  v
                            0  v.x1.vb
                            1  vin
                            2  vcc
model  __cdefault
                            0  c.x1.cload
model  pmod  [level=3]
                            0  m.x1.m2
                            1  mp
model  nmod  [level=3]
                            0  m.x1.m1
                            1  mn
Loaded models:

nmod         in use      LMIN=0.000000e+000      LMAX=1.000000e+000
WMIN=0.000000e+000    WMAX=1.000000e+000    TMIN=-2.731500e+002
TMAX=-2.731500e+002
pmod         in use      LMIN=0.000000e+000      LMAX=1.000000e+000
WMIN=0.000000e+000    WMAX=1.000000e+000    TMIN=-2.731500e+002
TMAX=-2.731500e+002
```

29.9 Model Statistics Display

The command CMCSTAT allows you to display CMA (Continuous Model Approach) statistics. It shows instance groups and instance parameters corresponding to a specific model. Result model parameters for each group are also displayed.

Syntax

```
cmcstat
```

An example of command output:

```
****internal information about CMC format model use
****
Total Overhead During Simulation =   0.000000

Model nmos.1 (model not in use)
wmin=1u*par+l-w         wmax=2u*par+l-w
group        0               W=3.000000e-006        L=3.000000e-006
1.000000e-006                  2.000000e-006

Model nmos.2 (model in use)
wmin=2u*par+l-w         wmax=4u*par+l-w
group        0               W=3.000000e-006        L=3.000000e-006
2.000000e-006                  4.000000e-006
group        1               W=3.000000e-006        L=3.000000e-006
2.000000e-006                  4.000000e-006
group        2               W=3.000000e-006        L=3.000000e-006
2.000000e-006                  4.000000e-006
group        3               W=3.000000e-006        L=3.000000e-006
2.000000e-006                  4.000000e-006

Model pmos.1 (model not in use)
wmin=1u*par+l-w         wmax=2u*par+l-w
group        0               W=3.000000e-006        L=3.000000e-006
1.000000e-006                  2.000000e-006

Model pmos.2 (model in use)
wmin=2u*par+l-w         wmax=4u*par+l-w
group        0               W=3.000000e-006        L=3.000000e-006
2.000000e-006                  4.000000e-006
group        1               W=3.000000e-006        L=3.000000e-006
2.000000e-006                  4.000000e-006
group        2               W=3.000000e-006        L=3.000000e-006
2.000000e-006                  4.000000e-006
group        3               W=3.000000e-006        L=3.000000e-006
2.000000e-006                  4.000000e-006
```

29.10 CALIBRATEDEVICES

The command CALIBRATEDEVICES allows you to take I-V characteristics for all circuit devices for a sourced deck.

Syntax

```
calibratedevices device_type <checkcrossiv>
```

device_type	Must be one of the following types: • d - diodes; • m - MOSFETs.
checkcrossiv	Runs simulation for generated decks and finds crossings in I-V characteristics.

The command CALIBRATEDEVICES will generate the model cards, the input decks, the options and script file to take I-V characteristics for the specified device type. The files will be stored in the Calibrate/device_type folder next to the input deck file.

To generate decks the command goes through all models, extracts model parameters for each model, and saves them into <model_name>.model file. Then the command goes through all device instances for all processed models, extracts device parameters for each device, and saves them into a <device_name>.sp file. Generated options.spi has all options used in the analyzed source deck.

Parameter checkcrossiv runs simulation for all generated files and finds crossings in I-V characteristics analyzing waveforms in raw-files. If the parameter checkcrossiv is omitted then you should run script.sh manually to get simulation results for all generated files and using command checkcrossiv <raw_file>.raw analyze all raw-files.

In Figure 29-8 is a typical I-V characteristics generated by the command.

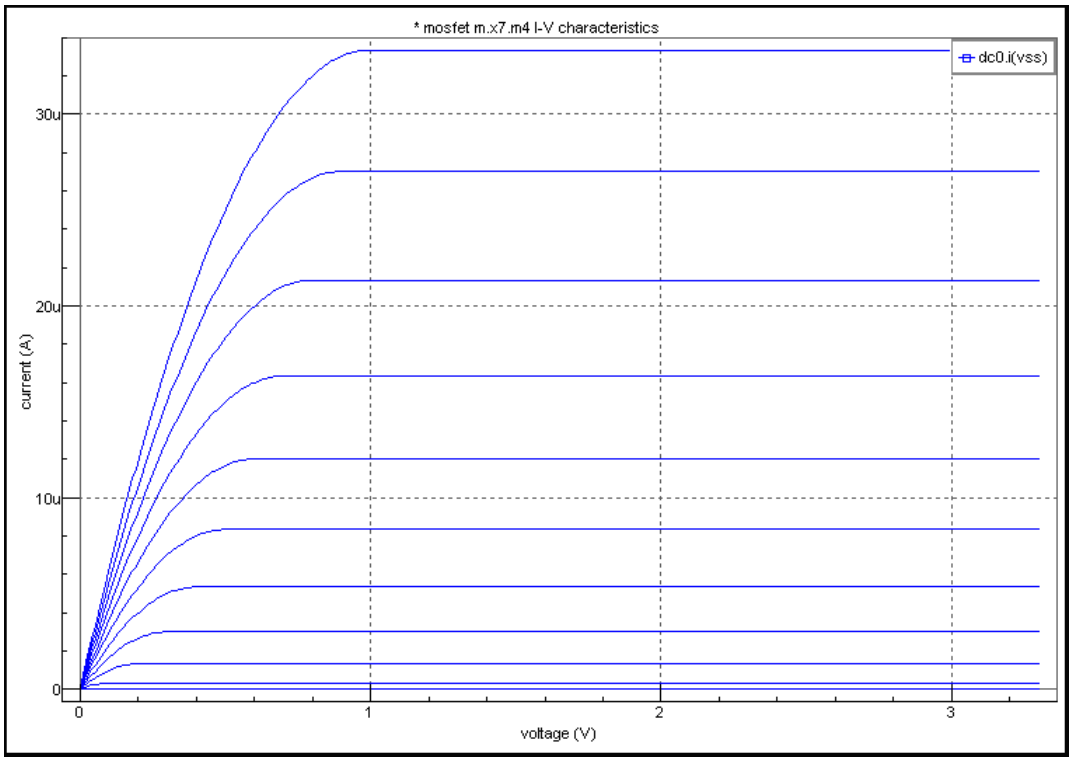


Figure 29-8 I-V Characteristics

For example, to take I-V characteristics for all MOSFET devices in deck `ex5.in` use the following steps:

- source `ex5.in`
- run command `calibratedevices m checkcrossiv` in the command window

The command `calibratedevices` will generate the following files:

MOSFETs model cards:

```
Calibrate/m/modn.model, Calibrate/m/modp.model
```

Generated input decks to take I-V characteristics and simulation results:

```
Calibrate/m/m.x7.m2.sp, Calibrate/m/m.x7.m4.sp,
Calibrate/m/m.x7.m2.out, Calibrate/m/m.x7.m4.out,
Calibrate/m/m.x7.m2.raw, Calibrate/m/m.x7.m4.raw
```

Common file with option for all generated input decks:

```
Calibrate/m/options.spi
```

Script file:

```
Calibrate/m/script.sh  - on Linux platforms
Calibrate/m/script.bat - on Windows platforms.
```

The contents of the modn.model file:

```
* model modn
.MODEL modn nmos
+ level = 1
+ vto = 1
+ uo = 550
```

```
+ cbd = 4e-014
+ pb = 0.8
+ beta = 2e-005
+ phi = 0.6
+ gamma = 0
```

The contents of the `m.x7.m4.sp` file:

```
mosfet m.x7.m4 I-V characteristics
.INCLUDE 'modn.model'
.INCLUDE 'options.spi'
VSS S 0 DC 0.0
VGS G 0 DC 0.0 SIN( 0 2.5 5G 0 0 0 )
VDS D 0 DC 0.0 1.0
VBS B 0 DC 0.0
M1 D G S B modn
+ w   = 1e-05
+ l   = 3e-06
+ as  = 1e-10
+ ad  = 1e-10
.TEMP 27
.DC VDS 1e-3 3.3 0.01 VGS 0 2 0.1
.TRAN 0.001n 10n 0n
.PROBE V(D) V(G) V(B) V(S) I(VDS) I(VGS) I(VBS) I(VSS)
.END
```

The contents of the `options.spi` file:

```
* options file
.option brief nomod
.OPTIONS nomod nodeck post probe acct=0 brief=0 conv=0
.OPTIONS mfileformat=2
.control
set parametric_data_in_raw=true
run batchprint
.endc
```

The contents of the `script.sh` file:

```
#! /bin/csh
smartspice -P 1 -PS 1 -sb m.x7.m4.sp -o m.x7.m4.out -r m.x7.m4.raw
smartspice -P 1 -PS 1 -sb m.x7.m2.sp -o m.x7.m2.out -r m.x7.m2.raw
```

29.11 Run-time Statistics Display and Menu Options

The drop-down menu on a **Run Time** screen contains the following menu entries:

- **Display simulation options:** `lvltim`, `reltol`, `abstol`, `volttol`, `trtol`, `accurate`, `fft_accurate`, `method`.
- **Display run-time statistics:** Timings for Total Analysis, Load, Setup, Temperature Setup, Truncation Control, Matrix Reordering, L-U Decomposition, Matrix Solve, Convergence control, Time step control, Simulation Data I/O, Measurements, Postprocessor and Other Operations phases during simulation.
- **Display transient statistics:** `totiter`, `totstep`, `CONrej`, `DEVrej`, `MATrej`, `LTErej`, `DERrej`.
- **Display simulation estimates:** TRAN speed, TRAN elapsed time, TRAN compile time.
- **Display silver monitor:** Matrix size, number of non-zeroes, Number of new elements, etc.
- **Display runmode settings:** Name, BYPASS, HSIMSPEED, MODELFAST, RUNLVL.

In addition, the format of units to display can be toggled between scientific and engineering:

- Engineering format

In GUI mode, the current state of simulation, displayed on a **Run Time** screen can be dumped to separate window and saved to file. Just click a menu item (or toolbar button):

- Take a Snapshot

Chapter 30
Stimulus Editor

30.1 Introduction

The stimulus editor tool in SmartSpice GUI mode allows the user to preview the the input stimulus of a circuit and change them without the need to run the complete simulation. This saves simulation time on an incorrect circuit evaluation.

The Stimulus editor tool allows you to preview the values of vector-type parameters of devices (such as PWL, PULSE, SINE) currently supported (Additional functionality will be added later). Also, it is possible to make modifications to the parameters curve values or build up a new one by applying changes into PWL.

To launch the Stimulus Editor through the main menu, first source the input deck containing V-sources, then go to **View→Circuit→Devices**. The **Devices** view will be opened.

Note: Currently, due to design limitations, on UNIX a single simulation run must performed in order to make device parameters be displayed.

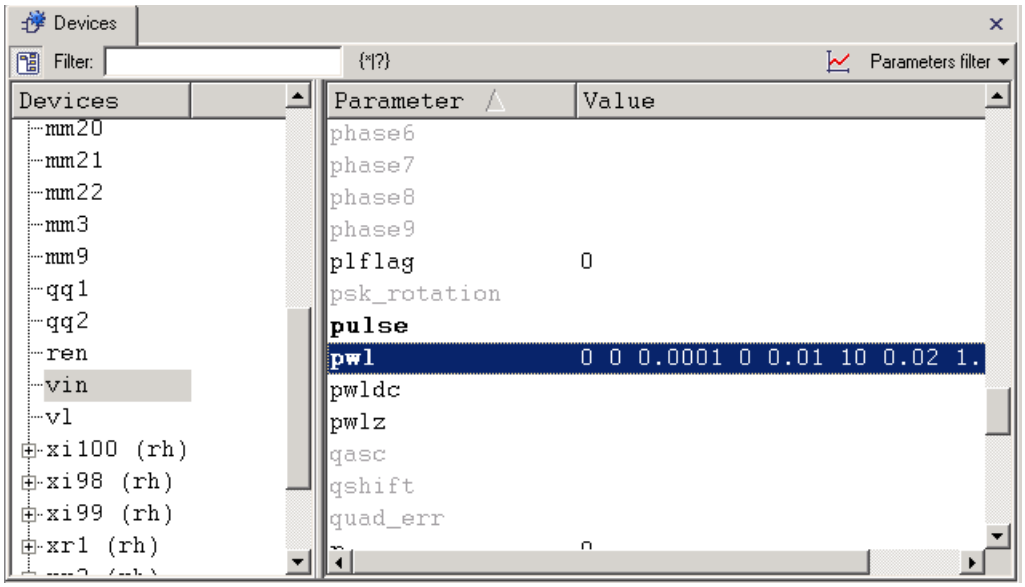


Figure 30-1 Stimulus Editor - Devices View

The left pane contains the list of devices for current circuit, from which you can select a V-source. The right pane contains the list of parameters for selected device. The complex parameters of vector-type which can be handled with Stimulus Editor are highlighted in bold. As a convenience, you can apply a parameters filter by choosing the type from the drop-down menu at the top-right corner of the **Devices** view. Currently there is only one option - **Vectors only**. If this option is activated, the right pane will display only parameters of that type:

Figure 30-2 Parameters filter - Devices View

To invoke the Stimulus Editor on PWL parameter, select it from the right pane and then press **Enter** or just double-click it. There is also a button for this purpose next to **Parameters filter**:

Figure 30-3 Stimulus Editor Invoke Button

Other parameters, such as PULSE or SINE can be previewed in the Stimulus Editor via an additional property sheet from which you can customize every single sub-parameter individually, see covered below.

Note: It is possible to edit and back-annotate the values of parameters in **Devices** view. The edit mode is activated by pressing **Enter** or double clicking on parameter which is subject to modification and during the 'IDLE' phase of simulator. But for above mentioned, complex parameters (highlighted in bold) these actions will trigger auxiliary means of editing - such as property sheets or the Stimulus Editor directly. In order to retain the default behavior hit select **Ctrl** key modifier alongside with **Enter** or double-click.

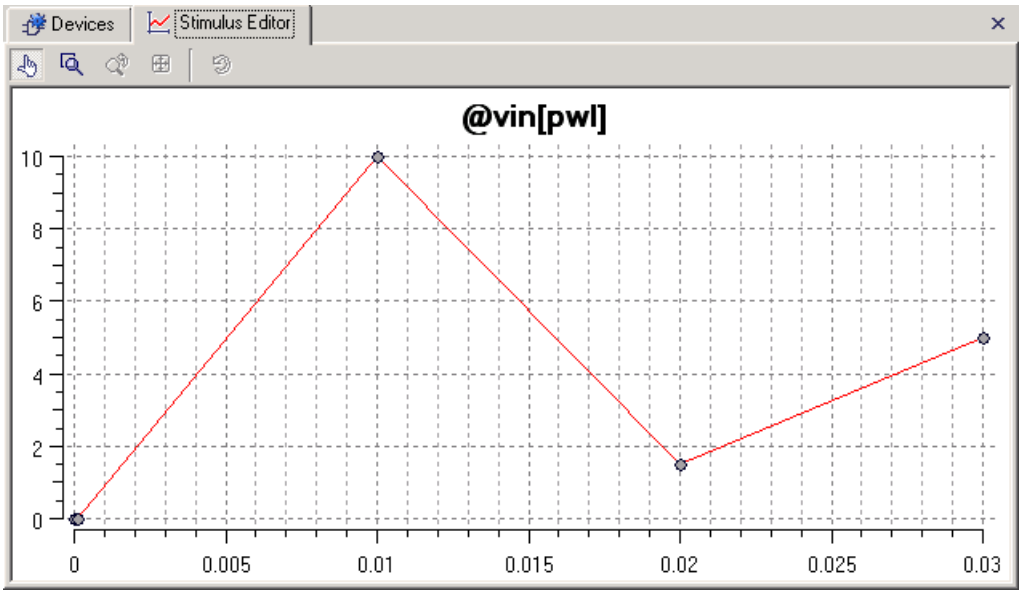


Figure 30-4 Stimulus Editor - Edit mode

30.2 Operating Stimulus Editor Modes

There are 2 modes to operate Stimulus Editor:

- Edit mode
- Zoom mode

Edit Mode

In Edit mode you can create/modify the stimulus for selected parameter. This is activated by default. Insert new nodes by left-clicking, or click over existing one and drag its position to modify. Right-click on a node to delete it. In addition to manipulating the node with the mouse, a numeric keypad can also be used. There is a one node that is to be currently selected (it's highlighted). You can move it by typing '7','8','9','4','6','1','2','3' keys with **Num Lock** ON or use the **Delete** key to delete it.

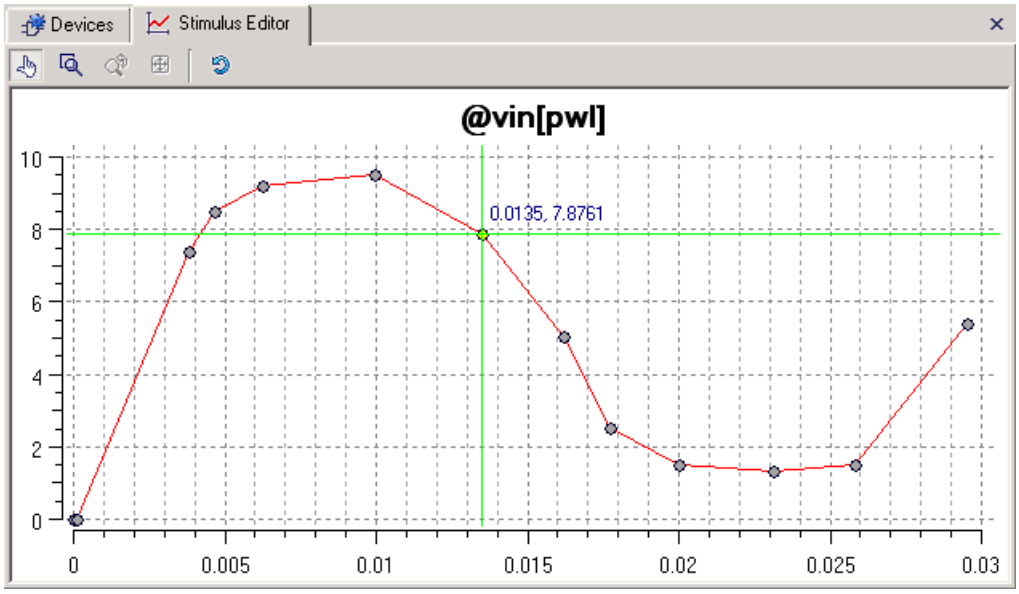


Figure 30-5 Stimulus Editor - Edit Mode

Zoom Mode

In Zoom mode you can rubberband selections on the plot allowing zooming in and out. Just left-click a position on a plot and drag it until the required area is covered. When the mouse button is released the plot will be zoomed into the selected rectangle.

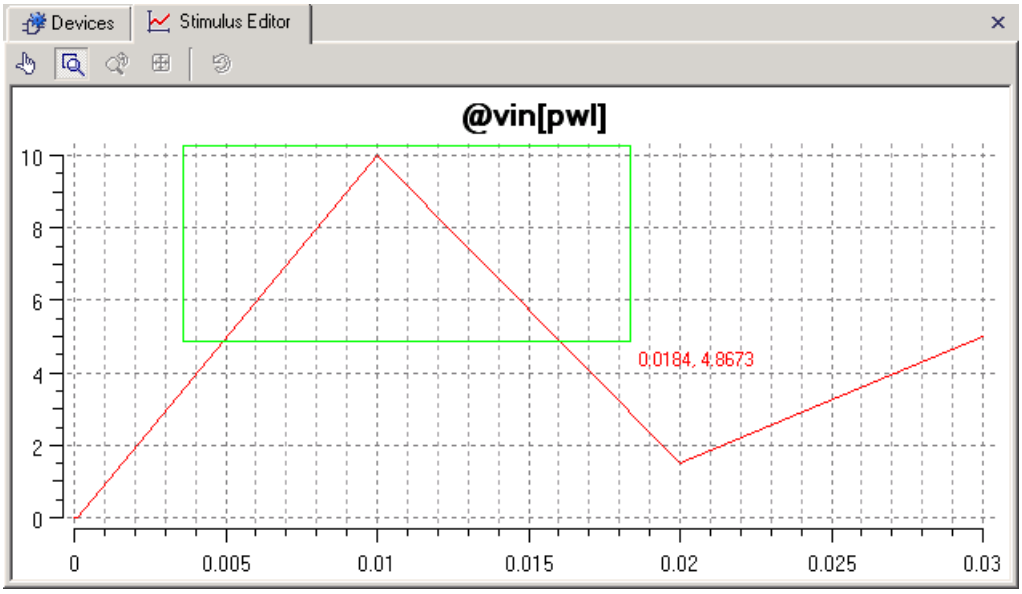


Figure 30-6 Stimulus Editor - Zoom Mode

The Stimulus Editor local toolbar's buttons from left to right are:

Figure 30-7 Stimulus Editor - Toolbar

- Edit mode - enable the Edit mode
- Zoom mode - enables the Zoom mode
- Zoom back - zooms back to the previous view state
- Zoom full - shows entire stimulus fitting it into the view
- Apply changes - after the stimulus is edited. Click it to bring changes into current circuit.
- View data - get a listing of a data
- X, Y - edit values of a current node

The scale type for either axis can be switched from Linear to Logarithmic by double-click on it.

30.3 Acquiring Data from a Vector

You can acquire data from a vector and bring it into Stimulus Editor with possibility to bind it to some PWL parameter of current circuit. To do this, right click on a vector in **Plots** pane of Tasks view (**View→Tasks List**):

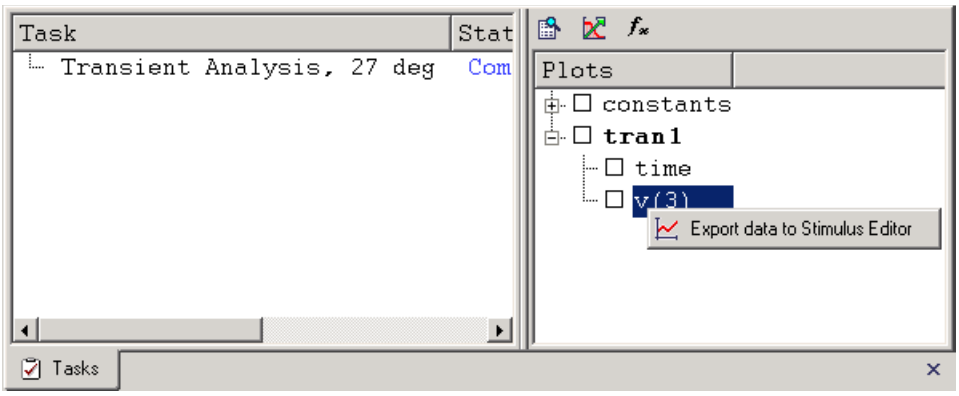


Figure 30-8 Inject Vector's Data into Stimulus Editor

From the context menu, select **Export data to Stimulus Editor**. The Stimulus Editor will be brought on top and the contents of the vector is plotted.

Note: You can't apply edited data back to the vector, but you can apply it to PWL device parameter after binding it.

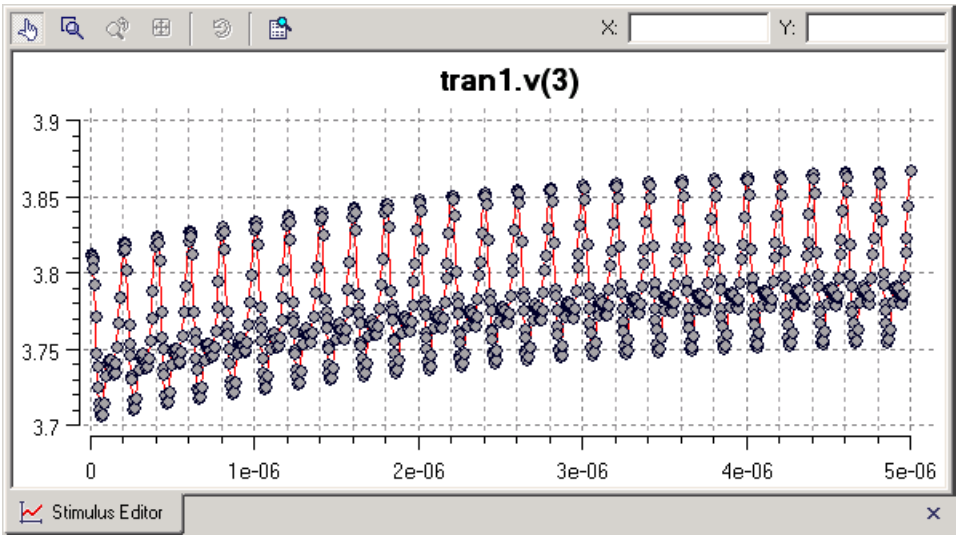


Figure 30-9 Stimulus Editor with Data from a Vector

30.4 Bind PWL Parameter with Stimulus Editor

To apply data from Stimulus Editor to PWL device parameter, right click on this parameter in **Parameters** pane of **Devices** view (**View→Circuit→Devices**) and select **Bind to Stimulus Editor**:

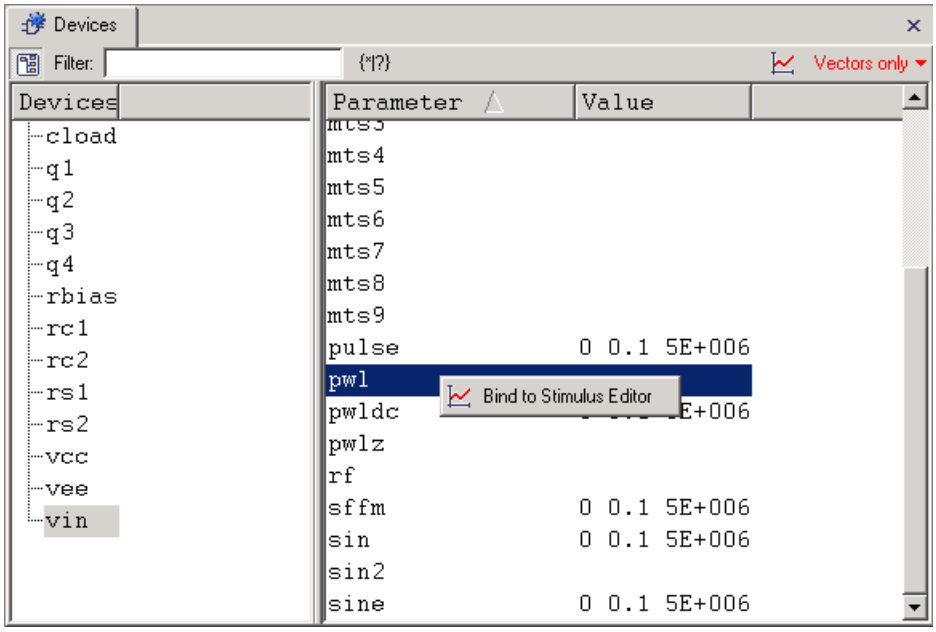


Figure 30-10 Bind Parameter to Stimulus Editor

After parameters is bound, the data can be edited in Stimulus Editor and applied to PWL parameter of current circuit on a regular basis.

You can print out data from Stimulus Editor in text format and save it to file to make it included in a netlist. Use the **View data** button on Stimulus Editor's toolbar.

30.5 Property Sheets for Complex Device Parameters

For complex device parameters there is a way to edit their components individually by means of property sheets and preview entire curve with Stimulus Editor. Currently this functionality is available for PULSE and SINE device parameters. Press **Enter** or double-click on a parameter to open auxiliary pane with a list view representing selected parameter dissected into components:

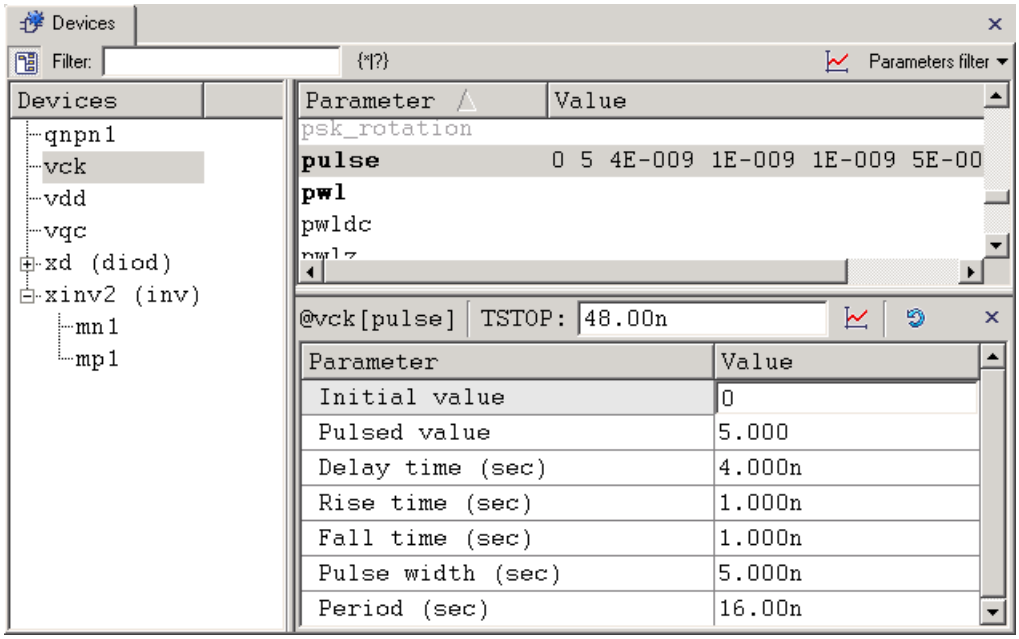


Figure 30-11 PULSE Property Sheet

The toolbar contains the following controls:

- Name of parameter
- TSTOP edit box - enter the range along X-axis on which the Stimulus Editor should display the parameter's curve.
- Call Stimulus Editor button to preview the parameter
- Apply changes back to circuit button
- Close button

The previous configuration, when previewed with Stimulus Editor, will render as follows:

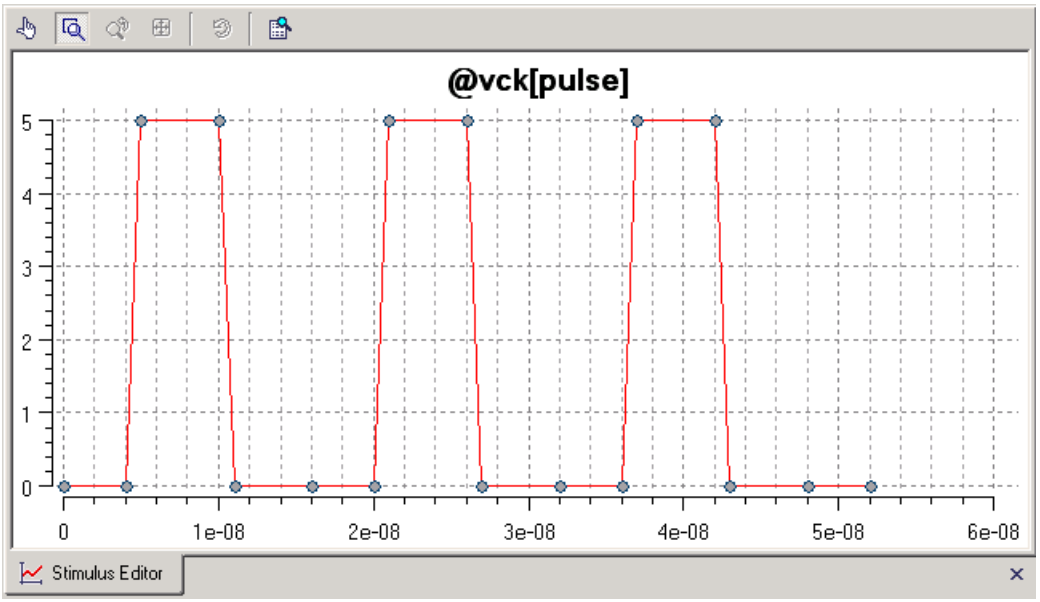


Figure 30-12 Stimulus Editor - PULSE

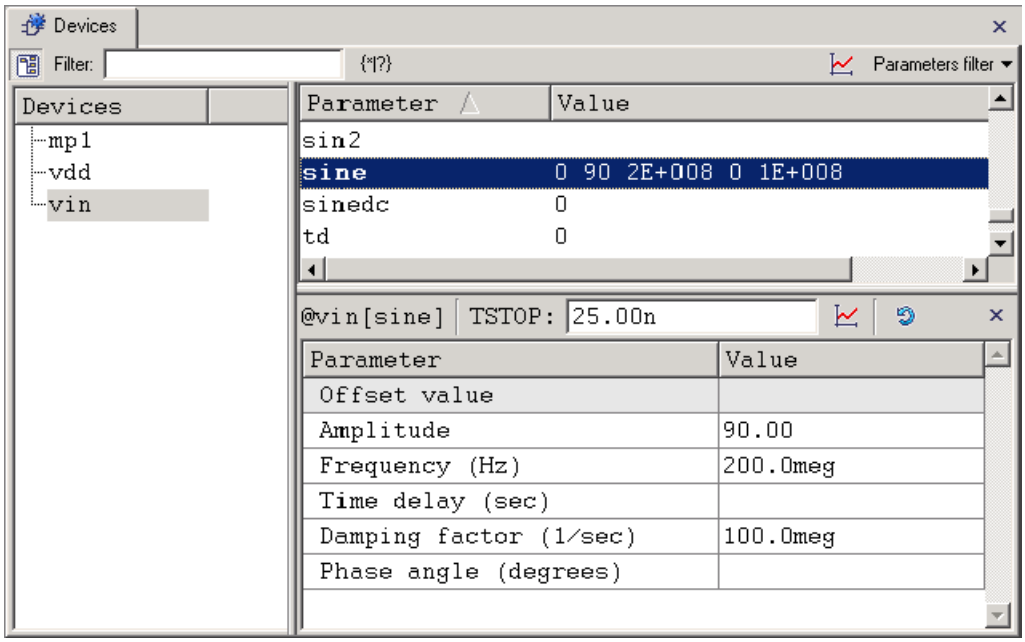


Figure 30-13 SINE Property Sheet

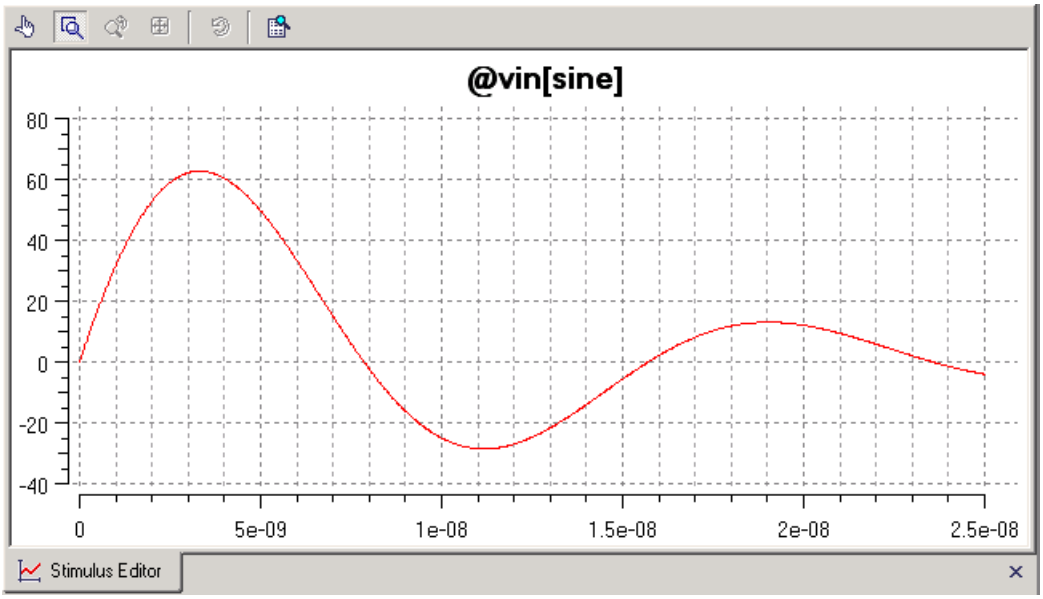


Figure 30-14 Stimulus Editor - SINE

Chapter 31
SmartSpice RLC-Reduction Tool

31.1 Introduction

SmartSpice has a built-in RLC-reduction tool that can be used on input decks that contain RLC parasitic elements. Previously, an external tool was required to reduce RLC parasitics before sourcing the deck in SmartSpice. Now SmartSpice has this functionality built-in and can perform RLC parasitic reductions without the need to use an external application.

Subcircuit-level and top-level RLC-reduction is supported. The RLC-reduction works for expanded netlists. The RLC-reduction tool will not work with any kind of input deck. At the moment, only decks produced by LPE parasitic extraction tools are supported.

Current revision limitations:

- Deck must contain linear RLC elements.
- Resistor, capacitor and inductor values must be plain numbers; expressions and parameters are not supported.
- Current implementation method doesn't support mutual inductance.

Note: A Clarity RLC license is required for tool invocation. If the license is not available, the tool will be blocked and SmartSpice will proceed without performing RLC reduction.

31.2 Reduction Technique

This RLC reduction technique uses Scattering-Parameter-based Macromodeling or Time domain Method, depending on which produces a better reduction. The RLC-reduction workflow is as follows:

- Time domain method is tried first with nodal time constant specified in the option file. If nodal time constant is not specified in the option file, the default value 1e-12 is used.
- If the TM method does not produce acceptable results for netlists which do not contain L elements, RLC-reduction tries to use the SPM method with the resistance and capacitance thresholds specified in the option file. The SPM method can not reduce L elements, only TM method can be used for reduction of L elements.

If nodal time threshold is set to 0 in the option file, then only SPM method is used for : RLC-reduction.

This functionality is embedded in SmartSpice as well as available in a separate product ClarityRLC (see the ClarityRLC User's Manual).

31.2.1 Default Method

In addition to SPM method, SmartSpice implements TM method, which is the default. RLC-reduction defaults to Time domain Method (TM) with the `TAU_MIN_RC` option set to 1e-12.

The full workflow is:

- Time domain method is tried first with the `TAU_MIN_RC` option set to 1e-12.
- If TM method does not produce acceptable results, RLC-reduction tool reverts back to Scattering Parameter Method (SPM) with the option defaults `RMIN_RC=1e-3`, `CMIN_RC=1e-22` and `LMIN_RC=1e-9`.

This reduction process can be controlled by manually setting options `RMIN_RC`, `CMIN_RC`, `LMIN_RC`, `METHOD_RC` and `TAU_MIN_RC` in the input deck. Set `METHOD_RC=0` for SPM method, `METHOD_RC=1` (default) for TM method, and `METHOD_RC=2` to omit main reduction.

The Time domain Method (TM) can only be used for the reduction of parasitic networks that include L elements. In this case, we use RL and C branches in parallel instead of R and C branches. Each RL branch can have zero R or L parasitic elements. If all Ls are zero, we then have a : RLC-reduction problem. For an RLC reduction problem, each node in the parasitic network has two time constants. The first constant is defined by R and C parasitic elements. The second one is defined by R and L parasitic elements. The minimal value from these two constants is assigned to the parasitic node. The node will be eliminated if this value is less than nodal time threshold specified in the option file.

Note: RLC-reduction tool reverts back to SPM method if TM is selected but `TAU_MIN_RC` is set to 0.

31.2.2 DSPF Annotated Netlist RLC-reduction

By invoking SmartSpice with `-rclevel 6` or `7` command line switch or with `.option rclevel 6` or `7`, RLC-reduction can be performed on DSPF-annotated netlists. In batchmode, reduction is done for each annotated `*|NET` section in the netlist. If run in GUI mode, SmartSpice pops up a message allowing you to select nets in a list for reduction. Method selection and option control are the same for this feature as for other `-rclevel X` invocation options. See Chapter 2 Graphical User Interface.

References

1. H.Liao and W. Wei-Ming Dai, *Partitioning and Reduction of RC Interconnect Networks Based on Scattering Parameter Macromodels*, Computer Engineering, University of California, Santa Cruz.

31.3 Tool Invocation and Workflow

To invoke RLC-reduction tool SmartSpice needs to be launched with `'-rclevel <level_number>'` command line option or with `'.option rclevel=level_number'`.

Example 1

