Ngspice Users Manual
Version 22plus
(This manual describes the actual CVS version)

Paolo Nenzi, Holger Vogt

March 14, 2011

Locations

The project and download pages of ngspice may be found at
    Ngspice home page http://ngspice.sourceforge.net/
    Project page at sourceforge http://sourceforge.net/projects/ngspice/
    Download page at sourceforge http://sourceforge.net/projects/ngspice/files/
    CVS source download http://sourceforge.net/scm/?type=cvs&group_id=38962

Status

This manual is a work in progress. Some todos are listed in the following. More is surely needed. You are invited to report bugs, missing items, wrongly described items, bad English style etc.

To Do

1. Review of chapt. 1.3

2. .func

3. hfet1,2, jfet2 model descriptions

4. tclspice compilation chapt. 20.5

5. adms chapt. 14

6. more examples

7. LINUX graphics interface chapt. 19.2

Part I

Ngspice User Manual

Contents

Prefaces

Preface to the first edition

This manual has been assembled from different sources:

1. The spice3f5 manual,

2. the Xspice user's manual,

3. the CIDER user's manual

and some original material needed to describe the new features and the newly implemented models. This cut and paste approach, while not being ortodox, allowed ngspice to have a full manual in a fraction of the time that writing a completely new text would have required. The use of LaTex and Lyx instead of TeXinfo, which was the original encoding for the manual, further helped to reduce the writing effort and improved the quality of the result, at the expense of an online version of the manual but, due to the complexity of the software I hardly think that users will ever want to read an online text version.

In writing this text I followed the cut of spice3f5 manual, both in the chapter sequence and presentation of material, mostly because that was already the user manual of spice.

Ngspice is an open source software, users can download the source code, compile, and run it. This manual has an entire chapter describing program compilation and available options to help users in building ngspice (see chapt. 31). The source package already comes with all "safe" options enabled by default, and activating the others can produce unpredictable results and thus is recommended to expert users only. This is the first ngspice manual and I have removed all the historical material that described the differences between ngspice and spice3, since it was of no use for the user and not so useful for the developer who can look for it in the Changelogs of in the revision control system.

I want to acknowledge the work dome Emmanuel Rouat and Arno W. Peters for converting to TeXinfo the original spice3f documentation, their effort gave ngspice users the only available documentation that described the changes for many years. A good source of ideas for this manual comes from the online spice3f manual written by Charles D.H. Williams (Spice3f5 User Guide), constantly updated and useful for some insight that he gives in it.

As always, errors, omissions and unreadable phrases are only my fault.
Paolo Nenzi
Roma, March 24th 2001

> Indeed. At the end of the day, this is engineering, and one learns to live
> within the limitations of the tools.

Kevin Aylward , Warden of the Kings Ale

Preface to the actual edition (as of 2010)

Due to the wealth of new material and options in ngspice the actual order of chapters has been revised. Several new chapters have been added. Thy lyx text processors has allowed adding internal cross references. The pdf format has become the standard format for distribution of the manual. Within each new ngspice distribution (starting with ngspice-21) a manual edition is provided reflecting the ngspice status at the time of distribution. At the same time, located at ngspice manuals, the manual is constantly updated. Every new ngspice feature should enter this manual as soon as it has been made available in the CVS source code.
Holger Vogt
Mülheim, 2010

Acknowledgments

ngspice contributors

Spice was originally written at The University of California at Berkeley (USA).

Since then, there have been many people working on the software, most of them releasing patches to the original code through the Internet.

The following people have contributed in some way:

Vera Albrecht,
Cecil Aswell,
Giles C. Billingsley,
Phil Barker,
Steven Borley,
Stuart Brorson,
Mansun Chan,
Wayne A. Christopher,
Al Davis,
Glao S. Dezai,
Jon Engelbert,
Daniele Foci,
Noah Friedman,
David A. Gates,
Alan Gillespie,
John Heidemann,
Jeffrey M. Hsu,
JianHui Huang,
S. Hwang,
Chris Inbody,
Gordon M. Jacobs,
Min-Chie Jeng,
Beorn Johnson,
Stefan Jones,
Kenneth H. Keller,
Robert Larice,
Mathew Lew,
Robert Lindsell,
Weidong Liu,
Kartikeya Mayaram,
Richard D. McRoberts,
Manfred Metzger,
Wolfgang Muees,
Paolo Nenzi,
Gary W. Ng,
Hong June Park,
Arno Peters,
Serban-Mihai Popescu,
Georg Post,
Thomas L. Quarles,
Emmanuel Rouat,

Jean-Marc Routure,
Jaijeet S. Roychowdhury,
Lionel Sainte Cluque,
Takayasu Sakurai,
Amakawa Shuhei,
Kanwar Jit Singh,
Bill Swartz,
Hitoshi Tanaka,
Steve Tell,
Andrew Tuckey,
Andreas Unger,
Holger Vogt,
Dietmar Warning,
Michael Widlok,
Charles D.H. Williams,
Antony Wilson,
and many others...

If someone helped in the development and has not been inserted in this list then this omission was unintentional. If you feel you should be on this list then please write to <ngspice-devel@lists.sourceforge.net>. Do not be shy, we would like to make a list as complete as possible.

XSPICE

Chapter 1

Introduction

Ngspice is a general-purpose circuit simulation program for nonlinear and linear analyses. Circuits may contain resistors, capacitors, inductors, mutual inductors, independent or dependent voltage and current sources, lossless and lossy transmission lines, switches, uniform distributed RC lines, and the five most common semiconductor devices: diodes, BJTs, JFETs, MESFETs, and MOSFETs.

Ngspice is an update of Spice3f5, the last Berkeley's release of Spice3 simulator family. Ngspice is being developed to include new features to existing Spice3f5 and to fix its bugs. Improving a complex software like a circuit simulator is a very hard task and, while some improvements have been made, most of the work has been done on bug fixing and code refactoring.

Ngspice has built-in models for the semiconductor devices, and the user need specify only the pertinent model parameter values. There are three models for bipolar junction transistors, all based on the integral-charge model of Gummel and Poon; however, if the Gummel-Poon parameters are not specified, the basic model (BJT) reduces to the simpler Ebers-Moll model. In either case and in either models, charge storage effects, ohmic resistances, and a current-dependent output conductance may be included. The second bipolar model BJT2 adds dc current computation in the substrate diode. The third model (VBIC) contains further enhancements for advanced bipolar devices.

The semiconductor diode model can be used for either junction diodes or Schottky barrier diodes. There are two models for JFET: the first (JFET) is based on the model of Shichman and Hodges, the second (JFET2) is based on the Parker-Skellern model. All the original six MOSFET models are implemented: MOS1 is described by a square-law I-V characteristic, MOS2 [1] is an analytical model, while MOS3 [1] is a semi-empirical model; MOS6 [2] is a simple analytic model accurate in the short channel region; MOS9, is a slightly modified Level 3 MOSFET model - not to confuse with Philips level 9; BSIM 1 [3, 4]; BSIM2 [5] are the old BSIM (Berkeley Short-channel IGFET Model) models. MOS2, MOS3, and BSIM include second-order effects such as channel-length modulation, subthreshold conduction, scattering-limited velocity saturation, small-size effects, and charge controlled capacitances. The recent MOS models for submicron devices are the BSIM3 (Berkeley BSIM3 web page) and BSIM4 (Berkeley BSIM4 web page) models. Silicon-on-insulator MOS transistors are described by the SOI models from the BSIMSOI family (Berkeley BSIMSOI web page) and the STAG [18] one. There is partial support for a couple of HFET models and one model for MESA devices.

Ngspice supports mixed-level simulation and provides a direct link between technology parameters and circuit performance. A mixed-level circuit and device simulator can provide greater simulation accuracy than a stand-alone circuit or device simulator by numerically modeling the critical devices in a circuit. Compact models can be used for noncritical devices. The mixed-level extensions to ngspice are two:

- CIDER: a mixed-level circuit and device simulator integrated into ngspice code. CIDER was originally the name of the mixed-level extension made to spice3f5.