```
smartspice -rclevel 0
```

Example 2

```
.option rclevel=3
```

Table 31-1:

Level Number	RLC-reduction on subcircuits	RLC-reduction on top-level netlist	RLC-reduction on expanded netlist	DSPF	Post-filtering
0 (default)	+				
1	+				+
2	+	+			
3	+	+			+
4			+		
5			+		+
6				+	
7				+	+

For `-rclevel 0 ... 3` SmartSpice invokes RLC-reduction after sourcing input deck. For `-rclevel 4` or `5` SmartSpice invokes RLC-reduction after subcircuit expansion for all expanded netlist. Each subcircuit, top-level or expanded netlist will be RLC-reduced and the circuit will be modified accordingly.

Before RLC-reduction, RLC-reduction multiplier algorithm is invoked. It works for top-level and expanded netlists. It changes resistance, capacitance or inductance for devices with multiplier before RLC-reduction. If multiplier value is a number, SmartSpice changes resistance, capacitance or inductance value and removes multiplier from the input line. For example, the initial input line:

```
r1 1 2 500 m=5
```

will be modified to:

```
r1 1 2 100
```

If multiplier value is expression, SmartSpice adds `res=` or `cap=` string before resistance, capacitance or inductance value to suppress RLC-reduction for this line. For example, the initial input line:

```
r1 1 2 500 m='1/par1'
```

will be modified to:

```
r1 1 2 res=500 m='1/par1'
```

The following is the workflow description:

- All ground node names (`0`, `gnd`, `gnd!`, `ground`) in subcircuit definition are forced to name `'0'`.
- Only external nodes, output dot statements nodes (for top-level and expanded netlist) and 0 reference node are retained; the RLC-reduction tool may produce different number and names for internal subcircuit nodes.
- RLC-reduction multiplier algorithms are forced before RLC-reduction for top-level and expanded netlist.
- Original subcircuit, top-level or expanded netlist is replaced with an RLC-reduced one and statistics are printed.
- For `-rclevel 0 ... 3`, the previous process is performed for the next subcircuit (top-level netlist) until all of them are processed.

31.3.1 rclevel Confirmation Messages

When RLC-reduction tool has been activated, SmartSpice types the following messages:

- `rclevel=0` - “RLC-reduction for subcircuits is running”
- `rclevel=1` - “RLC-reduction for subcircuits with post-filtering is running”
- `rclevel=2` - “RLC-reduction for subcircuits and top-level is running”
- `rclevel=3` - “RLC-reduction for subcircuits and top-level with post-filtering is running”
- `rclevel=4` - “RLC-reduction for expanded netlist is running”
- `rclevel=5` - “RLC-reduction for expanded netlist with post-filtering is running”
- `rclevel=6` - “RLC-reduction for DSPF annotated netlist is running”
- `rclevel=7` - “RLC-reduction for DSPF annotated netlist with post-filtering is running”

31.3.2 Command Option -rcdump

The command option `-rcdump` forces SmartSpice to save its expanded listing into the file `rcdump.in` after RLC-reduction.

Example

```
smartspice -rclevel 3 -rcdump
```

31.3.3 Command rcdump

Syntax

```
rcdump
```

When the build-in RLC-reduction tool changed the topology of subcircuit, top-level or expanded netlist, SmartSpice generates for them two files which contain original and modified netlists.

The file name for the original netlist is `rcdump.original`. The file name for the modified netlist is `rcdump.reduced`.

SmartSpice does not save any information into these files for unmodified subcircuit, top-level or expanded netlist.

This command works only when the option `rclevel` is specified in command line and circuit has been sourced.

31.3.4 Options RMIN_RC, CMIN_RC, LMIN_RC

When RLC-reduction is performed, elements are first filtered according to supplied R, C and L values threshold. By default, these values are `1e-3` for resistors, `1e-22` for capacitors and

1e-9 for inductors. RLC-reduction tool shortens all resistors and inductors, and opens all capacitors with values below those thresholds. Thresholds can be adjusted using the input deck options RMIN_RC, CMIN_RC and LMIN_RC. These options control not only pre-reduction element filtering, but also post-reduction filtering as well when -rclevel 1 3 or 5 is specified.

Example

```
.OPTIONS RMIN_RC=1 CMIN_RC=1e-15
```

This example will cause resistors < 1 ohm and capacitors < 1 fF to be filtered out before (and after) RLC-reduction.

31.3.5 RLC-reduction Warning Message

When a device number increased during RLC-reduction, SmartSpice issues the following warning message:

```
Warning: RLC-reduction: In <subckt_name> subcircuit number of <R, C
or L> devices increased.
```

31.3.6 Using SmartSpice GUI for Selection Subcircuits and DSPF Nets for RLC-reduction

SmartSpice provides Selectivity tool for subcircuits and DSPF nets which will be submitted for RLC-reduction. See Chapter 2 Graphical User Interface.

31.3.7 Optimized RLC-reduction

SmartSpice provides optimized RLC-reduction algorithm. SmartSpice will automatically adjust RMIN_RC option to get the most reduced resistors network. SmartSpice will print the adjusted RMIN_RC option value in the statistics output table.

To invoke optimized RLC-reduction algorithm, SmartSpice needs to be launched with the -rc_optimize command line option.

Example

```
smartspice -rclevel 0 -rc_optimize
```

The option RMAX_RC is used to control the maximum value of the RMIN_RC option during optimized RLC-reduction. Default is 10.

Example

```
.OPTION RMAX_RC=5
```

For the example above, the maximum value of the RMIN_RC option during optimized RLC-reduction will be 5 ohm.

The optimized RLC-reduction algorithm consists of the following steps:

1. SmartSpice collects all different parasitic resistor values, which are between RMIN_RC and RMAX_RC, and store them in the array in ascending order.
2. SmartSpice picks up the smallest resistor value from the saved array, adds a small value to it, sets RMIN_RC option to this value, and runs RLC-reduction with the new RMIN_RC option value.
3. Step 2 repeats for the largest resistor value.
4. SmartSpice analyzes the number of resistors in the reduced resistors networks from the previous runs, and based on the half-interval search algorithm picks up the next resistor

value from the array, adds a small value to it, sets RMIN_RC option to this value, and runs RLC-reduction with the new RMIN_RC option value.

5. The step 4 repeats, until the most reduced resistors network will be found.

31.3.8 Parallel MOSFETs Reduction

SmartSpice provides parallel MOSFETs reduction algorithm. The option REDUCE_PARALLEL_MOS=1 should be used to activate parallel MOSFETs reduction.

Example

```
.OPTION REDUCE_PARALLEL_MOS=1
```

The original netlist:

```
MP1 VDD A Y VDD W=15U L=2U AD=55P PD=31U AS=55P PS=31U
MP2 VDD A Y VDD W=15U L=2U AD=45P PD=26U AS=45P PS=26U
MP3 VDD A Y VDD W=15U L=2U AD=45P PD=26U AS=45P PS=26U
MP4 VDD A Y VDD W=15U L=2U AD=55P PD=31U AS=55P PS=31U
```

The created netlist after parallel MOSFETs reduction:

```
MP1 VDD A Y VDD W=15U L=2U AD=50P PD=28.5U AS=50P PS=28.5U M=4
```

31.4 Statistic Output

The following is a statistic output example. It was produced for an input deck containing subcircuits with RLC parasitics. SmartSpice was run with the `-rclevel 0` command line switch. `RMIN_RC` and `CMIN_RC` options were set in the input deck to values 10 and 1e-017.

Statistic shows columns with initial number of R, C and L elements, reduced numbers of R, C and L elements, a ratio of element reduction, reduction status, a reduction percentage, a time used for reduction and a subcircuit name. Each line in the table represents subcircuit. The ratio is computed as “reduced number of elements” divided by “initial number of elements”. SmartSpice prints “`!`” in the status column if ratio > 1, or “`OK`” if ratio <= 1. The reduced elements percentage is computed for all R, C and L elements combined. A total line contains total statistic for all processed subcircuits. Finally, an RLC-Reduction option values will be printed for reference.

See the following page for a Statistics Output Table example.