- GSS: GSS (now called GENIUS) TCAD is a 2D simulator developed independently from ngspice. The device simulator itself is free and not included into ngspice, but a socket interface is provided.

Ngspice supports mixed-signal simulation through the integration of XSPICE code into it. Xspice software, developed as an extension to Spice3C1 from GeorgiaTech, has been ported to ngspice to provide "board" level and mixed-signal simulation.

New devices can be added to ngspice by two means: the xspice old code-model interface and the new ADMS interface based on Verilog-A and XML.

Previously computed states can be loaded into the program to provide accurate initial guesses for subsequent analysis. Finally, numerous small bugs have been discovered and fixed, and the program has been ported to a wider variety of computing platforms.

1.1 Simulation Algorithms

Computer-based circuit simulation is often used as a tool by designers, test engineers, and others who want to analyze the operation of a design without examining the physical circuit. Simulation allows you to change quickly the parameters of many of the circuit elements to determine how they affect the circuit response. Often it is difficult or impossible to change these parameters in a physical circuit.

However, to be practical, a simulator must execute in a reasonable amount of time. The key to efficient execution is choosing the proper level of modeling abstraction for a given problem. To support a given modeling abstraction, the simulator must provide appropriate algorithms.

Historically, circuit simulators have supported either an analog simulation algorithm or a digital simulation algorithm. Ngspice inherits the XSPICE framework and supports both analog and digital algorithms and is a "mixed-mode" simulator.

1.1.1 Analog Simulation

Analog simulation focuses on the linear and non-linear behavior of a circuit over a continuous time or frequency interval. The circuit response is obtained by iteratively solving Kirchhoff's Laws for the circuit at time steps selected to ensure the solution has converged to a stable value and that numerical approximations of integrations are sufficiently accurate. Since Kirchhoff's laws form a set of simultaneous equations, the simulator operates by solving a matrix of equations at each time point. This matrix processing generally results in slower simulation times when compared to digital circuit simulators.

The response of a circuit is a function of the applied sources. Ngspice offers a variety of source types including DC, sinewave, and pulse. In addition to specifying sources, the user must define the type of simulation to be run. This is termed the "mode of analysis". Analysis modes include DC analysis, AC analysis, and transient analysis. For DC analysis, the time-varying behavior of reactive elements is neglected and the simulator calculates the DC solution of the circuit. Swept DC analysis may also be accomplished with ngspice. This is simply the repeated application of DC analysis over a range of DC levels for the input sources. For AC analysis, the simulator determines the response of the circuit, including reactive elements to small-signal sinusoidal inputs over a range of frequencies. The simulator output in this case includes amplitudes and phases as a function of frequency. For transient analysis, the circuit response, including reactive elements, is analyzed to calculate the behavior of the circuit as a function of time.

1.1.2 Digital Simulation

Digital circuit simulation differs from analog circuit simulation in several respects. A primary difference is that a solution of Kirchhoff's laws is not required. Instead, the simulator must only determine whether a change in the logic state of a node has occurred and propagate this change to connected elements. Such a change is called an "event".

When an event occurs, the simulator examines only those circuit elements that are affected by the event. As a result, matrix analysis is not required in digital simulators. By comparison, analog simulators must iteratively solve for the behavior of the entire circuit because of the forward and reverse transmission properties of analog components. This difference results in a considerable computational advantage for digital circuit simulators, which is reflected in the significantly greater speed of digital simulations.

1.1.3 Mixed-Mode Simulation

Modern circuits often contain a mix of analog and digital circuits. To simulate such circuits efficiently and accurately a mix of analog and digital simulation techniques is required. When analog simulation algorithms are combined with digital simulation algorithms, the result is termed "mixed-mode simulation".