```
-----------------------------------------------------------------------------------------------------------------------------
RLC reduction statistics for linear R C and L
-----------------------------------------------------------------------------------------------------------------------------
              R                         |              C                         |              L                         | Reduce |  Elapsed | Subcircuit name
 Before  |  After  | Ratio|Status|  Before  |  After  | Ratio|Status|  Before  |  After  | Ratio|Status|    %   |  time, s |
-----------------------------------------------------------------------------------------------------------------------------
       1 |       1 | 1.00 |  OK  |        0 |       0 |  -   |  -   |        0 |       0 |  -   |  -   |  0.00  |     0.13 | nfet_b
      13 |      13 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | zvtnfet
      13 |      13 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | zvtdgnfet
       3 |       3 | 1.00 |  OK  |        3 |       3 | 1.00 |  OK  |        1 |       1 | 1.00 |  OK  |  0.00  |     0.05 | vpp
       1 |       1 | 1.00 |  OK  |        0 |       0 |  -   |  -   |        0 |       0 |  -   |  -   |  0.00  |     0.00 | subc
      28 |      28 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |       28 |      28 | 1.00 |  OK  |  0.00  |     0.03 | singlewire
       5 |       5 | 1.00 |  OK  |        0 |       0 |  -   |  -   |        0 |       0 |  -   |  -   |  0.00  |     0.00 | sblkndres
      13 |      13 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | pfet33
      10 |      10 | 1.00 |  OK  |        3 |       3 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.02 | oprrpres
       8 |       8 | 1.00 |  OK  |        3 |       3 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | opppcres
       5 |       5 | 1.00 |  OK  |        0 |       0 |  -   |  -   |        0 |       0 |  -   |  -   |  0.00  |     0.00 | opndres
      13 |      13 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | nfet33
      13 |      13 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.02 | pfet
      13 |      13 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | nfet
       7 |       7 | 1.00 |  OK  |        3 |       3 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.01 | ncap
      11 |      11 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | lppfet
      11 |      11 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.02 | lpnfet
       3 |       3 | 1.00 |  OK  |        2 |       2 | 1.00 |  OK  |        1 |       1 | 1.00 |  OK  |  0.00  |     0.00 | lmmimcap
      18 |      18 | 1.00 |  OK  |       11 |      11 | 1.00 |  OK  |       14 |      14 | 1.00 |  OK  |  0.00  |     0.00 | indstack
      17 |      17 | 1.00 |  OK  |       10 |      10 | 1.00 |  OK  |       12 |      12 | 1.00 |  OK  |  0.00  |     0.02 | indline
       8 |       8 | 1.00 |  OK  |        2 |       2 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | havar
       2 |       2 | 1.00 |  OK  |        3 |       3 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | esdvpnpnw
       2 |       2 | 1.00 |  OK  |        1 |       1 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | esdndsx
       1 |       1 | 1.00 |  OK  |        0 |       0 |  -   |  -   |        0 |       0 |  -   |  -   |  0.00  |     0.00 | efuse
       1 |       1 | 1.00 |  OK  |        0 |       0 |  -   |  -   |        0 |       0 |  -   |  -   |  0.00  |     0.00 | diodepwpi
       1 |       1 | 1.00 |  OK  |        0 |       0 |  -   |  -   |        0 |       0 |  -   |  -   |  0.00  |     0.00 | diodepisx
      11 |      11 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.01 | dgpfet
      11 |      11 | 1.00 |  OK  |        8 |       8 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | dgnfet
       7 |       7 | 1.00 |  OK  |        3 |       3 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.02 | dgncap
      56 |      56 | 1.00 |  OK  |       24 |      24 | 1.00 |  OK  |       56 |      56 | 1.00 |  OK  |  0.00  |     0.03 | coupledwires
       4 |       4 | 1.00 |  OK  |        4 |       4 | 1.00 |  OK  |        0 |       0 |  -   |  -   |  0.00  |     0.00 | bondpad
-----------------------------------------------------------------------------------------------------------------------------
     310 |     310 | 1.00 |  OK  |      160 |     160 | 1.00 |  OK  |      112 |     112 | 1.00 |  OK  |  0.00  |     0.34 | Total
-----------------------------------------------------------------------------------------------------------------------------
Reduction parameters:
-----------------------------------------------------------------------------------------------------------------------------
        rmin_rc | 10
        lmin_rc | 1e-009
        cmin_rc | 1e-017
        rmax_rc | 10
      method_rc | Time domain Method (TM)
     tau_min_rc | 1e-012
 reduce_all_rlc | 1
-----------------------------------------------------------------------------------------------------------------------------
```

31.5 RLC Reduction - Interactive GUI Method

SmartSpice allows you to control an : RLC-reduction flow from dialog windows in GUI mode.

The following 2 settings activate the dialog:

1. Invoke SmartSpice with command line switch `'rclevel x'` (x=6 or 7 for DSPF format).
2. In the GUI menu **Edit→Preferences→Simulation**, tick the **Show : RLC dialog** box.

When a deck containing : RLC or DSPF information is sourced, the following dialog will appear:

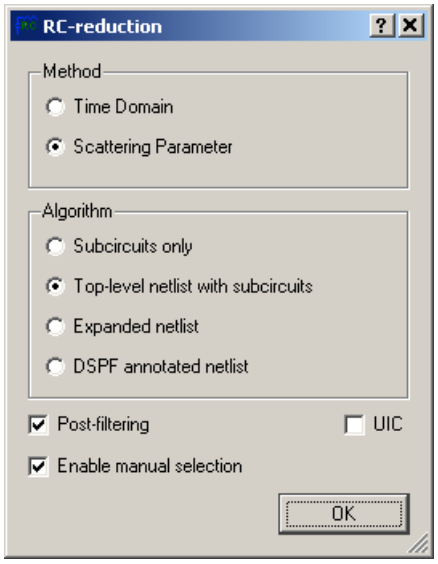


Figure 31-1 : RLC-reduction Dialog

From here, a Method of : RLC-reduction can be chosen:

- **Time Domain**
- **Scattering Parameter**

As well as an Algorithm:

- **Subcircuits only**
- **Top-level netlist with subcircuits**
- **Expanded netlist**
- **DSPF annotated netlist**

The following options can be checked:

- **Post-filtering:** Turns on/off post-filtering for chosen algorithm.
- **UIC:** Start transient from user-specified initial conditions.
- **Enable manual selection:** If this option is checked, another dialog will appear allowing you to choose between individual entries to process with : RLC-reduction. It's contents depend on a selected algorithm. If **Subcircuits only** and **Top-level netlist with subcircuits** are selected, the **Subcircuits** list dialog will appear (see Figure 31-2).

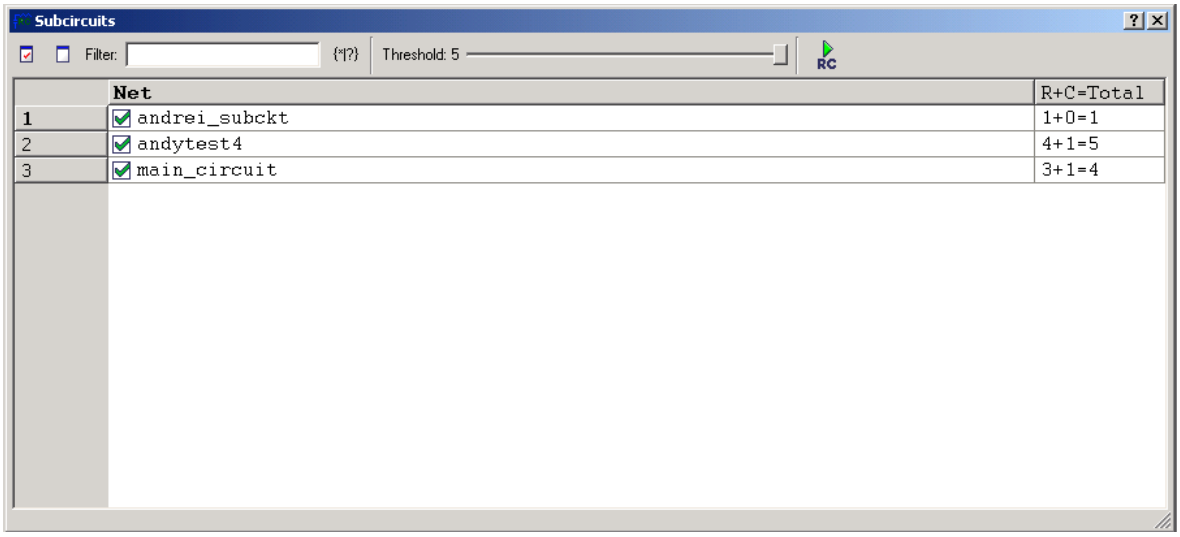


Figure 31-2 Subcircuits List Dialog

If **DSPF annotated netlist** algorithm is selected, the **DSPF Nets** list dialog will appear.

Example

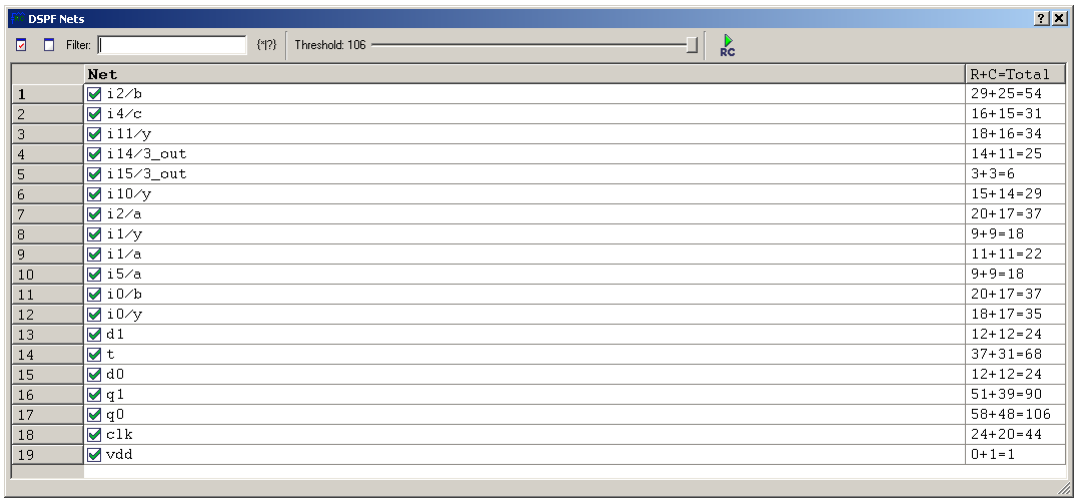
```
*|NET XxxxxCT_0/XxxxxPY_02/XLVPPL_0/XMVPPCNT2_0/XIDL05_ON_22
/NET013 0.000697177PF
```

The deck being sourced must contain the DSPF nets in order to proceed through : RLC-reduction with chosen algorithm.

No dialog will appear if there are no entries to display, or if the **Expanded netlist** algorithm is applied.

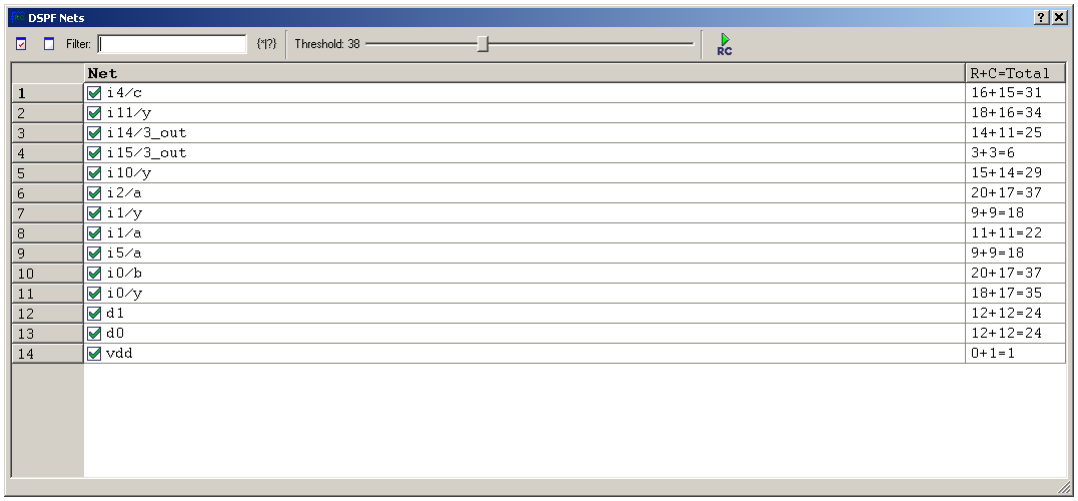
The checking/unchecking of individual entries should be performed with the left-mouse button or the Space Bar on the keyboard.

- **Check All:** Check all displayed entries.
- **Uncheck All:** Uncheck all displayed entries.
- **Filter:** A long list of sub circuits or DSPF nets can be filtered to provide quicker access to matching entries by applying a filter expression.
- **Use regular expression:** Check in order to use a regular expression to provide a greater control over the filtered list.
- **Threshold:** A filter by an amount of congregated quantity of resistances and capacitors, as it being represented in the second column. The entries with a greater value than denoted by the slider are cut off from a display and successive processing.
- **Proceed with : RLC reduction:** Press the RC button in order to proceed. No canceling is provided at this stage.



	Net	R+C=Total
1	i2/b	29+25=54
2	i4/c	16+15=31
3	i11/y	18+16=34
4	i14/3_out	14+11=25
5	i15/3_out	3+3=6
6	i10/y	15+14=29
7	i2/a	20+17=37
8	i1/y	9+9=18
9	i1/a	11+11=22
10	i5/a	9+9=18
11	i0/b	20+17=37
12	i0/y	18+17=35
13	d1	12+12=24
14	t	37+31=68
15	d0	12+12=24
16	q1	51+39=90
17	q0	58+48=106
18	clk	24+20=44
19	vdd	0+1=1

Figure 31-3 DSPF : RLC Threshold



	Net	R+C=Total
1	i4/c	16+15=31
2	i11/y	18+16=34
3	i14/3_out	14+11=25
4	i15/3_out	3+3=6
5	i10/y	15+14=29
6	i2/a	20+17=37
7	i1/y	9+9=18
8	i1/a	11+11=22
9	i5/a	9+9=18
10	i0/b	20+17=37
11	i0/y	18+17=35
12	d1	12+12=24
13	d0	12+12=24
14	vdd	0+1=1

Figure 31-4 DSPF : RLC Reduced Threshold Level

```
RC reduction statistics for linear R and C
                R                |                C                | Reduce |  Elapsed | Subcircuit name
 Before  |  After  | Ratio|Status| Before  |  After  | Ratio|Status|   %    |  time, s |
      29 |       1 | 0.03 |  OK  |      25 |       2 | 0.08 |  OK  |  94.44 |     0.9: | i2/b
      16 |       1 | 0.06 |  OK  |      15 |       2 | 0.13 |  OK  |  90.32 |     0.70 | i4/c
      18 |       1 | 0.06 |  OK  |      16 |       2 | 0.13 |  OK  |  91.18 |     0.55 | i11/y
      14 |       1 | 0.07 |  OK  |      11 |       2 | 0.18 |  OK  |  88.00 |     0.48 | i14/3_out
       3 |       0 | 0.00 |  OK  |       3 |       1 | 0.33 |  OK  |  83.33 |     0.45 | i15/3_out
      15 |       1 | 0.07 |  OK  |      14 |       2 | 0.14 |  OK  |  89.66 |     0.80 | i10/y
      20 |       1 | 0.05 |  OK  |      17 |       2 | 0.12 |  OK  |  91.89 |     0.52 | i2/a
       9 |       1 | 0.11 |  OK  |       9 |       2 | 0.22 |  OK  |  83.33 |     0.42 | i1/y
      11 |       1 | 0.09 |  OK  |      11 |       2 | 0.18 |  OK  |  86.36 |     0.44 | i1/a
       9 |       1 | 0.11 |  OK  |       9 |       2 | 0.22 |  OK  |  83.33 |     0.6: | i5/a
      20 |       1 | 0.05 |  OK  |      17 |       2 | 0.12 |  OK  |  91.89 |     0.47 | i0/b
      18 |       1 | 0.06 |  OK  |      17 |       2 | 0.12 |  OK  |  91.43 |     0.42 | i0/y
      12 |       1 | 0.08 |  OK  |      12 |       2 | 0.17 |  OK  |  87.50 |     0.47 | d1
      37 |       7 | 0.19 |  OK  |      31 |       4 | 0.13 |  OK  |  83.82 |     0.48 | t
     231 |      19 | 0.08 |  OK  |     207 |      29 | 0.14 |  OK  |  89.04 |     7.72 | Total
Reduction parameters:
    rmin_rc | 0.001
    cmin_rc | 1e-022
  method_rc | Time domain Method (TM)
 tau_min_rc | 1e-012
```

Figure 31-5 : RLC-reduction Output in SmartSpice

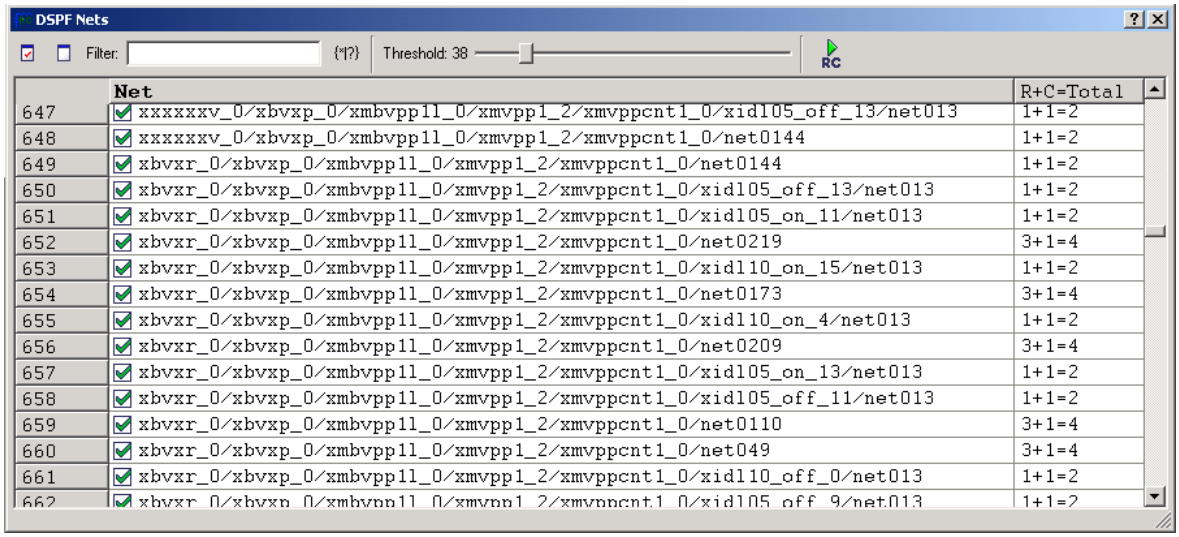


Figure 31-6 DSPF Nets List Dialog

31.6 Interconnect RC Networks Reduction

Starting from 4.0.1.R, SmartSpice has internal interconnect RC networks reduction feature. To enable this feature, option `int_rc_method` should be set to 1 or 2. The value specifies the implementation algorithm. Depending on `int_rc_method`, the RC networks around the grounded capacitors will be analyzed and circuit topology will be changed.

The two additional options can be specified to control reduction.

The option `int_rc_cmin` specifies the capacitance threshold. All grounded capacitors with capacitance less than this threshold will be removed. Default `int_rc_cmin` is 1e-22.

The option `int_rc_rmin` specifies the resistance threshold. Resistors will be reduced (or transformed) with resistance less than resistance threshold. Default `int_rc_rmin` is 1e-3.

If reduction feature is activated, SmartSpice will print out the following information during processing the input netlist with interconnect RC networks:

```
Internal RC-reduction starting...
-----------------------------
Internal RC-reduction: detecting nodes with grounded capacitors and
number of connections < 4 ...
Internal RC-reduction: grounded capacitors with capacitance less
than 1e-15 are : 5841
Internal RC-reduction: analyzing RC networks...
Internal RC-reduction: RC network chains: 5758
Internal RC-reduction: maximum length of RC networks: 3

Internal RC-reduction: TOTAL STATISTICS :
-------------------------------------------
Internal RC-reduction: capacitors before : 62706
Internal RC-reduction: capacitors removed : 5697
Internal RC-reduction: capacitors reduction ratio (%): 9.08525 %
Internal RC-reduction: resistors before : 100046
Internal RC-reduction: resistors removed : 815
Internal RC-reduction: resistors reduction ratio (%) : 0.814625
Internal RC-reduction: nodes before : 83101
Internal RC-reduction: nodes removed : 815
-------------------------------------------
```

31.6.1 Algorithm Description

Common Part

The common part for two algorithms is detecting RC networks satisfying the specific predefined condition. First, SmartSpice starts by detecting nodes with grounded capacitors and number of connections < 4. Second, SmartSpice leaves only capacitors with capacitance > `int_rc_cmin`. Final, SmartSpice detects all RC networks connected to such capacitors and links such networks in chain. In output above the total count of such chains is reported: "Internal RC-reduction: RC network chains: 5758". The maximum length of chain is also reported. After that, SmartSpice traverses through the all RC networks in single chain and transforms the topology of all RC devices in a chain using a corresponding reduction technique.

Reduction Method 1

The first reduction method is defined by using option `int_rc_method=1`. If specified; first, all capacitors in chain will be removed; then all internal nodes in the chain will be analyzed and all internal resistors will be disconnected if it's possible. The final resistor with connection to the external node of processing RC chain will be left and its resistance will be adjusted to Rtotal=Sum(Ri), where Ri - resistance of all internal serial resistors with resistance greater than `int_rc_rmin`. Then all unneeded internal nodes with number of connections is equal to zero will be removed. The final resistor will be connected to the external nodes of processing chain. See Figure 31-7 and Figure 31-8 for illustrated description of this process in example 1.

Reduction Method 2

The second reduction method is defined by using option `int_rc_method=2`. The difference with method 1 is following. First, after analyzing their resistance, all internal serial resistors will be removed and the new final resistor will be created with nodes connected to the external nodes of processing RC chain. Second, the resistance Rtotal of final resistor will be calculated based on all internal serial resistors.

After reduction SmartSpice will report the final reduction statistics with number of removed capacitors, resistors and nodes. Note, if the basic reduction is not possible, SmartSpice will not change the topology.

Examples

Example 1

```
.option int_rc_method=1 int_rc_cmin=1e-18 int_rc_rmin=0.01
….
R1       Extnode1   Node1     0.001
C1       Node1      0         1e-19
R2       Node1      Node2     10K
C2       Node2      0         1e-19
R3       Node2      Extnode2  10K
C3       Extnode2   0         1e-19
…
```

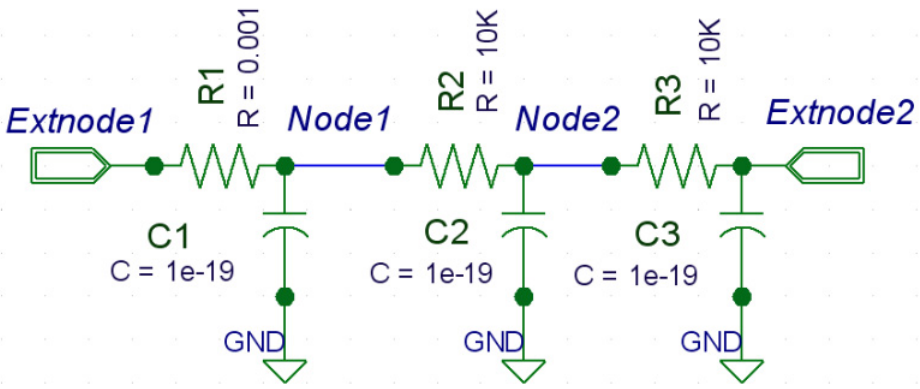


Figure 31-7 Example 1 - original block

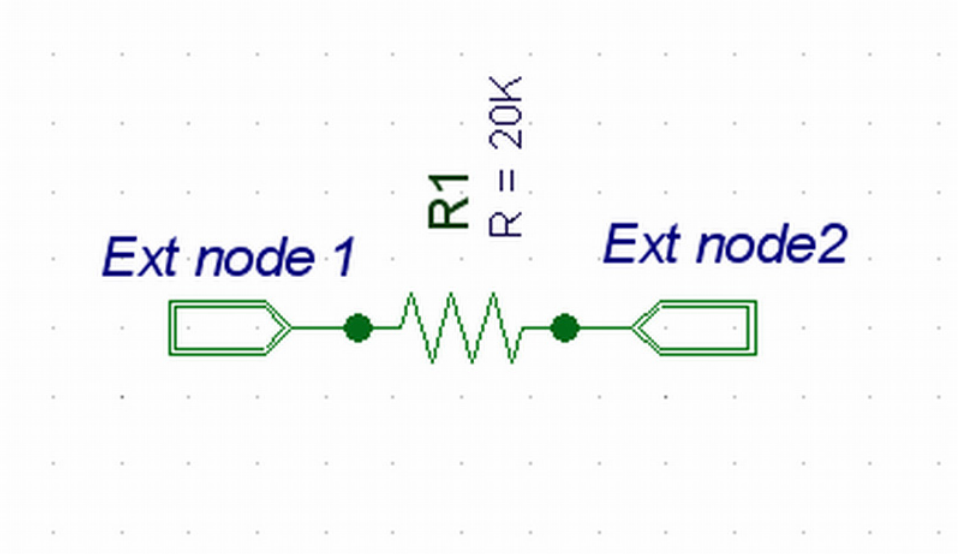


Figure 31-8 Example 1 - reduced block

SmartSpice Output

```
Internal RC-reduction starting...
-----------------------------
Internal RC-reduction:detecting nodes with grounded capacitors and
t_numconn < 4 ...
Internal RC-reduction:grounded capacitors with capacitance less
than 1e-018 are: 3
Internal RC-reduction: analyzing RC networks...
Internal RC-reduction: RC network chains: 1
Internal RC-reduction: maximum length of RC networks: 3

Internal RC-reduction: TOTAL STATISTICS :
-------------------------------------------
Internal RC-reduction: capacitors before : 3
Internal RC-reduction: capacitors removed : 3
Internal RC-reduction: capacitors reduction ratio (%): 100
Internal RC-reduction: resistors before : 4
Internal RC-reduction: resistors removed : 2
Internal RC-reduction: resistors reduction ratio (%) : 50
Internal RC-reduction: nodes before : 6
Internal RC-reduction: nodes removed : 2
-------------------------------------------
```

Example 2