Two basic methods of implementing mixed-mode simulation used in practice are the "native mode" and "glued mode" approaches. Native mode simulators implement both an analog algorithm and a digital algorithm in the same executable. Glued mode simulators actually use two simulators, one of which is analog and the other digital. This type of simulator must define an input/output protocol so that the

two executables can communicate with each other effectively. The communication constraints tend to reduce the speed, and sometimes the accuracy, of the complete simulator. On the other hand, the use of a glued mode simulator allows the component models developed for the separate executables to be used without modification.

Ngspice is a native mode simulator providing both analog and event-based simulation in the same executable. The underlying algorithms of ngspice (coming from XSPICE and its Code Model Subsystem) allow use of all the standard SPICE models, provide a pre-defined collection of the most common analog and digital functions, and provide an extensible base on which to build additional models.

User-Defined Nodes

Ngspice supports creation of "User-Defined Node" types. User-Defined Node types allow you to specify nodes that propagate data other than voltages, currents, and digital states. Like digital nodes, User-Defined Nodes use event-driven simulation, but the state value may be an arbitrary data type. A simple example application of User-Defined Nodes is the simulation of a digital signal processing filter algorithm. In this application, each node could assume a real or integer value. More complex applications may define types that involve complex data such as digital data vectors or even non-electronic data.

Ngspice digital simulation is actually implemented as a special case of this User-Defined Node capability where the digital state is defined by a data structure that holds a Boolean logic state and a strength value.

1.1.4 Mixed-Level Simulation

Ngspice can simulate numerical device models for diodes and transistors in two different ways, either through the integrated DSIM simulator or interfacing to GSS TCAD system. DSIM is an internal C-based device simulator which is part of the CIDER simulator, the mixed-level simulator based on spice3f5. CIDER within ngspice provides circuit analyses, compact models for semiconductor devices, and one- or two-dimesional numerical device models.

CIDER (DSIM)

DSIM provides accurate, one- and two-dimensional numerical device models based on the solution of Poisson's equation, and the electron and hole current-continuity equations. DSIM incorporates many of the same basic physical models found in the Stanford two-dimensional device simulator PISCES. Input to CIDER consists of a SPICE-like description of the circuit and its compact models, and PISCES-like descriptions of the structures of numerically modeled devices. As a result, CIDER should seem familiar to designers already accustomed to these two tools. CIDER is based on the mixed-level circuit and device simulator CODECS, and is a replacement for this program. The basic algorithms of the two programs are the same. Some of the differences between CIDER and CODECS are described below. The CIDER input format has greater flexibility and allows increased access to physical model parameters. New physical models have been added to allow simulation of state-of-the-art devices. These include transverse field mobility degradation important in scaled-down MOSFETs and a polysilicon model for poly-emitter bipolar transistors. Temperature dependence has been included over the range from -50C to 150C. The numerical models can be used to simulate all the basic types of semiconductor devices: resistors, MOS capacitors, diodes, BJTs, JFETs and MOSFETs. BJTs and JFETs can be modeled with or without a substrate contact. Support has been added for the management of device internal states. Post-processing of device states can be performed using the ngnutmeg user interface.

GSS TCAD

GSS is a TCAD software which enables two-dimensional numerical simulation of semiconductor device with well-known drift-diffusion and hydrodynamic method. GSS has Basic DDM (drift-diffusion method) solver, Lattice Temperature Corrected DDM solver, EBM (energy balance method) solver and Quantum corrected DDM solver which based on density-gradient theory. The GSS program is directed via input statements by a user specified disk file. Supports triangle mesh generation and adaptive mesh refinement. Employs PMI (physical model interface) to support various materials, including compound semiconductor materials such as SiGe and AlGaAs. Supports DC sweep, transient and AC sweep calculations. The device can be stimulated by voltage or current source(s).

GSS is no longer updated, but is still available as open source as a limited edition of the commercial GENIUS TCAD tool.