```
.option int_rc_method=2 int_rc_cmin=1e-18 int_rc_rmin=0.5
….
R1       Extnode1   Node1     1
C1       Node1      0         1e-19
R2       Node1      Node2     10K
C2       Node2      0         1e-19
R3       Node2      Extnode2  10K
C3       Extnode2   0         1e-19
   …
```

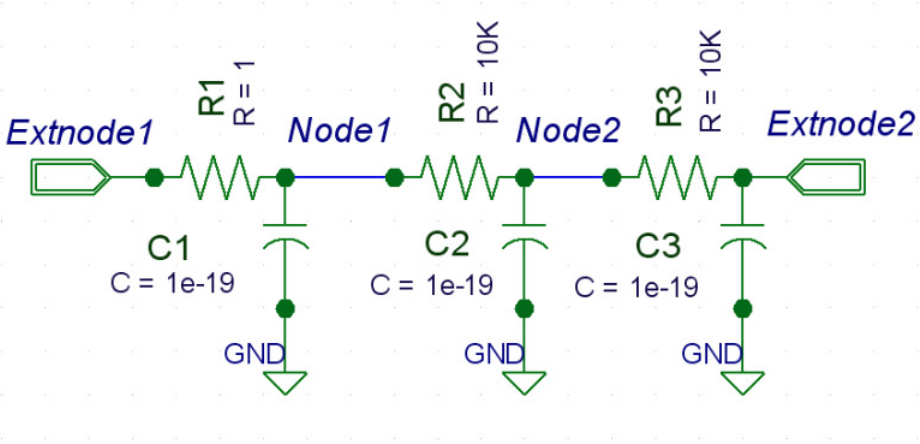


Figure 31-9 Example 2 (method 2) - original block

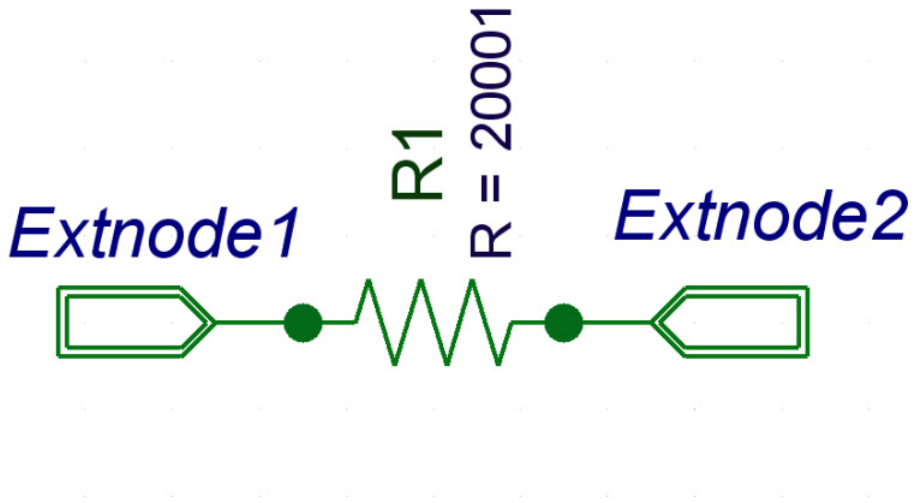


Figure 31-10 Example 2 (method 2) - reduced block

Conclusions and Recommendations

The Node reduction method must be applied to a specific RC named interconnect topology.

The main purpose of the methods described is a topology reduction implemented as circuit node reducer that in turn is an equation reducer. Some control option must be specified to activate an automatic reducer which automatically detects interconnects in libraries, netlists, include files, and applies reduction algorithm to generate reduced equivalent replacement of original interconnect network.

The performance speed up in model load and LU-decomposition could be observed after successful reduction.

Information from SmartSpice output during reduction (Internal RC-reduction: nodes removed) and `.option acct=2` helps to determine the number of eliminated circuit equations from the original circuit.

Threshold on RC allows (`int_rc_cmin` and `int_rc_rmin`) controlling the ratio of reduction. Larger thresholds will generate more compressed interconnect equivalent circuits. Breakdown may occur if not reasonably large thresholds are used. SmartSpice is not giving any recommendations for RC thresholds. You must check simulation results after applying RC reducer.

Chapter 32
Simulation Statistics Output

32.1 Regular SmartSpice Statistics

At the end of a simulation, SmartSpice prints out the following table:

```
AN.PHASE   |time   |totiter |totstep  |CONrej DEVrej  MATrej  LTErej   DERrej  breaks
TRAN       1.53    4126     1307      0       0       0       56        0       46
DCOP       0.016   45        0        0       0
```

AN.PHASE	Analysis phase or analysis name.
time	Wall clock time spent in the analysis.
totiter	Total number of iterations analysis performed completing simulation.
totstep	Total number of timepoints calculated by transient or number of frequencies for AC or number of sweeps for DC.
CONrej	Convergence rejection. The number is incremented each time the timepoint is not calculated for `ITL4` (`.OPTION ITL4=val`; default is 10), i.e., when convergence criteria is not satisfied. After such an exception is detected, SmartSpice reduces the current requested timestep by a factor specified by `.OPTION COEF1=val` or `FT=val` (default is 0.25), and then will repeat the calculation of voltages and currents. If you see a large number of `CONrej`, specify `.OPTION ITL4=val` in the range 20 - 100. Larger values are not recommended due to bad convergence properties of the circuit, which might be casued by an incorrectly implemented device model.
DEVrej	Device exceptions number indicating the number of NaN issues produced by the model during model equation evalution.
MATrej	Exception is produced by the matrix solver and indicates a singularity issue. Topology must be checked for incorrect connections, models and devices for correct parameters. Usually happens during OP. `.OPTION DCGNODE=val` in the range 1e-12 ~ 1e-15 may fix singularity. Floating Nodes and Nodes with less than one connection may cause such an issue. Switch on the other solver (`.OPTION SOLVER=XMS`, `AMS` or `BCG`). Try associating with solver options (see Chapter 21 Using Solvers in SmartSpice).

LTErej	Exception is produced by local error truncation mechanism associated with a device (capacitor, transistor). Internally, SmartSpice calculates the next timestep based on the previous voltage and current derivatives (see `.OPTION LVLTIM=val`) and predicts the next timestep. The predicted timestep is checked on an acceptable minimum of integration waveform error (`.OPTION TRTOL=val` and `.OPTION RELTOL=val`) by truncation module in the device, and a new value for timestep is suggested. If the new value is smaller than predicted by the derivative analyzer, SmartSpice generates such an exception and reduces timestep. To reduce `LTErej`, `.OPTION TRTOL=val` must be increased and `.OPTION LTERATIO=val` (see Section 3.14“.OPTIONS (Option Specification)”). `.OPTION RELTOL=val` must be increased. Try to switching to the `.OPTION INTEGR` method.
DERrej	Exception is generated by the derivative waveform check mechanism in timestep control. Might be totally blocked in some timestep algorithms using `.OPTION RMAX=val`.
breaks	Informative number and is not related to exceptions produced by simulator. Indicates the number of predefined timepoints forced by SmartSpice. Related to accuracy and can be control with `.OPTION MINBREAK`.

32.2 Alternative Enchanced Statistics Produced by .OPTION ACCT

```
Total memory used: 7656 Kbytes

Circuit:      DEMO EXAMPLE 1: INPUT FILE
File: /main/alpha/examples/smartspice/ex1.in
Date: Mon Feb  7 10:59:16 2011

temp (Operating temperature) = 27
tnom (Nominal temperature) = 27
clim (threshold on the linear capacitor value) = 0
rlim (threshold on the linear resistor value) = 0
rmin_rc (Resistance threshold for RC reduction) = 0.001
cmin_rc (Capacitance threshold for RC reduction) = 1E-22
lmin_rc (Inductance threshold for RLC reduction) = 1E-09
accept (Accepted timepoints) = 3030006
equations (Circuit Equations) = 17
loadtime (Load time) = 63.6
lutime (L-U decomposition time) = 66.81
rejected (Rejected timepoints) = 0
reordertime (Matrix reordering time) = 0
solvetime (Matrix solve time) = 6.67
time (Total Analysis Time) = 194.2
totiter (Total iterations) = 6060030
traniter (Transient iterations) = 6060006
tranlutime (Transient L-U decomp time) = 66.81
tranpoints (Transient timepoints) = 3030006
transolvetime (Transient solve time) = 6.67
trantime (Transient time) = 194.2
convtime (Convergence check overhead) = 3.06

Circuit elements short summary:
polar Junction Transistor : 4
Independent voltage source : 3
Capacitor : 1
Resistor : 5
```

32.3 Detailed Statistics for Each Device and SmartSpice Module

Detailed statistics for each device and SmartSpice module is called by using `.OPTION ACCT=2` in the netlist. The Description column contains the correspondent module name: "SPICE", "Solver name_of_used_solver" or "Smartspice". If Verilog-A module is present in the circuit, Verilog-A compilation time will be added to the total analysis time.

```
-------------------------------------------------------------------------------------------------
Description                                    |  Time, s |      % | Histogram
-------------------------------------------------------------------------------------------------
Total Analyses                                 |    10.77 |        |
Load                    ( Models )             |     4.97 |  46.15 | ########################
Setup                   ( Models )             |     0.00 |   0.00 |
Temperature Setup       ( Models )             |     0.00 |   0.00 |
Truncation Control      ( Models )             |     0.00 |   0.00 |
Matrix Reordering       ( Solver XMS(default) ) |     0.00 |   0.00 |
L-U Decomposition       ( Solver XMS(default) ) |     0.01 |   0.09 | #
Matrix Solve            ( Solver XMS(default) ) |     0.00 |   0.00 |
Other Matrix operations ( Solver XMS(default) ) |     0.00 |   0.00 |
Convergence control     ( Smartspice )         |     0.03 |   0.28 | #
Time step control       ( Smartspice )         |     0.00 |   0.00 | #
Simulation Data I/O     ( Smartspice )         |     0.02 |   0.19 | #
Measurements            ( Smartspice )         |     0.00 |   0.00 |
Postprocessor           ( Smartspice )         |     0.00 |   0.00 |
Other operations        ( Smartspice )         |     0.00 |   0.00 |
Verilog-A compilation   ( Verilog-A )          |     5.74 |  53.30 | ##############################
-------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------
Device          | Version    |   Count | Total, s | Setup, s |  Temp, s |  Load, s |    %  | Histogram
-------------------------------------------------------------------------------------------------
Vsource         | 1.8.14     |       3 |     0.00 |     0.00 |     0.00 |     0.00 |  0.00 |
Capacitor       | 1.8.16     |      11 |     0.00 |     0.00 |     0.00 |     0.00 |  0.00 |
Resistor        | 1.8.17     |      11 |     0.00 |     0.00 |     0.00 |     0.00 |  0.00 |
VLG             | 1.8.71     |      22 |     4.95 |     0.00 |     0.00 |     4.95 |100.00 | #########
-------------------------------------------------------------------------------------------------
```

32.4 Runtime Statistics Displaying

System environment variable (on UNIX setenv command) `SMARTSPICE_RT_STATS` turns on the display statistics from Section 32.3“Detailed Statistics for Each Device and SmartSpice Module” during runtime. You may see an indication of major performance slowdown reported in the statistics before the analysis is complete. Use `.OPTION SOLVER_MONITOR` which helps to see solver progress indicator. During critical loops in the solver the SmartSpice GUI may become frozen; the `SOLVER_MONITOR` indicates how much of the job has been completed.

32.5 New SmartSpice Statistics

Circuit Temperature Information contains the values of parameters `tnom` and `temp` used during simulation.

Machine Information contains information about CPU, CPU's model name, CPU speed, OS and compiler version. It's supported on Linux only.

```
******  Machine Information  ******

CPU:
model name: Intel(R) Core(TM) i5 CPU         760  @ 2.80GHz
cpu MHz: 2798.367

OS:
Linux        version        2.6.18-194.el5        (mockbuild@x86-
005.build.bos.redhat.com)  (gcc  version  4.1.2  20080704  (Red  Hat
4.1.2-48)) #1 SMP Tue Mar 16 21:52:39 EDT 2010
```

SmartSpice Threads Information contains information about using of multithreading for model/solver evaluation.

SmartSpice prints the following threads information section:

```
****** SmartSpice Threads Information ******
Command Line Model Threads Count : val_mtc
Command Line Solver Threads Count : val_stc
Available CPU Count : val_acpu
Actual Model Evaluation(Load) Threads Count : val_amtc
Actual Solver Threads Count : val_astc
```

where:

val_mtc	Model threads count specified in command line. If the option -P is not specified SmartSpice will print out "Not specified" in the Command Line Model Threads Count line.
val_stc	Solver threads count specified in command line. If the option -PS is not specified SmartSpice will print out "Not specified" in Command Line Solver Threads Count line.
val_acpu	Availiable CPU count on the machine.
val_amtc	Actual model threads count. By default, SmartSpice will use val_acpu for devices. But if the netlist contains less than 150 devices of certain type SmartSpice sets val_amtc to 1.
val_astc	Actual solver threads count. By default, SmartSpice will use val_acpu for solver. But if the matrix contains less than 20K equations SmartSpice sets val_astc to 1.

Circuit Statistics contains information about devices.

```
******  Circuit Statistics  ******

# nodes       =  14 # elements  =   13
# resistors   =  5 # capacitors =   1 # inductors  =   0
# mutual_inds =  0 # vccs       =   0 # vcvs       =   0
# cccs        =  0 # ccvs       =   0 # volt_srcs   =   3
# curr_srcs   =  0 # diodes     =   0 # bjts       =   4
# jfets       =  0 # mosfets    =   0 # U elements =   0
# T elements  =  0 # W elements =   0 # B elements =   0
# S elements  =  0 # P elements =    0 # va device =   0
```

Runtime Statistics contains runtime information about different phases of simulations.

At the end of a simulation, SmartSpice prints out the following table:

```
****** Runtime Statistics (seconds) ******
analysis time # points tot. iter conv.iter
op point 0.00 1 8
dc sweep 0.17 63 200
transient 1.43 303 684 336 rev= 49
ac analysis 0.74 183
readin 2.39
errchk 0.00
setup 0.00
output 0.00
```

```
maximum memory used 137023 kbytes
total cpu time 4.73 seconds
total elapsed time 4.73 seconds
job started at 14:07:00 06/27/2011
     job ended at 14:07:05 06/27/2011
```

where:

analysis	Analysis phase or analysis name (like op point, dc sweep, transient, ac analysis), parser phase (readin), error check during parsing (errchk), device's setup (setup), output (preparing and outputting information during analysi).
time	Wall clock time spent in the submitted job (sum of op point + ... + readin + errchk + setup + output).
# points	For transient : tstop/tstep + 1.
tot. iter	Total number of iterations analysis performed completing simulation.
conv.iter	Counts only converged iterations.
rev	Reversal - LTE rejections count (transient only).
maximum memory used	Peaked memory usage for Windows and virtual image size for UNIX.
total cpu time/ total elapsed time	Wall clock time spent in the submitted job.
job started at/ job ended at	Wall clock start/end job's timestamps.

Solver Statistics describes solver runtime information:

```
******  Solver Statistics (seconds)  ******

matrix reordering              0.30
lu decomposition               3.28
matrix solve                   1.01
other matrix operations        0.01
```

where:

matrix reordering	Wall clock time spent while reordering matrix
lu decomposition	Wall clock time spent in LU-factorization
matrix solve	Wall clock time spent while solving linear matrix
other matrix operations	Wall clock time spent in other matrix operations like cleaning matrix

32.6 Output Statistic Differences with Other Simulators

- A node associated with a dangling resistor is not added to the node count.
- SmartSpice does not add a dangling resistor to a node with an initial condition specified.
- Non zero of "lint" (Bsim3v3 model parameter) may cause internal nodes for parasitic resistors in some ACM (model parameter) cases. This situation was implemented in SmartSpice `-hspice` mode only.
- SmartSpice counts every S-element defined in the netlist. In some situations there may be a difference in S-element count between SmartSpice and Hspice.
- SmartSpice will retain the capacitor count even if they are found to be connected to a floating node during the OP analysis.
- SmartSpice internally creates voltage sources if a `.VEC` file is included and these sources are included in the total number of sources in the circuit statistics. Other simulators only output the number of sources included in the circuit and do not include the sources required for `.VEC` functionality.

Chapter 33
Examples

33.1 Introduction

This chapter contains carefully selected examples to demonstrate the basic and advanced features of SmartSpice. These examples help explain how to run SmartSpice in batch mode. The first two examples, with detailed comments, can also serve as a short tutorial on how to use basic SmartSpice features. The following explains how you can create circuit descriptions, specify analyses, output, and control commands necessary for circuit simulation.

33.2 Example 1: Diff-Pair Circuit

Differential-pair circuit is shown in Figure 33-1.

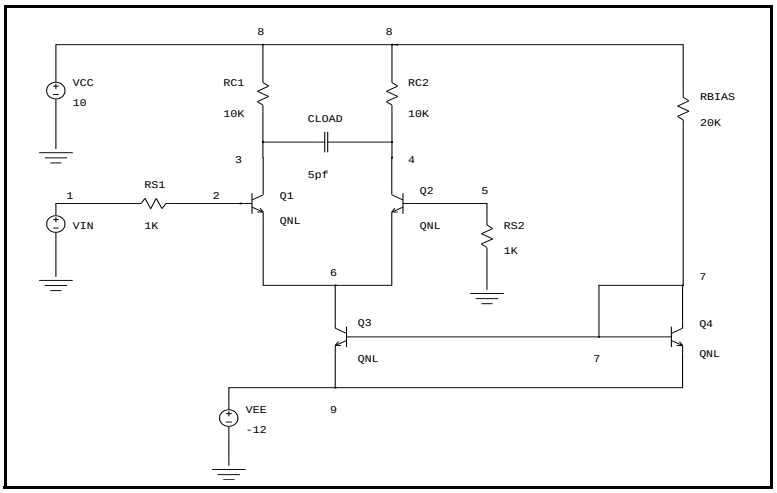


Figure 33-1 Differential-Pair Circuit

```
* DEMO EXAMPLE 1: INPUT FILE
*
VIN      1  0  DC   0    SIN(0 0.1 5MEG) AC 1
VCC      8  0  DC   10
VEE      9  0  DC   -12
RS1      1  2  1K
RS2      5  0  1K
RC1      3  8  10K
RC2      4  8  10K
RBIAS    7  8  20K
CLOAD    3  4  5PF
Q1       3  2  6    QNL
Q2       4  5  6    QNL
Q3       6  7  9    QNL
Q4       7  7  9    QNL
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS
+TR=6NS CJE=3PF CJC=2PF VA=50)
.TRAN 5NS 500NS
.DC VIN -0.25 0.25 0.025
.AC DEC 10 10KHZ 10GHZ
.PRINT TRAN V(3) V(4)
.PRINT DC V(1) V(3) V(4)
.PRINT AC VM(3) VP(3)
.ST QNL(BF) 80 100 10
.OPTIONS ACCT RELTOL=0.001
.END

* DEMO EXAMPLE 1: COMMENTS
*
```

This example illustrates several SmartSpice capabilities. It contains a circuit description and specifications for three basic types of analysis: Transient, DC, and AC analyses.

```
*
*......... PART I: CIRCUIT ..................
```

The following statements describe the circuit (see Section 33.2“Example 1: Diff-Pair Circuit”) and parameters of the bipolar transistor model.

```
VIN 1 0 DC 0 SIN(0 0.1 5MEG) AC 1
```

This is an input voltage source connected between nodes 1 and 0. It specifies a DC value of 0 V for any DC analysis phase, a sinusoidal waveform for any transient analysis phase, and an amplitude of 1V for AC analysis.

```
VCC 8 0 DC 10
VEE 9 0 DC -12
```

These are power supplies. The first one sets node 8 to a voltage value of 10V, the second sets node 9 to a voltage value of -12 V.

```
RS1 1 2 1K
```

This is a resistor. It is connected between nodes 1 and 2, and has resistance of 1 kilo ohms.

```
RS2    5  0  1K
RC1    3  8  10K
RC2    4  8  10K
RBIAS  7  8  20K
CLOAD  3  4  5PF
```

This capacitor has a capacitance of 5 picofarads.

```
Q1      3   2   6    QNL
```

This is a transistor. It refers to the model QNL. 3, 2 and 6 are collector, base and emitter nodes.

```
Q2      4   5   6    QNL
Q3      6   7   9    QNL
Q4      7   7   9    QNL
*
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS
+ CJE=3PF CJC=2PF + VA=50)
```

This is the definition of the Gummel-Poon bipolar transistor model. The keyword NPN indicates the type of the model. QNL is the name of the model definition for references in the circuit description. All transistors of the circuit must refer to corresponding models. This definition sets the model parameters BF, RB, CCS, TF, TR, CJE, CJC and VA. Some of the model parameters will use their default values.

```
*...PART II: ANALYSIS, OUTPUT and CONTROL STATEMENTS..
```

This tells SmartSpice what to do with the circuit above. It contains analysis, output, and control commands. When an input contains several analyses of the same type, SmartSpice will perform all of them. If an input file contains analysis statements of different types, then SmartSpice performs all of them, following the sequence of analyses defined by you. SmartSpice performs all of the output commands after a current analysis is finished, again following the sequence of commands that you defined.

```
.TRAN 5NS 500NS
```

This statement causes the transient analysis to be performed on the circuit. SmartSpice: Computes the DC operating point at the time 0NS. Computes the nonlinear circuit's behavior over the time interval from 0 to 500NS and saves the calculated results for printing plotting and measuring on the same interval. The maximum limit of the internal time step is 500NS / 50. During an entire transient simulation, all the voltage and current sources maintain their transient analysis values: VIN=SIN, VCC=DC=10V and VEE=DC=-12V. Transient analysis is the most frequently used and is the most universal analysis in SmartSpice.

```
.DC VIN -0.25 0.25 0.025
```

This computes a DC transfer curve for the circuit with capacitors opened and inductors shorted. SmartSpice does a DC analysis, sweeping the voltage source VIN from -0.25V to 0.25V in steps of 0.025V. During the DC transfer curve simulation, all the voltage and current sources maintain their DC analysis values: VCC=DC=10V and VEE=DC=-12V. The currently calculated values of the swept VIN source override its DC value of 0V.

```
.AC DEC 10 10KHZ 10GHZ
```

The .AC statement performs an AC linear small-signal analysis on the circuit. SmartSpice first creates a linearized small-signal model at the DC operating point of the circuit (VIN=DC=0, VCC=DC=10V and VEE=DC=-12V) and then computes the frequency sweep by 10 points per decade from 10KHZ to 10GHZ.

```
 .PRINT TRAN V(3) V(4)
```

This prints the output variables V(3) and V(4) computed in the transient analysis.

```
.PLOT DC V(1) V(3) V(4)
```

This plots the output variables V(1), V(3) and V(4) computed in the DC analysis.

```
.PRINT AC VM(3) VP(3)
```

This prints the magnitude and phase of the node voltage V(3) computed in the AC analysis.

```
.ST QNL(BF) 80 100 10
```

This causes SmartSpice to perform multiple runs. It sweeps parameter BF of the model QNL from 80 to 100 in increments of 10. All of the analyses and output commands are done for each step.

```
.OPTIONS ACCT RELTOL=0.001
```

This statement sets the options for runs.

The following statement is the last in the input file.

```
.END
```

33.2.1 Example 1 Output

Transient analysis output signals are shown in Figure 33-2. DC output signals are shown in Figure 33-3. AC output signals are shown in Figure 33-4.

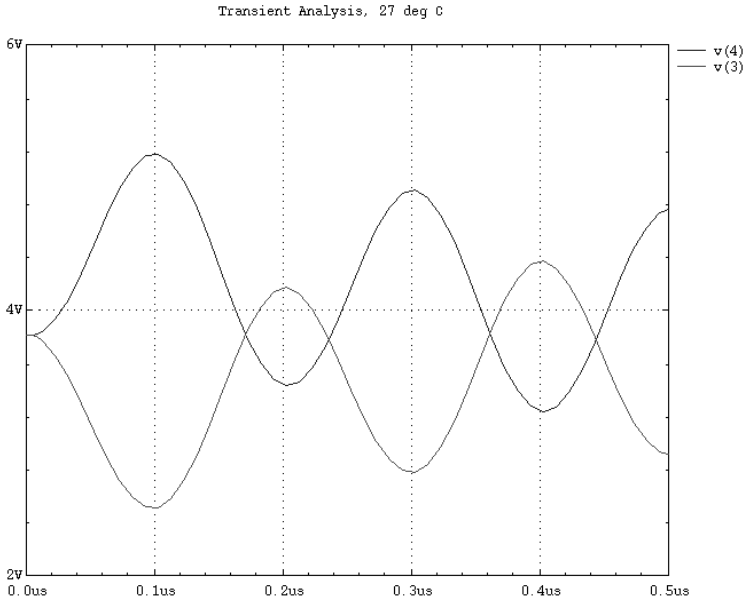


Figure 33-2 Transient Analysis Output Signals

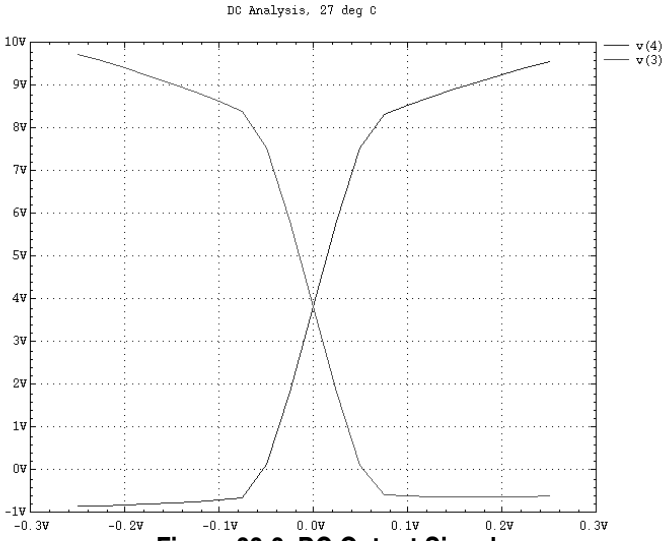


Figure 33-3 DC Output Signals

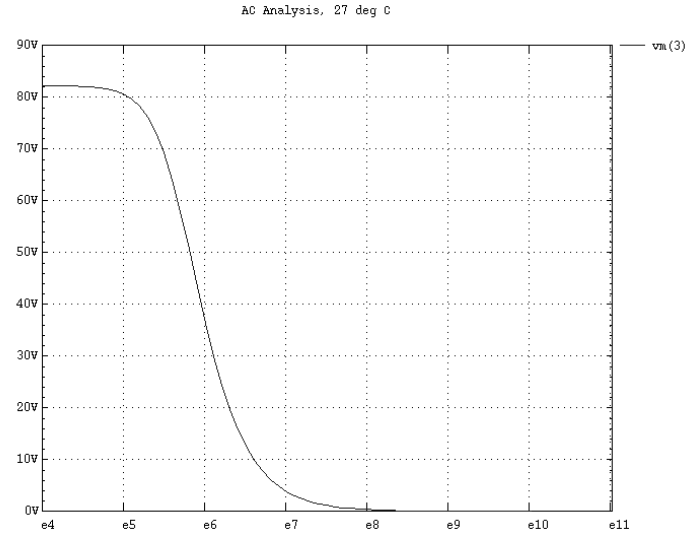


Figure 33-4 AC Output Signals

33.3 Example 2: Subcircuits

Subcircuit schematic examples are shown in Figure 33-5 and Figure 33-6.

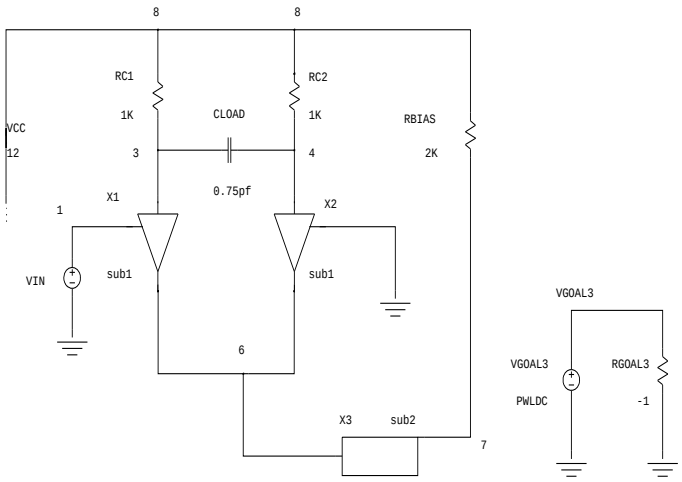


Figure 33-5 Circuit with Subcircuits

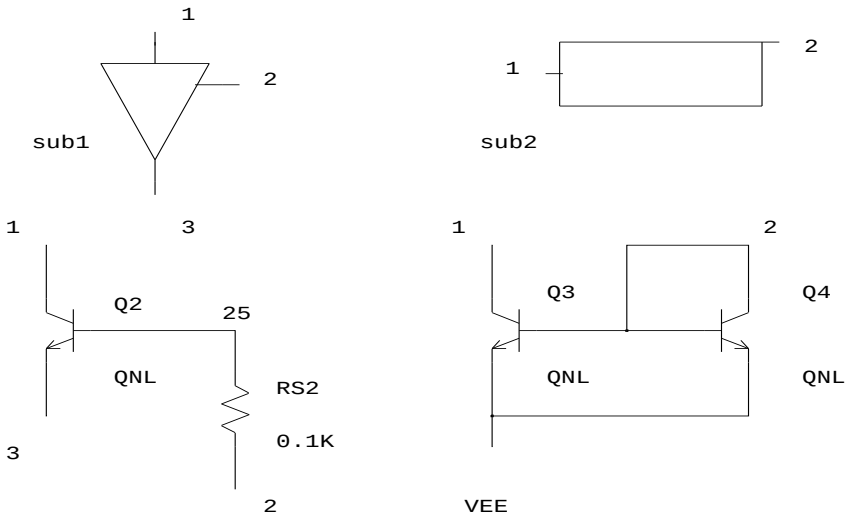


Figure 33-6 Circuit with Subcircuits-(part2)

Subcircuits

```
* DEMO EXAMPLE 2: INPUT FILE
*
* .SUBCKT SUB1 1 2 3
Q2   1  25  3     QNL
RS2  25 2   0.1K
.ENDS
.SUBCKT SUB2 1 2
Q3   1  2   VEE   QNL
Q4   2  2   VEE   QNL  1
.ENDS
.GLOBAL VEE
VIN 1 0 DC -0.25 PULSE(-0.25 0.25 1N 2N 2N 18N 40N)
+ AC 1
VEE   VEE  0  DC    -12
VCC   8    0  DC    12
X1    3    1  6     SUB1
CLOAD 3    4  0.75  PF
RC1   3    8  RMOD  1K
RBIAS 7    8  2K
X2    4    0  6     SUB1
X3    6    7  SUB2
RC2   4    8  RMOD  1K
VGOAL3 VGOAL3 0 PWLDC(-0.25V 12 -0.2V 11.97 -0.15V 11.9
+ -0.1V 11 +-0.05V 9.2 0V 5.6 0.05V 2.1 0.1V 0.8 0.15V
+ 0.4 2V 0.31 2.5V 0.3)
RGOAL3  VGOAL3   0   -1
.MODEL QNL NPN(BF=80 RB=100 CCS=0.2PF TF=0.03NS
+ TR=0.6NS CJE=0.3PF CJC=0.2PF)
.MODEL RMOD R (TC1=0.003)
.TRAN 5NS 80NS 1NS 1.5NS CALLV SAVEV
.MEASURE TRAN MAX_TR_V3 MAX V(3)
.MEASURE TRAN MIN_TR_V3 MIN V(3)
.MEASURE TRAN FALL_1_TR_V3 WAVE V(3)
+ FALL=1 VAL0=MIN_TR_V3 VAL1=MAX_TR_V3
.MEASURE TRAN DEL_V3_V1 DELAY V(1) RISE=1 VAL=0
+ TARG=V(3) FALL=1 VAL0=MIN_TR_V3 VAL1=MAX_TR_V3
.MEASURE TRAN CR_V3_V4_3 CROSS V(3) V(4) OCCUR=3
.DC VIN -0.25 0.25 0.05 CALLV
.SNS DC V(3) V(X1.25) V(Q.X1.Q2#B)I(VIN) I(VCC)
+ I(VEE) V(4)
+ TO VIN(DC) RBIAS(RES) R.X1.RS2(RES) QNL(RB) TEMP GMIN
+ ARG=-0.25V
.MEASURE DC ERR_V3_VGOAL3 ERR V(VGOAL3) V(3)
.AC DEC 10 100KHZ 10GHZ CALLV UIC
.MEASURE AC MAX_VM3 MAX VM(3)
.MEASURE AC MIN_VM3 MIN VM(3) TO=1MHZ
.MEASURE AC RISE_1_VM3 WAVE VM(3)
+ COEF=0.01 RISE=1 VAL0=MIN_VM3 VAL1=MAX_VM3
.MODIF RBIAS(RES)=2.5K CLOAD(CAP)=0.75PF
+MODIF LOOP=5 STOP DEL_V3_V1 LE 2NS RBIAS(RES)*=0.9
+CLOAD(CAP)*=0.6
.OPTIONS RELTOL=0.01 ABSTOL=1N VNTOL=50U ITL1=100
+ ITL2=50 ITL4=10 LIST_ALL_CURRENTS
.END
```

```
* DEMO EXAMPLE2: COMMENTS
*
```

This example illustrates several advanced features of SmartSpice: parameter modifications with stop conditions; sensitivity analysis; calculation of circuit performance measures; operating point saving and calling; subcircuits and global nodes. The circuit in Section 33.2“Example 1: Diff-Pair Circuit” and Section 33.3“Example 2: Subcircuits” are essentially the same except that some parameter values in the circuit in Subcircuits have changed.

```
*
*......... PART I: COMPONENTS and CIRCUITS .........
```

The following statements describe circuits and their components: subcircuits and models.

```
*............... Subcircuits ....................
.SUBCKT SUB1 1 2 3
Q2 1 25 3 QNL
RS2 25 2 0.1K
.ENDS
```

This is the definition of the subcircuit shown in Figure 33-6. SUB1 is name of the subcircuit for references in circuit description. The circuit contains two such subcircuits. All subcircuits called from the circuit must refer to corresponding subcircuit definitions. 1, 2 and 3 are the names of the subcircuit terminals. 25 is the name of an internal subcircuit node. The subcircuit contains two elements. The first is transistor Q2. It refers to the model QNL. 1, 25 and 3 are the collector, base and emitter nodes. The second element is resistor RS2. It is connected between node 25 and node 2, and has a resistance value of 100 ohms. .ENDS specifies the end of the subcircuit definition.

```
*
.SUBCKT SUB2 1 2
Q3 1 2 VEE QNL
Q4 2 2 VEE QNL 1
.ENDS
```

This is the second subcircuit definition. The subcircuit contains two transistors. The model name QNL of Q4 is followed by the parameter area =1. The node VEE is reserved for a power supply.

```
*
.GLOBAL VEE
```

This statement defines the global node VEE. After this the node VEE is directly available. The .GLOBAL statement allows all nodes specified to overwrite the local subcircuit definition.

```
*
*................. Circuits ..................
*
VIN 1 0 DC -0.25 PULSE(-0.25 0.25 1N 2N 2N 18N 40N) AC 1
```

This is the input voltage source. It specifies a DC value of -0.25V for any DC analysis phase, a pulse waveform for transient analysis, and an amplitude of 1V for AC analysis.

```
*
VEE VEE 0 DC -12
VCC 8 0 DC 12
```

These are power supplies. The first one sets the global node VEE to a voltage value of -12V.

```
*
X1 3 1 6 SUB1
```

This is a pseudo-element. It calls the subcircuit definition SUB1. 3, 1 and 6 are the actual circuit nodes. SmartSpice automatically inserts to this place the elements Q2 and RS2 of the subcircuit SUB1, and replaces node names 1, 2 and 3 by actual circuit node names 3, 1 and 6, respectively.

```
*
CLOAD 3 4 0.75PF
```

This is a capacitor. It is connected between nodes 3 and 4, and has a capacitance of 0.75 picofarads.

```
RC1 3 8 RMOD 1K
```

This is a resistor. It refers to the model definition RMOD and has resistance 1 kilo ohms.

```
*
RBIAS  7  8  2K
```

This line specifies the resistor RBIAS without a model.

```
*
X2  4 0  6 SUB1
X3  6 7  SUB2
RC2 4 8  RMOD 1K
*
.....................................................
VGOAL3 VGOAL3 0 PWLDC(-0.25V 12 -0.2V 11.97 -0.15V 11.9
+ -0.1V 11 + -0.05V 9.2 0V 5.6 0.05V 2.1 0.1V 0.8 0.15V
+ 0.4 2V 0.31 2.5V 0.3)
RGOAL3 VGOAL3 0 -1
```

This small circuit shown in Figure 33-5 is included in the input file in order to compare the results desired (measured) with the actual results of the DC simulation for the node voltage V(3). The voltage V(VGOAL3) and the current through VGOAL3 are equal. Both of them can be printed, plotted, and measured.

```
.....................................................
.MODEL QNL NPN(BF=80 RB=100 CCS=0.2PF TF=0.03NS
+ TR=0.6NS CJE=0.3PF CJC=0.2PF)
```

This is the definition of the bipolar transistor model.

```
*
.MODEL RMOD R (TC1=0.003)
```

This is an optional model for a resistor. It specifies the first order temperature coefficient TC1. RMOD is the name of the model. The keyword R indicates the type of the model.

```
*
*...PART II: ANALYSIS, OUTPUT and CONTROL STATEMENTS...
```

This part tells SmartSpice what to do with the circuit above. It contains analysis, output and control commands. When an input contains several analysis commands, SmartSpice will perform all of them, following the sequence of analyses defined by the user. SmartSpice can save the results of previous analysis to use them in the next analysis. SmartSpice performs all of the output commands after a current analysis is finished, again following the sequence of

commands defined by the user. The .MEASURE command is used to calculate a few numbers, such as propagation delay, rise time, peak-to-peak difference etc., which can characterize circuit performances in general. SmartSpice saves the results of previous measurement to use them for the next measurement. The .PRINT and .PLOT statements can produce large amounts of output. The .MEASURE statements allow you to decrease the amount of output data. For this reason, none of the .PRINT and .PLOT commands were included in present input file.

```
*
*.....................................................
.TRAN 5NS 80NS 1NS 1.5NS CALLV SAVEV
```

This statement causes transient analysis to be performed on the circuit. SmartSpice calls a previously saved operating point; recomputes the DC operating point at the time 0NS; saves the found operating point; computes a nonlinear circuit's behavior over the time interval from 0 to 80ns; and saves calculated results for printing, plotting, and measuring on the interval from the time 1NS to 80NS. The maximum internal time step is 1.5NS. During an entire transient simulation, all the voltage and current sources maintain their transient analysis values:

```
* VIN=PULSE, VEE=DC=-12V,VCC=DC=12V,VGOAL3=DC=0.
*
.MEASURE TRAN MAX_TR_V3 MAX V(3)
```

This statement causes SmartSpice to find the maximum voltage of the node 3 during the transient analysis. The result is stored under the name MAX_TR_V3. By default, calculations are performed for the entire transient analysis interval.

```
*
.MEASURE TRAN MIN_TR_V3 MIN V(3)
```

This is for minimum value of the V(3) calculation. The result is stored under the name MIN_TR_V3.

```
*
.MEASURE TRAN FALL_1_TR_V3 WAVE V(3)
+ FALL=1 VAL0=MIN_TR_V3 VAL1=MAX_TR_V3
```

This calculates the time of the first fall (transfer from voltage value VAL1 to VAL0) for the output waveform V(3). There are references to the results of the measurements previously calculated (MIN_TR_V3,MAX_TR_V3) to get VAL0,VAL1.

```
*
.MEASURE TRAN DEL_V3_V1 DELAY V(1) RISE=1 VAL=0
+ TARG=V(3) FALL=1 VAL0=MIN_TR_V3 VAL1=MAX_TR_V3
```

This computes the propagation delay between the middle point of the first transfer from VAL1 to VAL0 of V(3) and the point where the first rising voltage of node 1 reaches 0V. This measurement is stored under the name DEL_V3_V1, and has been included in the stop condition of the control command .MODIF.

```
*
.MEASURE TRAN CR_V3_V4_3 CROSS V(3) V(4) OCCUR=3
```

This is for calculation of the point of the third intersection of V(3) and V(4).

```
.DC VIN -0.25 0.25 0.05 CALLV
```

This computes a DC transfer curve for the circuit with capacitors opened and inductors shorted. SmartSpice calls the DC operating point calculated and saved in the transient

analysis; and performs DC analysis, sweeping the voltage source VIN from -0.25V to 0.25V in steps of 0.05V. During the DC transfer curve simulation, all the voltage and current sources maintain their DC analysis values: VEE=DC=-12V, VCC=DC=12V, VGOAL3=PWLDC. The currently calculated values of the swept VIN source override its DC value of -0.25V.

```
*
.SNS DC V(3) V(X1.25) V(Q.X1.Q2#B) I(VIN) I(VCC) I (VEE)V(4)
+ TO VIN(DC) RBIAS(RES) R.X1.RS2(RES) QNL(RB) TEMP GMIN
+ ARG=-0.25V
```

This computes sensitivities of the node voltages V(3), V(X1.25), V(Q.X1.Q2#B) and of the current I(VIN) with respect to device parameters VIN(DC), RBIAS(RES), R.X1.RS2(RES), to model parameter QNL(RB), and to the global parameters TEMP and GMIN at a DC sweep argument value of -0.25V. SmartSpice computes sensitivities of only the basic variables of the circuit. These are: all node voltages, and currents through voltage sources of all types, and through inductors.

```
*
.MEASURE DC ERR_V3_VGOAL3 ERR V(VGOAL3) V(3)
```

This computes the difference between the node voltages V(3) and the measured node voltage V(VGOAL3).

```
*
.AC DEC 10 100KHZ 10GHZ CALLV UIC
```

The .AC statement performs an AC linear small-signal analysis on the circuit. SmartSpice calls the DC operating point calculated and saved in the transient analysis, creates a linearized small-signal model around the called operating point, and computes the frequency sweep by 10 points per decade from 100KHZ to 10GHZ.

```
*
.MEASURE AC MAX_VM3 MAX VM(3)
.MEASURE AC MIN_VM3 MIN VM(3) TO=1MHZ
```

In this statement, SmartSpice finds the minimum magnitude of the node 3 during the AC analysis. The result is stored under the name MIN_VM3. Calculations are performed until the argument is less than 1MHZ.

```
*
.MEASURE AC RISE_1_VM3 WAVE VM(3)
+ COEF=0.01 RISE=1 VAL0=MIN_VM3 VAL1=MAX_VM3
```

Real levels for rise calculation are VAL0+0.01*(VAL1-VAL0) and VAL1-0.01*(VAL1-VAL0).

```
.MODIF RBIAS(RES)=2.5K CLOAD(CAP)=0.75PF
+MODIF LOOP=5 STOP DEL_V3_V1 LE 2NS RBIAS(RES)*=0.9
+CLOAD(CAP)*=0.6
```

This causes SmartSpice to perform multiple runs. SmartSpice sets the parameter RES of the resistor RBIAS to a value of 2K, and the parameter CAP of the capacitor CLOAD to a value of 0.75PF. Performs all analysis commands of the input file. Performs five iterations specified in the second parameter set of the statement. On each iteration it multiplies the parameter RES by the coefficient 0.9, and the parameter CAP by the coefficient 0.6, and then performs all analysis commands. SmartSpice will terminate the parameter modification process when the condition DEL_V3_V1 <= 2NS is satisfied.

```
*
.OPTIONS RELTOL=0.01 ABSTOL=1N VNTOL=50U ITL1=100
```

```
+ ITL2=50 ITL4=10 LIST_ALL_CURRENTS ACCT
```

The .OPTIONS statement allows you to reset one or more SmartSpice control and computational parameters. The default values of these parameters specify the standard technologies SmartSpice will use. The first line of the statement shows the default values of some parameters. For certain simulation requirements, you can alter these technologies by changing the parameters that define them. If a parameter is changed for an analysis, that parameter will remain changed for every analysis in the input file. The keyword LIST_ALL_CURRENTS causes SmartSpice to save the currents derived from the basic variables. These are the currents through resistors, capacitors, diodes, transistors, and so forth. In batch mode, the default value of the LIST_ALL_CURRENTS flag option is OFF.

```
*
*
.END
```

33.3.1 Example 2: Output

Graphic examples of transient, DC, and AC output signals are shown in Figure 33-7, Figure 33-8, and Figure 33-9, respectively.

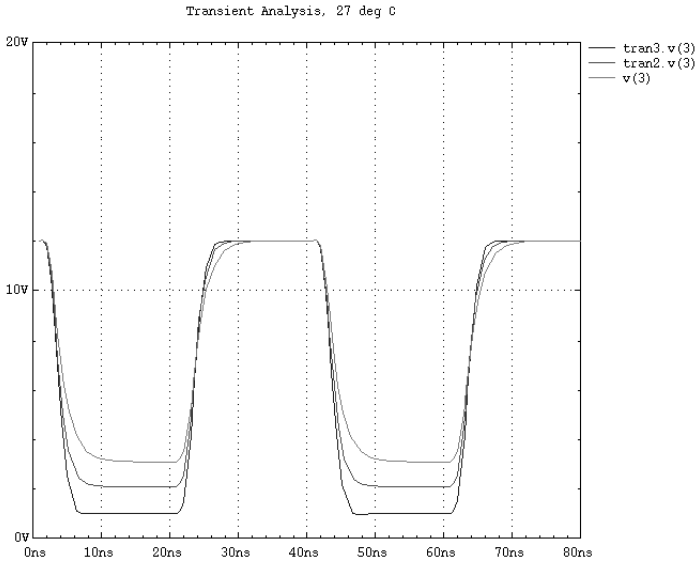


Figure 33-7 Transient Analysis Output Signals

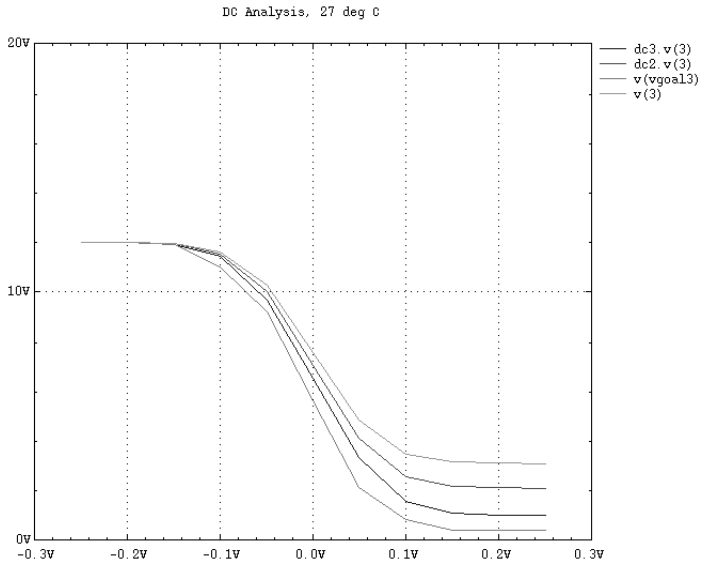


Figure 33-8 DC Output Signals

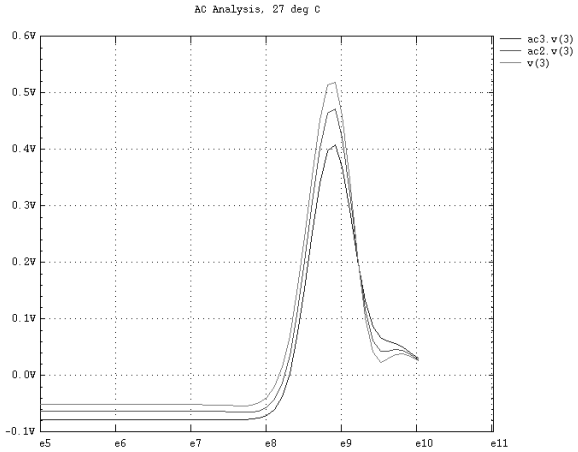


Figure 33-9 AC Output Signal

The results of the third (final) `.MODIF` iteration are shown below (sensitivities of some output variables are omitted).

```
MODIF : PARAMETER SET #2 STEP #2
MODIFIED PARAMETER VALUES :
rbias(res) = 2.025e+03 cload(cap) = 2.700e-13

measure max_tr_v3 max OUT = v(3)
Ymax = 1.202e+01
Xmax = 1.448e-09
measure min_tr_v3 min OUT = v(3)
Ymin = 9.436e-01
Xmin = 4.780e-08
measure fall_1_tr_v3 wave OUT = v(3)
FALL DURATION(ARG2 - ARG1) = 3.166e-09
ARG2 = 5.482e-09 ARG1 = 2.316e-09 FALL #1
measure del_v3_v1 delay INP = v(1) OUT(TARG) = v(3)
DELAY(OUT - INP) = 1.850e-09
ARGout = 3.850e-09 ARGinp = 2.000e-09
measure cr_v3_v4_3 cross INP = v(3) OUT = v(4)
```

```
CROSS POINT = 4.270e-08
CROSS VALUE = 8.282e+00 OCCUR #3
DC sensitivity of outvar v(3) = 11.998 at arg = -0.25

                parameter   sensitivity  norm.sens
                value       (out/par)
(out/(1%*par))
to dc(vin)      -2.500e-01  -7.785e-02   1.946e-04

Max Normalized  parameter   sensitivity  norm.sens
Sensitivity     value       (out/par)
(out/(1%*par))
to dc(vin)      -2.500e-01  -7.785e-02   1.946e-04
Worst Case
The worst case is                        4.406e-04

DC sensitivity of outvar v(4) = 0.986702 at arg = -0.25

                 parameter   sensitivity  norm.sens
                 value       (out/par)
(out/(1%*par))
to dc(vin)       -2.500e-01  7.785e-02    -1.946e-04
to res(rbias)    2.025e+03   5.423e-03    1.098e-01
to res(r.x1.rs2) 1.000e+02   0.000e+00    0.000e+00
to rb(qnl)       1.000e+0    7.696e-05    7.696e-05
to temp          3.001e+02   -3.354e-02   -1.007e-01
to gmin          1.000e-12   0.000e+00    0.000e+00

Max Sensitivity  parameter   sensitivity  norm.sens
                 value       (out/par)
(out/(1%*par))
to dc(vin)       -2.500e-01  7.785e-02    -1.946e-04

Max Normalized   parameter   sensitivity  norm.sens
Sensitivity      value       (out/par)
(out/(1%*par))
to res(rbias)    2.025e+03   5.423e-03    1.098e-01

Worst Case
The worst case is  2.107e-01
measure err_v3_vgoal3 err IN = v(vgoal3) OUT = v(3)
  ERR(IN - OUT) = 8.870e-01
  # POINTS = 11
measure max_vm3 max OUT = vm(3)
  Ymax = 4.102e-01
  Xmax = 6.310e+08
measure min_vm3 min OUT = vm(3)
  Ymin = 7.823e-02
  Xmin = 1.000e+05 (start of measure interval)
measure rise_1_vm3 wave OUT = vm(3)
  RISE DURATION(ARG2 - ARG1) = 5.824e+08
  ARG2 = 6.058e+08 ARG1 = 2.336e+07 RISE #1
MODIF : STOP CONDITION HAS BEEN SATISFIED
        FOR PARAMETER SET #2 STEP #2
MODIF : del_v3_v1 = 1.850e-09
```

33.4 Example 3: Typical Input Deck

The following input deck contains some of the most frequently used SmartSpice statements. Parameter labels are used to define device and model parameter values.

```
* EXAMPLE 3 :  Typical Input Deck

This example contains some of the most commonly used
structures and statements.
************** Optional Parameters
.OPTIONS NOMOD
.OPTIONS SCALE=1u
.OPTIONS RELTOL=0.002
************** Parameter Labels
.PARAM vd=5 rise=2n fall=2n
.PARAM pw=10 nw=4  length=1.2 pl=1.2 cap= 1pf
.PARAM  tox = 2.400E-8  delta = 0.2e-8
.PARAM  vto0 = -0.9 deltavto=0.04 vtop='vto0+deltavto'
************** Global Net Declarations
.GLOBAL GND vdd
************** Circuit  Netlist
Vcc vdd GND  'vd'
Vin inp GND  DC 0 PULSE(0 vd 0ns rise fall 10ns 24ns) AC 1
Mn mid inp GND GND nmod W='nw' L='length'
Mp mid inp vdd vdd pmod W=pw L=length
X1  mid out  inv  nl=1.2 pl=1.4 vbb=0.0
************** Subcircuit  Definition
.SUBCKT inv  input output  nl=1.3  pl=1.3 vbb=-1
M1 output input GND vbias  nmod W=nw L=nl
M2 output input vdd vdd  pmod W=pw L=pl
Cload  output 0  'cap'
Vb  vbias  0 dc  'vbb'
.ENDS inv
************** Analysis Statements
.TRAN  0.1n 24n
.AC DEC 10 1 1G
************** Output Statements
.SAVE v(out)
.LET dvout=deriv(v(out))
.MEAS TRAN min_dvout  MIN 'dvout'
.MEAS TRAN min_vout   MIN 'v(out)'
.MEAS TRAN max_vout   MAX 'v(out)'
.MEAS TRAN range_vout EXPR VAL='max_vout - min_vout'
*****************************************************
Model Definitions
*
Fast-Fast Transistor Model
*
.MODEL nmod     NMOS (TOX= 'tox+delta'
+ BEX=-1.5                          LD=1.850E-7
+ WD=0.000E-7     NSUB=3.000E16     VTO=0.720
+ UO=630.00       DELTA=0.000       VMAX=1.950E5
+ XJ=3.0E-7       KAPPA=0.100       ETA=0.026
+ THETA=0.060     TPG=1             NFS=1.20E11
+ CJ=420E-6       CJSW=271E-12      PB=0.940
+ MJ=0.430        MJSW=0.160        CGDO=3.2E-10
```

```
+ CGSO=3.2E-10    CGBO=0
+ RSH=35.0        CAPMOD=2          LEVEL=3     )
*
.model pmod     PMOS (TOX= 'tox+delta'
+ BEX=-1.5                          LD=2.70E-7
+ WD=-0.500E-7    NSUB=2.700E16     VTO= 'vtop'
+ UO=190.00       DELTA=0.000       VMAX=4.600E5
+ XJ=4.0E-7       KAPPA=0.100       ETA=0.042
+ THETA=0.100     TPG=1             NFS=0.50E11
+ CJ=392E-6       CJSW=373E-12      PB=0.880
+ MJ=0.460        MJSW=0.210        CGDO=3.2E-10
+ CGSO=3.2E-10    CGBO=0
+ RSH=60.0        LEVEL=3           )
.END
```

33.4.1 Example 3: Output

```
min_dvout    =  -1.3380e+09 at=   1.3848e-08
min_vout     =  -1.8757e-03 at=   9.0720e-10
max_vout     =   5.0043e+00 at=   1.2900e-08
range_vout   =   5.0062e+00
```

33.5 Example 4: RCA 3040 Wideband Amplifier

The schematic of the RCA 3040 wide band amplifier is shown in Figure 33-10.

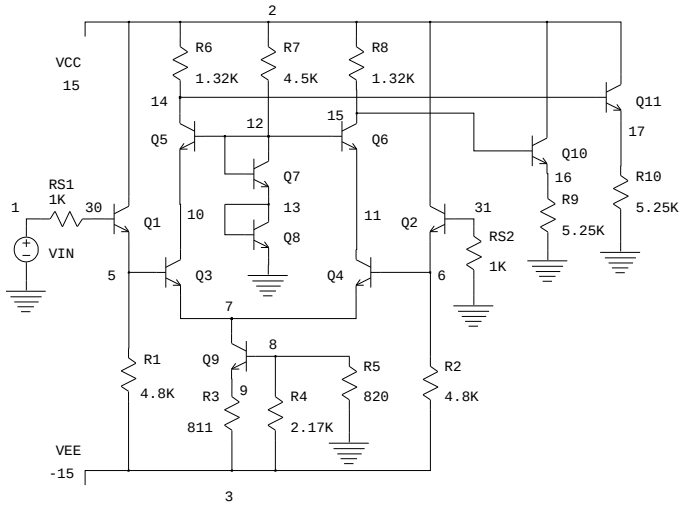


Figure 33-10 RCA 3040 Wideband Amplifier

```
* EXAMPLE4: RCA3040 WIDEBAND AMPLIFIER
*
* DC, AC and NOISE STATEMENTS
*
```

VIN	1	0	DC	0	SIN(0 0.1 50MEG 0.5NS) AC 1
VEE	3	0	-15		
VCC	2	0	15		
RS2	31	0	1K		
RS1	30	1	1K		
Q1	2	30	5	QNL	
Q2	2	31	6	QNL	
Q3	10	5	7	QNL	
Q4	11	6	7	QNL	
Q5	14	12	10	QNL	
Q6	15	12	11	QNL	
Q7	12	12	13	QNL	
Q8	13	13	0	QNL	
Q9	7	8	9	QNL	
Q10	2	15	16	QNL	
Q11	2	14	17	QNL	
R1	5	3	4.8K		
R2	6	3	4.8K		
R3	9	3	811		
R4	8	3	2.17K		
R5	8	0	820		
R6	2	14	1.32K		
R7	2	12	4.5K		

R8	2	15	1.32K
R9	16	0	5.25K
R10	17	0	5.25K

```
.MODEL QNL NPN(BF= 80 RB = 100 CCS=2PF TF= 0.3NS AF=1.0
+ KF=5.4E-16 TR=6NS CJE =3PF CJC =2PF VA = 50)
*
.DC VIN -0.25 0.25 0.005
.AC DEC 10 1 10GHZ
.NOISE V(17) VIN DEC 20 20 1K 1
*
.PRINT DC V(16) V(17)
.PRINT AC VR(16) VR(17)
.PRINT NOISES ONOISE_S ONOISE_S.Q1 ONOISE_S.Q1.1OVERF
.OPTIONS ACCT RELTOL=0.001
.END
```

33.5.1 Example 4: Output

Example 4, DC, AC, and noise analysis output signals are shown graphically in Figure 33-11, Figure 33-12, and Figure 33-13, respectively.

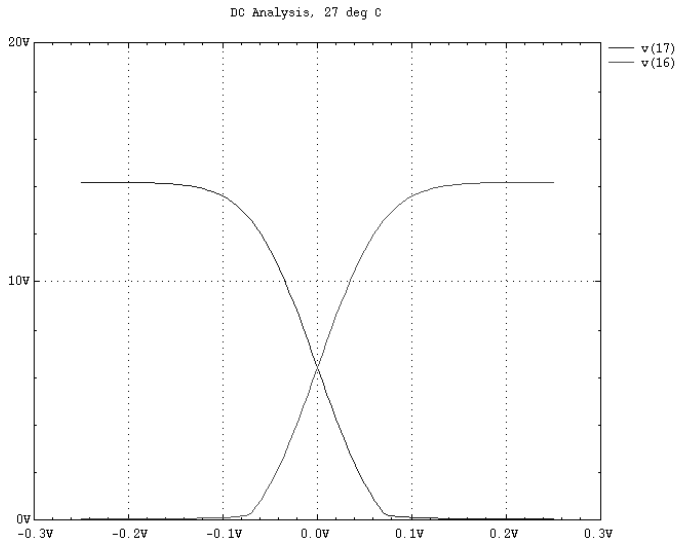


Figure 33-11 DC Output Signals

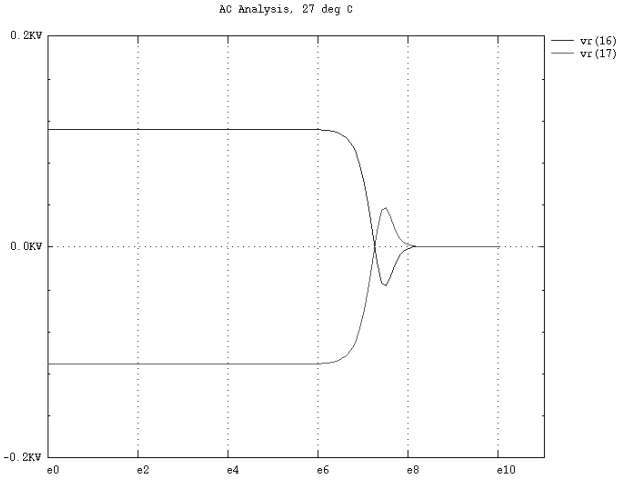


Figure 33-12 AC Output Signals

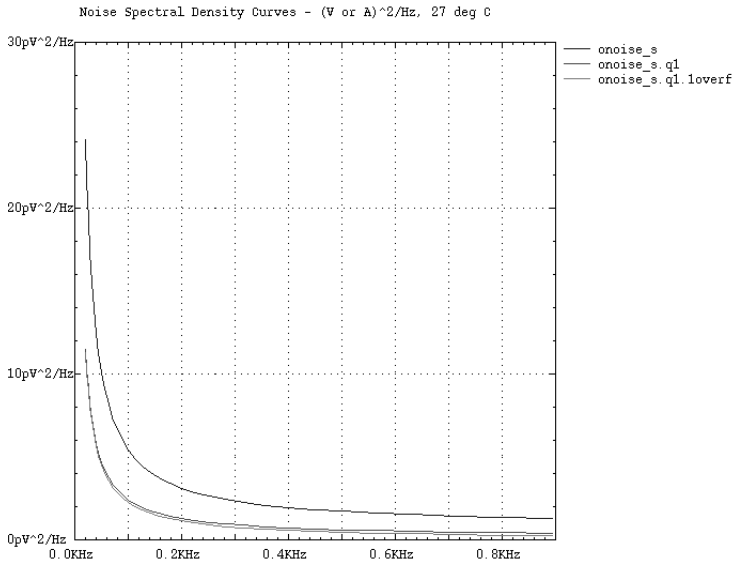


Figure 33-13 Noise Analysis Output Signals

33.6 Example 5: 32- Input Nand Gate

A 32-input Nand Gate schematic is shown in Figure 33-14.

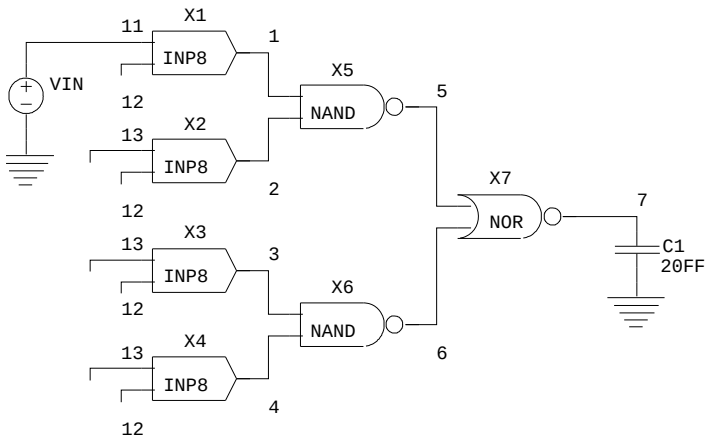


Figure 33-14 32 Input Nand Gate

```
* EXAMPLE 5 : 32-INPUT NAND GATE (132 MOSFETS)
*
SmartSpice INVESTIGATES THE DEPENDENCE OF THE
PROPAGATION DELAY OF THE CIRCUIT OUTPUT SIGNALS ON
TWO MOSFET MODEL PARAMETERS
*
.SUBCKT NAND 1 2 3
```

M1	**3**	**1**	**VDD**	**VDD**	**MODP**	**W=10UM**	**L=3U**	**AS=100P**	**AD=100P**
M2	3	2	VDD	VDD	MODP	W=10UM	L=3U	AS=100P	AD=100P
M3	3	1	6	0	MODN	W=10UM	L=3U	AS=100P	AD=100P
M4	6	2	0	0	MODN	W=10UM	L=3U	AS=100P	AD=100P

```
.ENDS NAND
.SUBCKT NOR 1 2 3
```

M1	6	1	VDD1	VDD1	MODP	W=10UM	L=3U	AS=100P	AD=100P
M2	3	2	6	VDD1	MODP	W=10UM	L=3U	AS=100P	AD=100P
M3	3	1	0	0	MODN	W=10UM	L=3U	AS=100P	AD=100P
M4	3	2	0	0	MODN	W=10UM	L=3U	AS=100P	AD=100P

```
.ENDS NOR
.SUBCKT INV 1 2
```

M1	**2**	**1**	**VDD2**	**VDD2**	**MODP**	**W=10UM**	**L=3U**	**AS=100P**	**AD=100P**
M2	2	1	0	0	MODN	W=10UM	L=3U	AS=100P	AD=100P

```
.ENDS INV
.SUBCKT INP8 11 12 10
```

X1	11	12	3	NAND
X2	12	12	4	NAND
X3	12	12	5	NAND
X4	12	12	6	NAND
X5	3	4	7	NOR

```
X6    5    6    8        NOR
X7    7    8    9        NAND
X8    9    10            INV
.ENDS 8-INP
.GLOBAL VDD VDD1 VDD2
*......... CIRCUIT .................................
VDD     VDD     0       DC          5.0V
VDD1    VDD1    0       DC          5.0V
VDD2    VDD2    0       DC          5.0V
V1      11      0       PULSE       (0 5 2NS 1NS 1NS 20NS 40NS)
V2      12      0       PULSE       (0 5 2NS 1NS 1NS 60NS 120NS)
V3      13      0       DC          5.V
X1      11      12      1           INP8
X2      13      12      2           INP8
X3      13      12      3           INP8
X4      13      12      4           INP8
X5      1       2       5           NAND
X6      3       4       6           NAND
X7      5       6       7           NOR
CL      7       0       20.         FF
.MODEL MODP PMOS (LEVEL=1 VTO=-1.0V UO=275 CBD=40.0FF)
.MODEL MODN NMOS (LEVEL=1 VTO=1.0V UO=550 CBD=40.0FF)
*...... TRAN, MEASURE and MODIF STATEMENTS.............
.MODIF
+MODIF LOOP=5 STOP DEL_V7_RISE LE 2NS MODP(CBD)*=0.5
+MODN(CBD)*=0.5
.TRAN 1NS 80NS CALLV SAVEV
.MEASURE TRAN DEL_V7_RISE DELAY V(11) VAL=2.5 RISE=2
+ TARG=V(7) VAL=2.5 RISE=2
.MEASURE TRAN DEL_V1_RISE DELAY V(11) VAL=2.5 RISE=2
+ TARG=V(1) VAL=2.5 RISE=2
.MEASURE TRAN DEL_V7_FALL DELAY V(11) VAL=2.5 FALL=2
+ TARG=V(7) VAL=2.5 FALL=2
.MEASURE TRAN DEL_V1_FALL DELAY V(11) VAL=2.5 FALL=2
+ TARG=V(1) VAL=2.5 FALL=2
.OPTION ACCT NOMOD
.END
```

SmartSpice performs six or fewer iterations. On the first iteration, it performs the transient analysis without changing parameters. On the following iterations, SmartSpice multiplies the parameters `CBD` of the models `MODP` and `MODN` by the coefficient `0.5`. It will terminate the parameter modification process for the second set of parameters when the stop condition `DEL_V7_RISE <=2NS` is satisfied.

33.6.1 Example 5: Output

Transient analysis output signals are shown graphically in Figure 33-15.

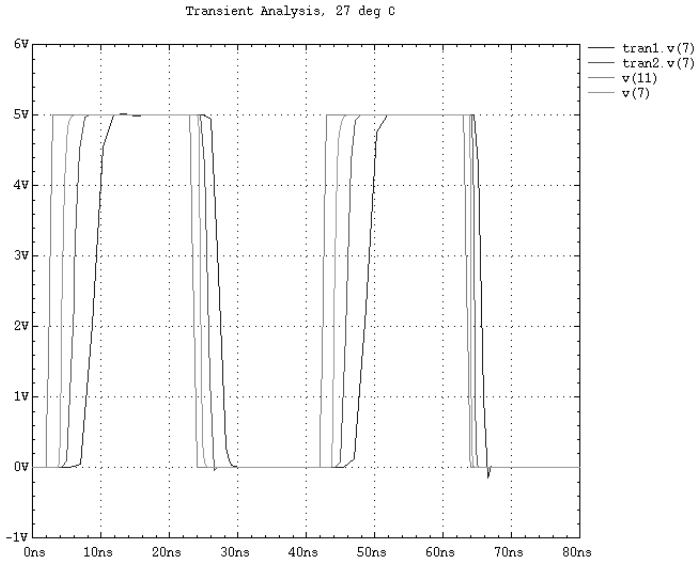


Figure 33-15 Transient Analysis Output Signals

The results of all .MODIF iterations are shown as follows.

```
MODIF : PARAMETER SET #1 STEP #1

measure del_v7_rise delay INP = v(11) OUT(TARG) = v(7)
  DELAY(OUT - INP) = 5.764e-09
  ARGout = 4.826e-08 ARGinp = 4.250e-08
measure del_v1_rise delay INP = v(11) OUT(TARG) = v(1)
  DELAY(OUT - INP) = 3.562e-09
  ARGout = 4.606e-08 ARGinp = 4.250e-08
measure del_v7_fall delay INP = v(11) OUT(TARG) = v(7)
  DELAY(OUT - INP) = 1.957e-09
  ARGout = 6.546e-08 ARGinp = 6.350e-08
measure del_v1_fall delay INP = v(11) OUT(TARG) = v(1)
  DELAY(OUT - INP) = 1.265e-09
  ARGout = 6.477e-08 ARGinp = 6.350e-08

MODIF : PARAMETER SET #2 STEP #1
MODIFIED PARAMETER VALUES : modp(cbd) = 2.000e-14
modn(cbd) = 2.000e-14

measure del_v7_rise delay INP = v(11) OUT(TARG) = v(7)
  DELAY(OUT - INP) = 3.071e-09
  ARGout = 4.557e-08 ARGinp = 4.250e-08
measure del_v1_rise delay INP = v(11) OUT(TARG) = v(1)
  DELAY(OUT - INP) = 1.878e-09
  ARGout = 4.438e-08 ARGinp = 4.250e-08
measure del_v7_fall delay INP = v(11) OUT(TARG) = v(7)
  DELAY(OUT - INP) = 1.066e-09
  ARGout = 6.457e-08 ARGinp = 6.350e-08
measure del_v1_fall delay INP = v(11) OUT(TARG) = v(1)
```

```
  DELAY(OUT - INP) = 6.861e-10
  ARGout = 6.419e-08 ARGinp = 6.350e-08

MODIF : PARAMETER SET #2 STEP #2
MODIFIED PARAMETER VALUES :
modp(cbd) = 1.000e-14 modn(cbd) = 1.000e-14

measure del_v7_rise delay INP = v(11) OUT(TARG) = v(7)
  DELAY(OUT - INP) = 1.716e-09
  ARGout = 4.422e-08 ARGinp = 4.250e-08
measure del_v1_rise delay INP = v(11) OUT(TARG) = v(1)
  DELAY(OUT - INP) = 1.019e-09
  ARGout = 4.352e-08 ARGinp = 4.250e-08
measure del_v7_fall delay INP = v(11) OUT(TARG) = v(7)
  DELAY(OUT - INP) = 5.754e-10
  ARGout = 6.408e-08 ARGinp = 6.350e-08
measure del_v1_fall delay INP = v(11) OUT(TARG)= v(1)
  DELAY(OUT - INP) = 3.411e-10
  ARGout = 6.384e-08 ARGinp = 6.350e-08

MODIF :STOP CONDITION HAS BEEN SATISFIED
FOR PARAMETER SET #2 STEP #2
MODIF :del_v7_rise = 1.716e-09
```

33.7 Example 6: Two-Bit BJT Adder

BJT Nand, one-bit, and two-bit subcircuit schematics are shown in Figure 33-16, Figure 33-17, and Figure 33-18, respectively.

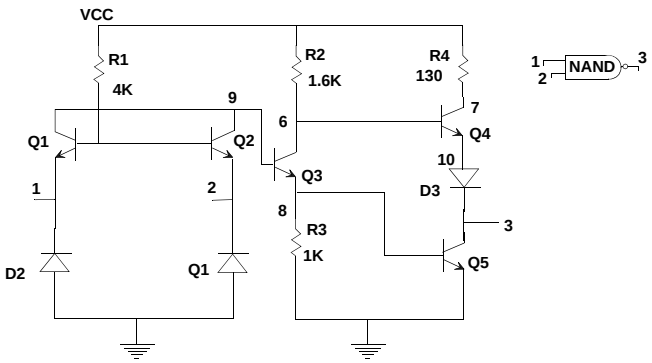


Figure 33-16 BJT NAND Subcircuit

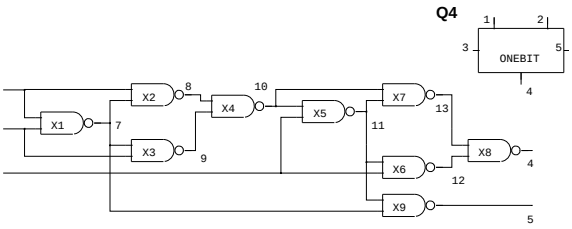


Figure 33-17 One-Bit Subcircuit

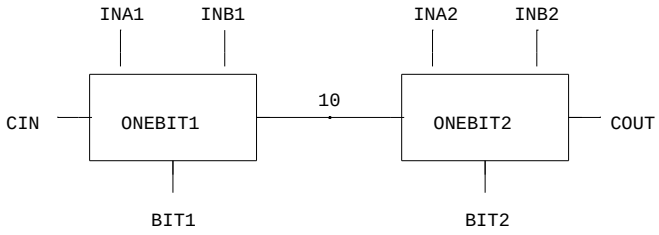


Figure 33-18 Two-Bit Subcircuit

```
* EXAMPLE 6 : Two-Bit BJT Adder
*
.GLOBAL VCC
.SUBCKT NAND 1 2 3
.ENDS NAND
```

Q1	9	5	1	QNL
Q2	9	5	2	QNL
Q3	6	9	8	QNL
Q4	7	6	10	QNL
Q5	3	8	0	QNL
D1	0	1	DIOD	
D2	0	2	DIOD	
D3	10	3	DIOD	
R1	VCC	5	4K	

```
R2      VCC     6       1.6K
R3      8       0       1K
R4      VCC     7       130

************* INA INB CIN OUT COUT
.SUBCKT ONEBIT 1 2 3 4 5
X1      1       2       7       NAND
X2      1       7       8       NAND
X3      2       7       9       NAND
X4      8       9       10      NAND
X5      3       10      11      NAND
X6      3       11      12      NAND
X7      10      11      13      NAND
X8      12      13      4       NAND
X9      11      7       5       NAND
.ENDS ONEBIT
.SUBCKT TWOBIT INA1 INB1 INA2 INB2 BIT1 BIT2 CIN COUT
X1      INA1    INB1    CIN     BIT1    10      ONEBIT
X2      INA2    INB2    10      BIT2    COUT    ONEBIT
.ENDS TWOBIT
VINA1     1     0     PULSE (0 3 0 10NS 10NS 10NS 50NS)
VINB1     2     0     PULSE (0 3 0 10NS 10NS 20NS 100NS)
VINA2     3     0     PULSE (0 3 0 10NS 10NS 40NS 200NS)
VINB2     4     0     PULSE (0 3 0 10NS 10NS 80NS 400NS)
X1        1     2     3     4     BIT1     BIT2     0     COUT     TWOBIT
RB1       BIT1      0     1K
RB2       BIT2      0     1K
RCOUT     COUT      0     1K
VCC       VCC       0     DC   5
.MODEL QNL NPN(BF=75 RB=100 CJE=1PF CJC=3PF)
.MODEL DIOD D
.TRAN 1NS 400NS
.MEASURE TRAN MAX_BIT1 MAX V(BIT1)
.MEASURE TRAN DEL_BIT1_INA1_RISE DELAY V(1) RISE=4
+ VAL=1.5 TARG=V(BIT1) RISE=2 VAL0=0 VAL1=MAX_BIT1
.MEASURE TRAN DEL_BIT1_INA1_FALL DELAY V(1) FALL=4
+ VAL=1.5 TARG=V(BIT1) FALL=2 VAL0=0 VAL1=MAX_BIT1
.OPTIONS ACCT RELTOL=0.05 NOMOD
.END
```

33.7.1 Example 6: Output

The graphic analysis of the transient output signal is shown in Figure 33-19, and the transient input signal is shown in Figure 33-20.

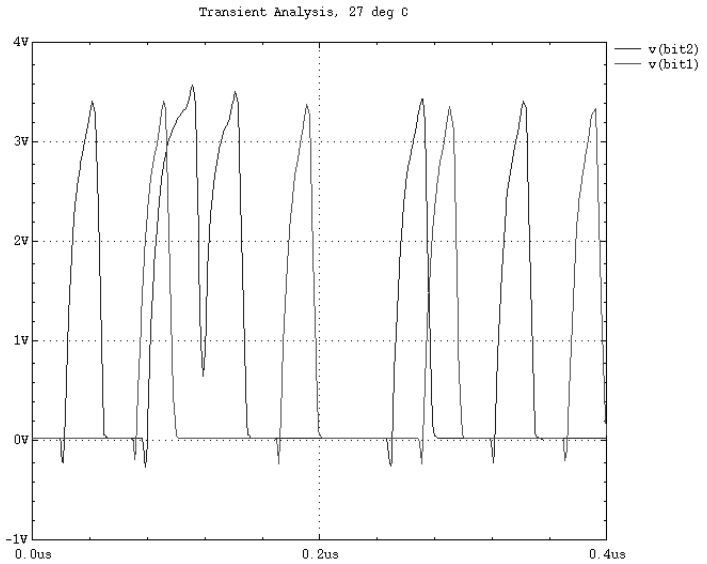


Figure 33-19 Transient Analysis Output Signals

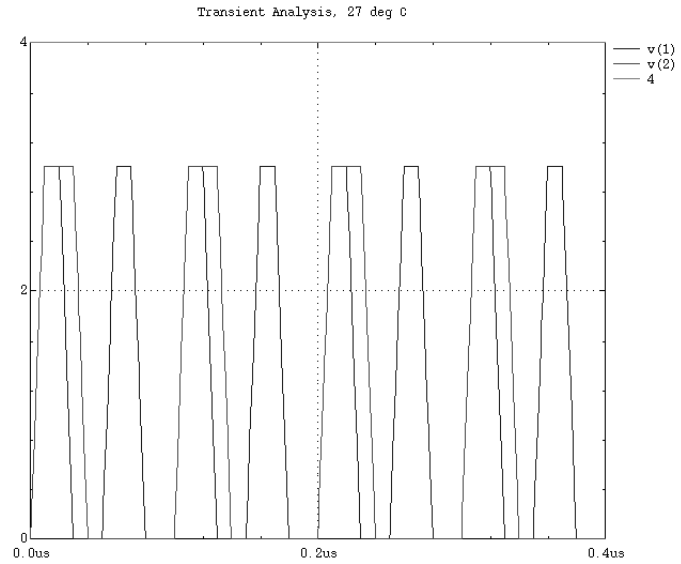


Figure 33-20 Transient Analysis Input Signal

The numerical results of the two-bit BJT adder simulation are shown below:

```
measure max_bit1 max OUT = v(bit1)
  Ymax = 3.426e+00
  Xmax = 9.167e-08
measure del_bit1_ina1_rise delay INP = v(1) OUT(TARG) =
v(bit1)
  DELAY(OUT - INP) = 2.545e-08
  ARGout = 1.805e-07 ARGinp = 1.550e-07
measure del_bit1_ina1_fall delay INP = v(1) OUT(TARG) =
v(bit1)
  DELAY(OUT - INP) = 2.093e-08
  ARGout = 1.959e-07 ARGinp = 1.750e-07
```

33.8 Example 7: Two-Bit MOSFET Adder

MOSFET NAND subcircuit schematic is shown in Figure 33-21.

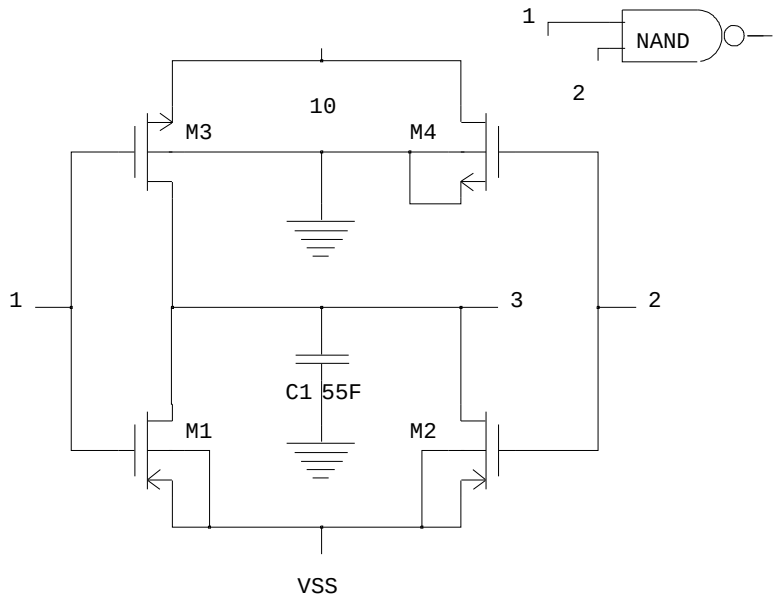


Figure 33-21 MOSFET NAND Subcircuit

```
* EXAMPLE 7 : TWO-BIT MOSFET ADDER
*
*
.GLOBAL VSS
.SUBCKT NAND 1 2 3
M1   3    1   VSS    VSS    MOSP   W=10U    L=1.3U
M2   3    2   VSS    VSS    MOSP   W=10U    L=1.3U
M3   3    1   10     0      MOSN   W=5U     L=1.3U
M4   10   2   0      0      MOSN   W=5U     L=1.3U
C1   3    0   55FF
.ENDS NAND
************* INA INB CIN OUT COUT
.SUBCKT ONEBIT 1 2 3 4 5
X1   1    2    7    NAND
X2   1    7    8    NAND
X3   2    7    9    NAND
X4   8    9    10   NAND
X5   3    10   11   NAND
X6   3    11   12   NAND
X7   10   11   13   NAND
X8   12   13   4    NAND
X9   11   7    5    NAND
.ENDS ONEBIT
```

```
.SUBCKT TWOBIT INA1 INB1 INA2 INB2 BIT1 BIT2 CIN COUT
X1   INA1    INB1    CIN    BIT1    10      ONEBIT
X2   INA2    INB2    10     BIT2    COUT    ONEBIT
.ENDS TWOBIT
VINA1      1      0    PULSE(2.8 0.4 0 1NS 1NS 14NS 30NS)
VINB1      2      0    PULSE(2.8 0.4 0 1NS 1NS 29NS 60NS)
VINA2      3      0    PULSE(2.8 0.4 0 1NS 1NS 14NS 30NS)
VINB2      4      0    PULSE(2.8 0.4 0 1NS 1NS 29NS 60NS)
VINCIN     CIN    0    PULSE(0.4 2.8 0 1NS 1NS 6.5NS 15NS)
VSS        VSS    0    DC   4.5
X1         1      2    3    4    BIT1   BIT2   CIN   COUT TWOBIT
.MODEL MOSP PMOS(LEVEL=3 VTO=-0.8 TOX=0.41E-7 UO=150
+ LD=0.14U
+ KP=14U ETA=0.014 THETA=0.042 VMAX=4.8E4 NFS=1E11
+ NSS=2E10 NSUB=2E16 JS=0.1M GAMMA=0.68 KAPPA=0.24
+ XJ=0.45U RSH=85 PB=0.7 CJ=0.2M CJSW=0.2N)
.MODEL MOSN NMOS(LEVEL=3 VTO=0.7 TOX=0.41E-7 UO=500
+ LD=0.1U KP=28U ETA=0.01 THETA=0.042 VMAX=1.9E5
+ NFS=1E11 NSS=2E10 NSUB=9E16 JS=0.1M GAMMA=1.52
+ KAPPA=0.24 XJ=0.45U RSH=85 PB=0.7 CJ=0.2M CJSW=0.2N)
*
.TRAN 1NS 60NS
.MEASURE TRAN MAX_BIT1 MAX V(BIT1)
.MEASURE TRAN MAX_COUT1 MAX V(X1.10)
.MEASURE TRAN DEL_BIT1_CIN DELAY
+ V(CIN) FALL=1 VAL=1.4
+ TARG=V(BIT1) FALL=1 VAL0=0 VAL1=MAX_BIT1
.MEASURE TRAN FALL_BIT1 WAVE
+ V(BIT1) FROM=40NS FALL=1 VAL0=0 VAL1=4.5
.MEASURE TRAN WIDTH_COUT1 DELAY
+ V(X1.10) RISE=1 VAL0=0 VAL1=MAX_COUT1
+ TARG=V(X1.10)FALL=1 VAL0=0 VAL1=MAX_COUT1
.OPTIONS ACCT NOMOD
.END
```

33.8.1 Example 7: Output

Transient analysis input and output signals are shown graphically in Figure 33-22.

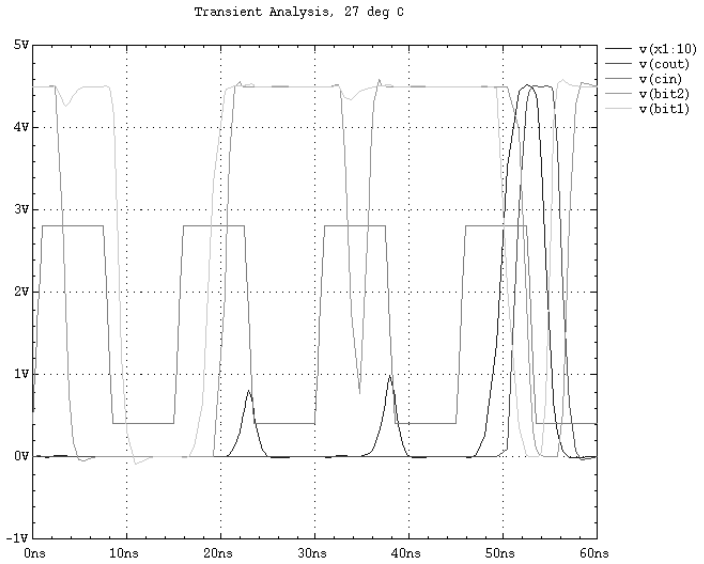


Figure 33-22 Transient Analysis Input and Output Signals

The numerical results of the two-bit MOSFET adder simulation are shown below:

```
measure max_bit1 max OUT = v(bit1)
  Ymax = 4.525e+00
  Xmax = 2.318e-08
measure max_cout1 max OUT = v(x1.10)
  Ymax = 4.501e+00
  Xmax = 5.291e-08
measure del_bit1_cin delay INP = v(cin) OUT(TARG) =
v(bit1)
  DELAY(OUT - INP) = 1.139e-09
  ARGout = 9.222e-09 ARGinp = 8.083e-09
measure fall_bit1 wave OUT = v(bit1)
  FALL DURATION(ARG2 - ARG1) = 1.474e-09
  ARG2 = 5.147e-08 ARG1 = 4.999e-08 FALL #1
measure width_cout1 delay INP = v(x1.10) OUT(TARG) =
v(x1.10)
  DELAY(OUT - INP) = 5.048e-09
  RGout = 5.469e-08 ARGinp = 4.964e-08
```

33.9 Example 8: Boost Converter

This example of analog behavioral modeling contains linear and nonlinear controlled sources of all types. The macro model of the Boost Converter is essentially the same as in the paper by Aram, et al.,: *Unified SPICE Compatible Average Model of PWM Converters*, IEEE Transactions on Power Electronics, Vol. 6, No. 4, (October 1991), pp. 585-594.

```
* EXAMPLE 8 : BOOST CONVERTER
* THIS INPUT FILE CONTAINS THE LINEAR AND NONLINEAR
* CONTROLLED SOURCES OF ALL TYPES
* SmartSpice CALCULATES THE BIAS POINT BY MEANS OF
* TRANSIENT ANALYSIS AND THEN PERFORMS AC ANALYSIS.
*
VIN         1       0       DC          10
RTR         2       0       0.01
D1          3       5       DNN
RCC         6       5       .07
C1          6       0       516U
R1          5               0           20
VDON        4       0       AC          1       DC .4
GDON        4       0       4           0       1.0E-4
AB1         2       1       i=i(HL)
FB2         2       1       VCDEMI      -1
AC          3       1000    i=i(ADON)*i(A7001)
VCDEMI      1000    1       DC          0
ADON        2001    2000    v=v(4)*(v(2)-v(1))
ADOF        2000    0       v=v(6100)*(v(3)-v(1))
HL          2001    2003    HL          0.4
L1          2003    0       254U
A3000       3000    0       v=((v(3)-v(1))^2)+1e-8
R3000       3000    0       1MEG
A3001       3010    0       v=v(3002)*100
R3001       3001    0       10MEG
RLIM        3001    3010    100
DLIM        3020    3001    DIN
VLIM        3020    0       DC          0.0     5
A3002       3002    3003    v=(i(ADON)^2)+1e-8
R3002       3002    0       1MEG
A3003       0       3003    v=(v(3001)^2)*v(3000)
E4000       4000    0       4001        0       100
R4000       4000    0       1MEG
A4001       4001    4002    v=v(3001)
R4001       4001    0       1MEG
A4002       0       4002    v=v(4000)*v(4003)
A4003       4003    0       v=v(4004)*v(4)
R4003       4003    0       1MEG
E4004       4010    0       4000        0       29
```

```
R4004       4004       0          1MEG
RLIMIT      4004       4010       50
DLIMIT      4020       4000        DIN
VLIMIT      4020       0          DC          0.0       1
A6000       6000       0          v=100-100*v(4)
E6002       6002       0          4004        0         100
R6002       6002       6003        2K
D6003       6003       6000        DIN
EDOFF       6100       0          6003        0         0.01
RDOFF       6100       0           100K
A7000       7000       0          v=v(6100)+v(4)
R7000       7000       0          1MEG
A7001       7001       0          v=v(7002)*1000 + 1E-3
R7001       7001        0          1
E7002       002        7003       6100        0         1
R7002       7002       0          1MEG
A7003       0          7003       v=v(7001)*v(7000)

.MODEL  DNN  D (IS=1.E-13 RS=.1 IBV=1.E-16)
.MODEL  DIN  D (IS=1.E-7 N=.85 RS=0.01 IBV=1.E-16)
*
.MODIF LOOP=3 VDON(DC)+=(0.3)0.1 GDON(TRCON)=1E-4
+ FB2(GAIN)=-1 HL(TRRES)=0.4 E4004(GAIN)=29 PRTBL
+MODIF LOOP=3 VDON(DC)+=(0.3)0.1 HL(TRRES)=0.5 PRTBL
*
.TRAN 10U 10M TRANOP SAVEV=10M
.AC DEC 10 1 10K CALLV UIC
.MEASURE TRAN maxV5 MAX V(5)
.MEASURE AC min_PH_V5 MIN VP(5)
.MEASURE AC expr_PH_V5 expr val="min_PH_V5*180/3.146"
.OPTION NOMOD
.END


* EXAMPLE 8 : BOOST CONVERTER
* COMMENTS
*
VIN      1   0   DC      10
RTR      2   0   0.01
D1       3   5   DNN
RCC      6   5   .07
C1       6   0   516U
R1       5   0   20
VDON     4   0   AC      1     DC    .4
GDON     4   0   4       0     1.0E-4
```

This is a linear voltage-controlled current source. It is connected between node `4` and node `0`, and also uses these nodes as the controlling nodes. `1.0E-4` is the trans conductance of the source.

```
AB1 2 1 i=i(HL)
```

This is a nonlinear current-controlled current source. It is connected between node 2 and node 1. The controlling current i(HL) flows through the linear controlled voltage source HL from node 2001 to node 2003.

```
FB2 2 1 VCDEMI -1
```

This is a linear current-controlled current source. The controlling current flows through the independent voltage source VCDEMI from node 1000 to node 1. -1 is the current gain.

```
AC 3 1000 i=i(ADON)*i(A7001)
```

This is a nonlinear current-controlled current source.

```
VCDEMI    1000    1       DC    0
ADON      2001    2000    v=v(4)*(v(2)-v(1))
```

This is a nonlinear voltage-controlled current source. It uses three controlling node voltages: v(1), v(2) and v(4). Its current i(ADON) is a basic variable of the system of equations describing circuit behavior. Voltage sources of all types, namely V, E, H and A (if the last one is a voltage source) create basic current variables. You are allowed to use them as controlling currents. Note: GDON, AB1, FB2 and AC commented on above are not voltage sources.

```
ADOF 2000 0 v=v(6100)*(v(3)-v(1))
HL 2001 2003 HL 0.4
```

This is a linear current-controlled voltage source. It uses its current as the controlling variable. 0.4 is the transresistance.

```
L1        2003    0       254U
A3000     3000    0       v=((v(3)-v(1))^2)+1e-8
R3000     3000    0       1MEG
A3001     3010    0       v=v(3002)*100
R3001     3001    0       10ME
                          G
RLIM      3001    3010    100
DLIM      3020    3001    DIN
VLIM      3020    0       DC    0.05
A3002     3002    3003    v=(i(ADON)^2)+1e-8
```

This is a nonlinear current-controlled voltage source.

```
R3002     3002    0       1MEG
A3003     0       3003    v=(v(3001)^2)*v(3000)
E4000     4000    0       4001    0    100
```

This is a linear voltage-controlled voltage source. 100 is the voltage gain.

```
R4000     4000    0       1MEG
A4001     4001    4002    v=v(3001)
R4001     4001    0        1MEG
A4002     0       4002    v=v(4000)*v(4003)
A4003     4003    0       v=v(4004)*v(4)
R4003     4003    0       1MEG
E4004     4010    0       4000    0       29
R4004     4004    0       1MEG
RLIMIT    4004    4010    50
```

```
DLIMIT    4020    4000    DIN
VLIMIT    4020    0       DC      0.01
A6000     6000    0       v=100-100*v(4)
E6002     6002    0       4004    0       100
R6002     6002    6003    2K
D6003     6003    6000    DIN
EDOFF     6100    0       6003    0       0.01
RDOFF     6100    0        100K
A7000     7000    0       v=v(6100)+v(4)
R7000     7000    0       1MEG
A7001     7001    0       v=v(7002)*1000 + 1E-3
R7001     7001    0       1
E7002     7002    7003    6100    0       1
R7002     7002    0       1MEG
A7003     0       7003    v=v(7001)*v(7000)

.MODEL DNN D (IS=1.E-13 RS=.1 IBV=1.E-16)
.MODEL DIN D (IS=1.E-7 N=.85 RS=0.01 IBV=1.E-16)

.MODIF LOOP=3 VDON(DC)+=(0.3)0.1 GDON(TRCON)=1E-4
+ FB2(GAIN)=-1 HL(TRRES)=0.4  E4004(GAIN)=29 PRTBL
+ MODIF LOOP=3 VDON(DC)+=(0.3)0.1 HL(TRRES)=0.5 PRTBL
```

The `.MODIF` statement contains two sets of parameters. For the first set, SmartSpice sets parameters of the controlled sources `GDON`, `FB2`, `HL` and `E4004` to values and sweeps the input voltage source `VDON` from `0.3V` to `0.5V` performing three steps. For the second set it resets `TRRES` of the controlled source `HL`, and sweeps `VDON` again. On each step it performs transient and AC analysis and calculates measures. SmartSpice prints the tables of parameters and measures for both sets of parameters.

```
.TRAN 10U 10M TRANOP SAVEV=10M
```

SmartSpice computes the transient operating point (steady state), then performs the actual transient analysis starting from found operating point. It saves the vector of basic variables at the final time point of the transient analysis.

```
.AC DEC 10 1 10K CALLV UIC
```

SmartSpice calls the vector of basic variables, linearizes the circuit for this vector, then performs the AC analysis.

```
.MEASURE TRAN maxV5 MAX V(5)
.MEASURE AC min_PH_V5 MIN VP(5)
.MEASURE AC expr_PH_V5 expr val="min_PH_V5*180/3.146"
.OPTION FORMAT=0
.END
```

33.9.1 Example 8: Output

The boost converter AC output analysis is shown in Figure 33-23.

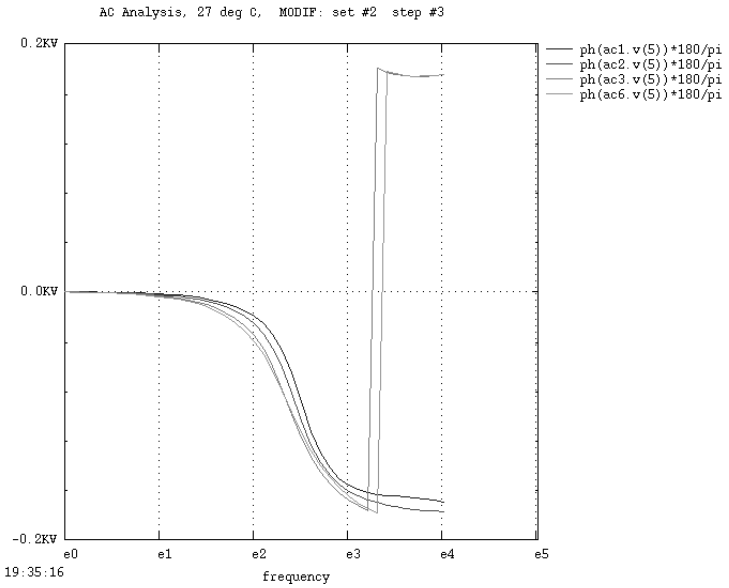


Figure 33-23 Boost Converter Output

The numerical results of the Boost Converter simulation are shown below:

```
MODIF : PARAMETER SET #2 STEP #1

MODIFIED PARAMETER VALUES :
vdon(dc) = 3.000e-01 hl(trres) = 5.000e-01

measure maxv5 max OUT = v(5)
  Ymax = 1.279e+01
  Xmax = 5.120e-04

measure min_ph_v5 min OUT = vp(5)
  Ymin = -2.958e+00
  Xmin = 1.000e+04 (end of measure interval)

measure expr_ph_v5 expr
  VAL = -1.695e+02

MODIF : PARAMETER SET #2 STEP #2

MODIFIED PARAMETER VALUES :
vdon(dc) = 4.000e-01 hl(trres) = 5.000e-01

measure maxv5 max OUT = v(5)
  Ymax = 1.477e+01
  Xmax = 1.024e-03

measure min_ph_v5 min OUT= vp(5)
  Ymin = -3.098e+00
  Xmin = 1.000e+04 (end of measure interval)
measure expr_ph_v5 expr
  VAL = -1.775e+02
```

```
MODIF : PARAMETER SET #2 STEP #3

MODIFIED PARAMETER VALUES :
vdon(dc) = 5.000e-01 hl(trres) = 5.000e-01

measure maxv5 max OUT = v(5)
  Ymax = 1.735e+01
  Xmax = 4.048e-03

measure min_ph_v5 min OUT = vp(5)
  Ymin = -3.122e+00
  Xmin = 1.995e+03
measure expr_ph_v5 expr
  VAL = -1.789e+02
```

The parameter and measurement results for Set #2 are listed below.

vdon(dc)	hl(trres)	maxv5	min_ph_v5	expr_ph_v5
3.000e-01	5.000e-01	1.279e+01	-2.958e+00	-1.695e+02
4.000e-01	5.000e-01	1.477e+01	-3.098e+00	-1.775e+02
5.000e-01	5.000e-01	1.735e+01	-3.122e+00	-1.789e+02

33.10 Example 9: Cell Characterization Using .MODIF and .ALTER

The .ALTER statement causes two sets of simulations to be executed. The first portion of the deck contains a .MODIF statement that modifies the specified parameter labels three times and performs a transient simulation for each modification. After the .ALTER statement, the MOSFET models are changed, and the .MODIF statement is also changed. The new .MODIF statement results in the transient simulation being performed six more times.

An example of cell characterization using .MODIF and .ALTER is given in this netlist.

```
* EXAMPLE 9 : Cell Characterization using .MODIF and .ALTER
* Characterization of a cell using a combination of the
* .MODIF and .ALTER statements.
************** Optional Parameters
.OPTIONS NOMOD
.OPTIONS SCALE=1u
.OPTIONS RELTOL=0.002
.OPTIONS POST=2
************** .MODIF Statement
.MODIF loop=3 vd=    LIST(5   4.5   5.5)
+             rise = LIST(2n  1.5n  2.5n)
+             fall = LIST(2n  1.5n  2.5n)
+             cap  = LIST(1pf 0.75p 1.25p)
+             PRTBL
+             PROFF
************** Parameter Labels
.PARAM vd=5 rise=2n fall=2n
.PARAM pw=10 nw=4  length=1.2 pl=1.2 cap= 1pf
.PARAM  tox = 2.400E-8  delta = 0.2e-8
.PARAM  vto0 = -0.9 deltavto=0.04 vtop='vto0+deltavto'

************** Global Net Declarations
.GLOBAL GND vdd
************** Circuit  Netlist
Vcc vdd GND  'vd'
Vin inp GND  DC 0 PULSE(0 vd 0ns rise fall 10ns 24ns )
Mn mid inp GND GND nmod W='nw' L='length'
Mp mid inp vdd vdd pmod W=pw L=length
X1  mid out  inv  nl=1.2 pl=1.4 vbb=0.0
************** Subcircuit  Definition
.SUBCKT inv  input output  nl=1.3  pl=1.3 vbb=-1
M1 output input GND vbias  nmod W=nw L=nl
M2 output input vdd vdd  pmod W=pw L=pl
Cload  output 0  'cap'
Vb  vbias  0 dc  'vbb'
.ENDS inv
************** Analysis Statement
.TRAN  0.1n 24n
************** Output Statements
.SAVE i(c.X1.Cload)
+     @m.X1.M1[L]
+     @nmod[TOX]
.LET power='ABS(i(Vcc)*v(vdd))'
.LET Gcap = '@Mn[cgtot]'
.MEAS TRAN del_rise DELAY v(inp) VAL='vd/2' RISE=1
+                   TARG v(out)  VAL='vd/2' RISE=1
.MEAS TRAN del_fall TRIG v(inp) VAL='vd/2'  FALL=1
```

```
+                  TARG v(out)  VAL='vd/2' FALL=1
.MEAS TRAN  diff    EXPR  VAL='del_fall-del_rise'
.MEAS TRAN  avg_pow  AVG  power
.MEAS TRAN  parL     MAX @m.X1.M1[L]
**************************************************
* Model Definitions
*
* Fast-Fast Transistor Model
*
.MODEL nmod     NMOS (TOX= 'tox+delta'
+ BEX=-1.5                          LD=1.850E-7
+ WD=0.000E-7     NSUB=3.000E16     VTO=0.720
+ UO=630.00       DELTA=0.000       VMAX=1.950E5
+ XJ=3.0E-7       KAPPA=0.100       ETA=0.026
+ THETA=0.060     TPG=1             NFS=1.20E11
+ CJ=420E-6       CJSW=271E-12      PB=0.940
+ MJ=0.430        MJSW=0.160        CGDO=3.2E-10
+ CGSO=3.2E-10    CGBO=0
+ RSH=35.0        CAPMOD=2          LEVEL=3     )
*
.model pmod     PMOS (TOX= 'tox+delta'
+  BEX=-1.5                          LD=2.70E-7
+  WD=-0.500E-7    NSUB=2.700E16     VTO= 'vtop'
+  UO=190.00       DELTA=0.000       VMAX=4.600E5
+  XJ=4.0E-7       KAPPA=0.100       ETA=0.042
+  THETA=0.100     TPG=1             NFS=0.50E11
+  CJ=392E-6       CJSW=373E-12      PB=0.880
+  MJ=0.460        MJSW=0.210        CGDO=3.2E-10
+  CGSO=3.2E-10    CGBO=0
+  RSH=60.0        LEVEL=3           )

************** .ALTER Statement
.ALTER
**************************************************
* Model Definitions
*
* Slow-Slow Transistor Model
*
.MODEL nmod     NMOS (TOX= 'tox+delta'
+ BEX=-1.5                          LD=1.850E-7
+ WD=0.000E-7     NSUB=3.000E16     VTO=0.720
+ UO=600.00       DELTA=0.000       VMAX=1.950E5
+ XJ=3.0E-7       KAPPA=0.100       ETA=0.026
+ THETA=0.060     TPG=1             NFS=1.20E11
+ CJ=520E-6       CJSW=300E-12      PB=0.940
+ MJ=0.430        MJSW=0.160        CGDO=4.2E-10
+ CGSO=4.2E-10    CGBO=0
+ RSH=35.0        CAPMOD=2          LEVEL=3     )
*
.model pmod     PMOS (TOX= 'tox+delta'
+ BEX=-1.5                          LD=2.70E-7
+ WD=-0.500E-7    NSUB=2.700E16     VTO= 'vtop'
+ UO=170.00       DELTA=0.000       VMAX=4.600E5
+ XJ=4.0E-7       KAPPA=0.100       ETA=0.042
+ THETA=0.100     TPG=1             NFS=0.50E11
+ CJ=410E-6       CJSW=400E-12      PB=0.880
```

```
+ MJ=0.460         MJSW=0.210          CGDO=4.2E-10
+ CGSO=4.2E-10     CGBO=0
+ RSH=60.0         LEVEL=3             )

************** .MODIF Statement
.MODIF loop=3 TEMP=27
+              vd=    LIST(5   4.5   5.5 )
+              rise = LIST(2n  1.5n  2.5n )
+              fall = LIST(2n  1.5n  2.5n )
+              cap  = LIST(1pf 0.75p 1.25p)
+              PRTBL
+              PROFF
+MODIF loop=3 TEMP= 75
+              vd=    LIST(5   4.5   5.5 )
+              rise = LIST(2n  1.5n  2.5n )
+              fall = LIST(2n  1.5n  2.5n )
+              cap  = LIST(1pf 0.75p 1.25p)
+              PRTBL
+              PROFF
.END
```

33.10.1 EXAMPLE 9 Output

	vd	rise	fall	cap	del_rise
1	5.0000e+00	2.0000e-09	2.0000e-09	1.0000e-12	1.5535e-09
2	4.5000e+00	1.5000e-09	1.5000e-09	7.5000e-13	1.2916e-09
3	5.5000e+00	2.5000e-09	2.5000e-09	1.2500e-12	1.7835e-09
	del_fall	**diff**	**avg_pow**	**parl**	
1	2.0911e-09	5.3761e-10	1.2092e-03	1.2000e+00	
2	1.7250e-09	4.3342e-10	7.3214e-04	1.2000e+00	
3	2.4246e-09	6.4112e-10	1.8445e-03	1.2000e+00	

```
After .ALTER #1
```

	temp	vd	rise	fall	cap
1	2.7000e+01	5.0000e+00	2.0000e-09	2.0000e-09	1.0000e-12
2	2.7000e+01	4.5000e+00	1.5000e-09	1.5000e-09	7.5000e-13
3	2.7000e+01	5.5000e+00	2.5000e-09	2.5000e-09	1.2500e-12
	del_rise	**del_fall**	**diff**	**avg_pow**	**parl**
1	1.7024e-09	2.1991e-09	4.9675e-10	1.2085e-03	1.2000e+00
2	1.4268e-09	1.8174e-09	3.9063e-10	7.3423e-04	1.2000e+00
3	1.9452e-09	2.5434e-09	5.9815e-10	1.8388e-03	1.2000e+00

	temp	vd	rise	fall	cap
1	7.5000e+01	5.0000e+00	2.0000e-09	2.0000e-09	1.0000e-12
2	7.5000e+01	4.5000e+00	1.5000e-09	1.5000e-09	7.5000e-13
3	7.5000e+01	5.5000e+00	2.5000e-09	2.5000e-09	1.2500e-12
	del_rise	**del_fall**	**diff**	**avg_pow**	**parl**
1	1.9972e-09	2.4629e-09	4.6572e-10	1.1959e-03	1.2000e+00
2	1.6719e-09	2.0412e-09	3.6932e-10	7.2891e-04	1.2000e+00
3	2.2808e-09	2.8453e-09	5.6447e-10	1.8149e-03	1.2000e+00

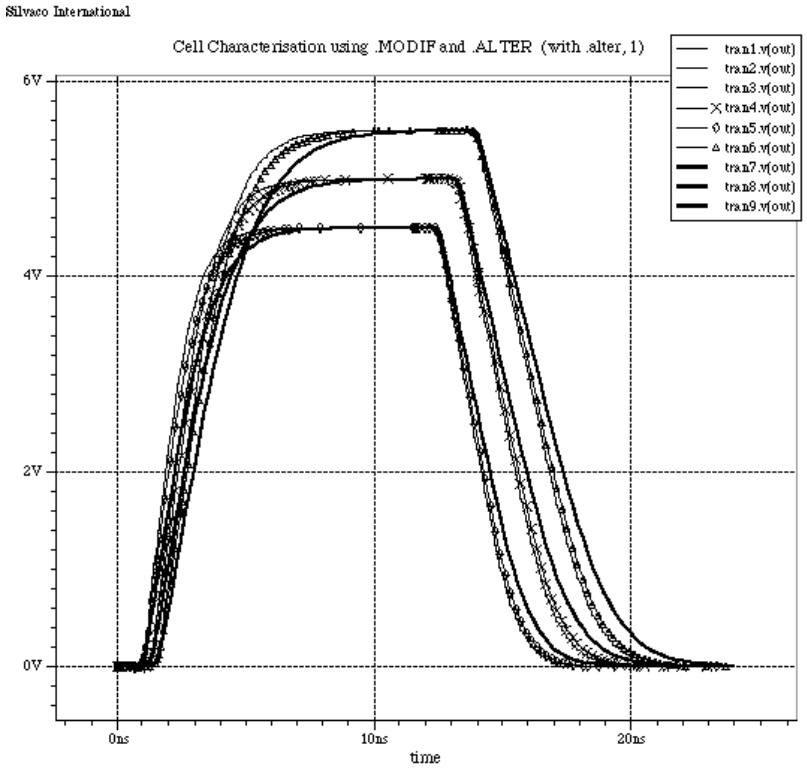


Figure 33-24 Cell Characterization using .MODIF and .ALTER

Chapter 34
Encryption Module

34.1 Introduction

SmartSpice can use any file from an input deck in an encrypted form. This is a one-way process, no De-Encrypt program is provided so it is important to keep the source file safe and not discard this. See the Sencrypt User's Manual for further details.

34.2 Process

1. Run the encryption program to create an encrypted file (see Sencrypt User's Manual for syntax)
2. In your SPICE deck use "`.include encrypted_file`" (same syntax/use as a standard file `include`)
3. Encryption can be done at any level, from `.model`, `.subckt`, `.include` up to and including the full netlist.

This allows SmartSpice to run the encrypted file, but if anyone tries to print the file they will get a "jumbled mess" that is not understandable.

Encryption can be done at any level of the SmartSpice input deck structure. This means right from the main netlist to included files and model libraries down to individual models file contents can be protected and unreadable. This feature has already found applications, such as joint company ventures, where netlist circuit blocks can be used for simulation, but Intellectual properties are confidential.

Since these encrypted files are only usable inside SmartSpice, it recognizes the contents should be protected and all viewing functions that would allow the contents display are automatically disabled. This ensures the protected circuit can be evaluated for electrical performance but the contents remain confidential. Because the encryption process is made so secure there is no way to reconstruct the original contents. It is recommended to keep 2 versions of the files as:

- Encrypted file for 3rd party use.
- The original readable file.

Figure 34-1 is a Diagrammatic example of the process:

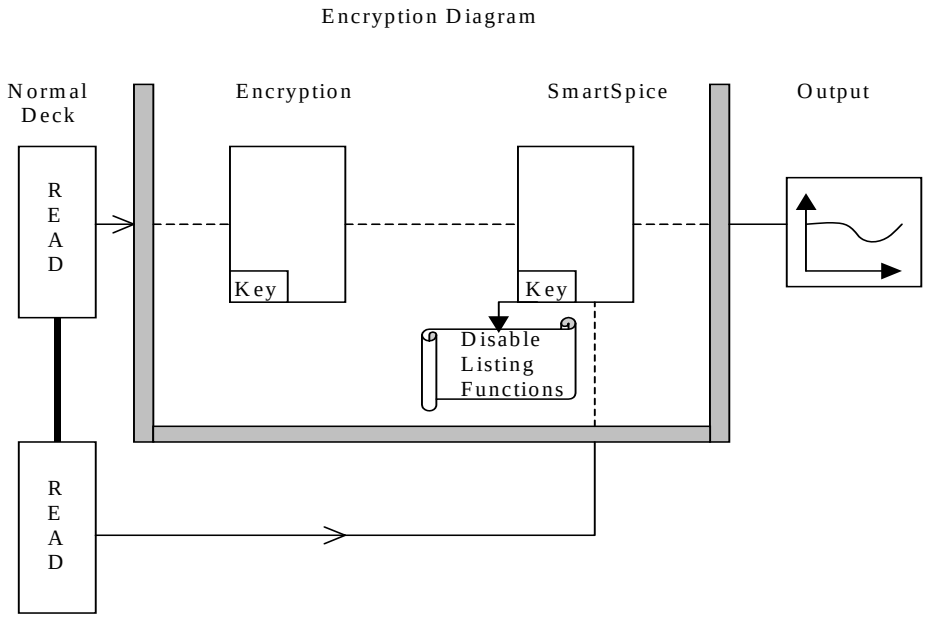


Figure 34-1 Encryption Diagram

The bottom section in the diagram above may or may not exist depending whether all or just part of the simulation files are encoded.

34.2.1 Output Protection

Because SmartSpice protects encrypted descriptions, output information may be different for encrypted netlist comparing with the same non-encrypted netlist:

- Model-related warning or error messages will not be printed for encrypted models.
- Command `NODELIST` (and `.option NODE`) will not print encrypted nodes.
- Command `SHOW` (and `.show` statement) will not print model parameters for encrypted models, instance names and parameters for encrypted instances.
- Command `PRINTPAR` will not print model parameters for encrypted models, instance names and parameters for encrypted instances.
- Operating point output will not contain any information about encrypted nodes and instances.
- Floating nodes output will not contain any information about encrypted nodes.
- Terminal connected together warning messages will not be printed for encrypted instances.
- Any error and warning messages will not be printed for encrypted input lines. Thus any module should be checked for errors before encryption.

Chapter 35
SmartSpice Licensing

35.1 Sleep Mode and License Timeout

In GUI mode, SmartSpice can release a license if no user or simulation activity is detected during a specified period of time.

This license functionality is controlled only by the import/export preferences capabilities as shown enabled in Figure 35-1.

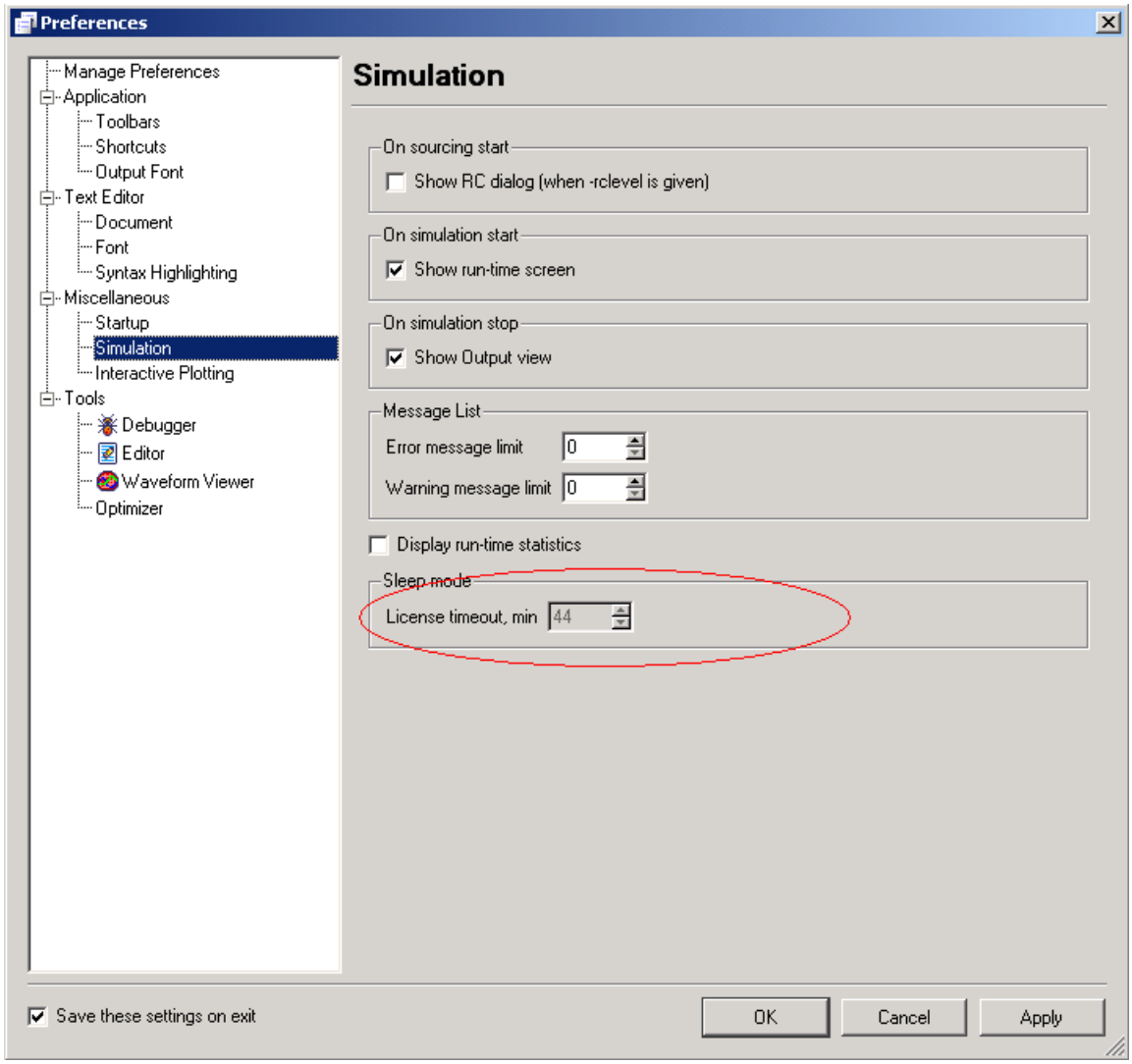


Figure 35-1 Sleep Mode/License Timeout Enabled

By default the Sleep Mode is off and the License Timeout is set to 30 minutes (inactive).

To activate:

1. Export preferences to the `.spf` file (**Edit→Preferences→Manage Preferences→Export**);
2. Find in the `.spf` file the following entries and adjust them as required:
   - `General/SleepMode = true or false`
   - `General/LicenseTimeout = 1 ... 9999 [min]`
3. Import preferences from this file back to SmartSpice (**Edit→Preferences→Manage Preferences→Import**);

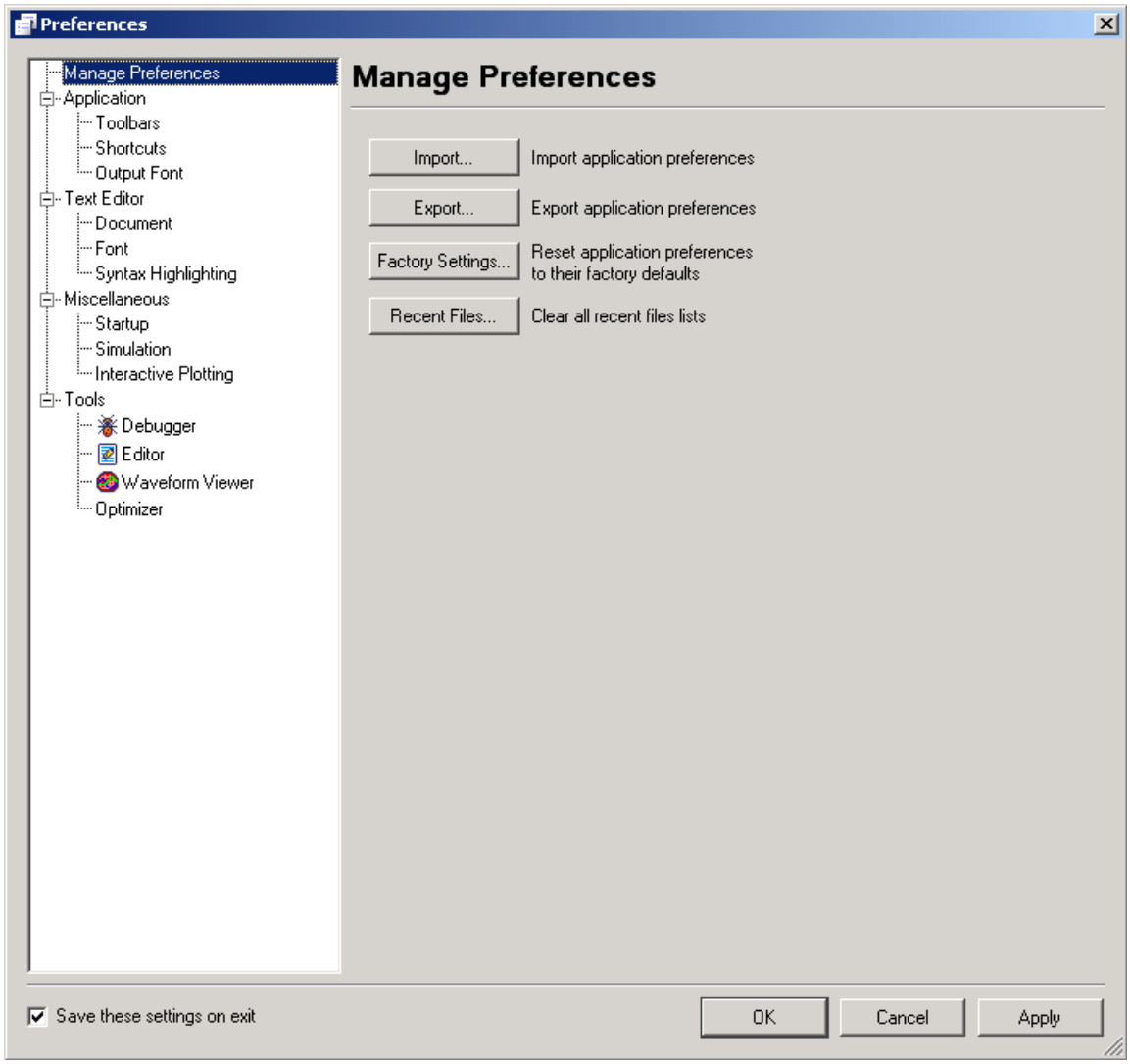


Figure 35-2 Import/Export Preferences

When you exit SmartSpice these settings will be saved and re-enabled the next time you start SmartSpice.

Appendix A
Transient Simulation with Large Data Sets

A.1 Overview

During simulation, it is likely that large amounts of data will be generated. This can lead to a number of difficulties for SmartSpice in terms of available memory and dynamic allocation of memory. If the amount of data generated is large, SmartSpice may run out of available operating memory. (The host computer's configuration is also an important factor). This will typically cause SmartSpice to discontinue the simulation and exit. SmartSpice allocates memory as needed, but dynamic reallocation of memory can cause the available memory to be exhausted even faster.

To reduce the amount of data stored in operating memory, during simulation, SmartSpice stores only those vectors that are required for post-processing. The SmartSpice statements that can influence the number of vectors SmartSpice will store for postprocessing are:

.SAVE	.MEAS<URE>
.PROBE </CSDF>	.LET
.PLOT	.MC
.PRINT	.WCASE
.IPLOT	.FOUR
.IPRINT	.GRAPH

Of these statements, .SAVE and .PROBE/CSDF generally create the largest amount of data. The statements:

.SAVE ALL	Stores all of the node voltages and currents through independent voltage sources.
.PROBE </CSDF>	Statement with no argument saves all available node voltages and device currents. See Chapter 3 Statements for detailed descriptions of these statements.

Even if the set of voltages and currents saved is the minimum required, the amount of data stored can still become quite large. To alleviate this problem, we have provided a number of options that can be specified in the .OPTIONS statement. See Chapter 3 Statements.

LIMPTS	Determines the maximum number of data points SmartSpice will include in the transient analysis. SmartSpice will stop simulating when the value of LIMPTS is reached.
RAWPTS	Affects the way SmartSpice stores transient analysis results in a raw file in batch mode. SmartSpice dumps data into the rawfile at every RAWPTS point. The number of points stored in memory will always be less than or equal to RAWPTS. If you specify this option, none of the output statements (.PRINT, .GRAPH, .MEASURE, etc.) are executed. Thus, we suggest using the RAWPTS option when a large amount of output data is expected. SmartSpice sets the option RAWPTS by default to 500 if the option POST is given.

For example, the following statement will cause SmartSpice to stop simulating when 50,000 data points have been processed:

```
.OPTIONS LIMPTS = 50000
```

The following statement will append the most recent 3000 data points to the raw file every 3000 data points.

```
.OPTIONS RAWPTS = 3000
```

These options work best for the standard ASCII rawfile format and the `.PROBE` format. However, the `.PROBE/CSDF` format does not work as well, especially when SmartSpice reloads the rawfile. During the simulation, the rawfile is kept in a consistent state so that you can use another SmartSpice process to load in an intermediate rawfile and view the current state of the simulation. This will only work, however, for the standard ASCII rawfile and `.PROBE` rawfile.

You can also use the `LIMPTS` and `RAWPTS` options while running SmartSpice in Window Mode, using the command:

```
run batchprint
```

The `batchprint` keyword will execute all output control statements from the input deck (`.PRINT`, `.PLOT`, `.MEASURE`, etc.) in the same manner as they would be executed in batch mode.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z