1.2   Supported Analyses

The ngspice simulator supports the following different types of analysis:

1. DC Analysis (Operating Point and DC Sweep)

2. AC Small- Signal Analysis

3. Transient Analysis

4. Pole-Zero Analysis

5. Small-Signal Distortion Analysis

6. Sensitivity Analysis

7. Noise Analysis

Applications that are exclusively analog can make use of all analysis modes with the exception of Code
Model subsystem that do not implements Pole-Zero, Distortion, Sensitivity and Noise analyses. Event-
driven applications that include digital and User-Defined Node types may make use of DC (operating
point and DC sweep) and Transient only.

In order to understand the relationship between the different analyses and the two underlying simu-
lation algorithms of ngspice, it is important to understand what is meant by each analysis type. This is
detailed below.

1.2.1   DC Analyses

The dc analysis portion of ngspice determines the dc operating point of the circuit with inductors shorted
and capacitors opened. The dc analysis options are specified on the `.DC`, `.TF`, and `.OP` control lines.

There is assumed to be no time dependence on any of the sources within the system description.
The simulator algorithm subdivides the circuit into those portions which require the analog simulator
algorithm and those which require the event-driven algorithm. Each subsystem block is then iterated to
solution, with the interfaces between analog nodes and event-driven nodes iterated for consistency across
the entire system.

Once stable values are obtained for all nodes in the system, the analysis halts and the results may be
displayed or printed out as you request them.

A dc analysis is automatically performed prior to a transient analysis to determine the transient initial
conditions, and prior to an ac 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 is also computed as a part of the dc solution.
The dc analysis can also be used to generate dc transfer curves: a specified independent voltage, current
source, resistor or temperature[1] is stepped over a user-specified range and the dc output variables are
stored for each sequential source value.

1.2.2   AC Small-Signal Analysis

AC analysis is limited to analog nodes and represents the small signal, sinusoidal solution of the analog
system described at a particular frequency or set of frequencies. This analysis is similar to the DC analysis
in that it represents the steady-state behavior of the described system with a single input node *at a given
set of stimulus frequencies*.

The program first computes the dc operating point of the circuit and determines linearized, small-
signal models for all of the nonlinear devices in the circuit. The resultant linear circuit is then analyzed
over a user-specified range of frequencies. The desired output of an ac small-signal analysis is usually a
transfer function (voltage gain, transimpedance, etc). If the circuit has only one ac input, it is convenient
to set that input to unity and zero phase, so that output variables have the same value as the transfer
function of the output variable with respect to the input.

---

[1]Temperature (`TEMP`) and resistance sweeps have been introduced in Ngspice, they were not available in the original code
of Spice3f5.

1.2.3 Transient Analysis

Transient analysis is an extension of DC analysis to the time domain. A transient analysis begins by obtaining a DC solution to provide a point of departure for simulating time-varying behavior. Once the DC solution is obtained, the time-dependent aspects of the system are reintroduced, and the two simulator algorithms incrementally solve for the time varying behavior of the entire system. Inconsistencies in node values are resolved by the two simulation algorithms such that the time-dependent waveforms created by the analysis are consistent across the entire simulated time interval. Resulting time-varying descriptions of node behavior for the specified time interval are accessible to you.

All sources which are not time dependent (for example, power supplies) are set to their dc value. The transient time interval is specified on a `.TRAN` control line.

1.2.4 Pole-Zero Analysis

The pole-zero analysis portion of Ngspice computes the poles and/or zeros in the small-signal ac transfer function. The program first computes the dc operating point and then determines the linearized, small-signal models for all the nonlinear devices in the circuit. This circuit is then used to find the poles and zeros of the transfer function. Two types of transfer functions are allowed: one of the form (output voltage)/(input voltage) and the other of the form (output voltage)/(input current). These two types of transfer functions cover all the cases and one can find the poles/zeros of functions like input/output impedance and voltage gain. The input and output ports are specified as two pairs of nodes. The pole-zero analysis works with resistors, capacitors, inductors, linear-controlled sources, independent sources, BJTs, MOSFETs, JFETs and diodes. Transmission lines are not supported. The method used in the analysis is a sub-optimal numerical search. For large circuits it may take a considerable time or fail to find all poles and zeros. For some circuits, the method becomes "lost" and finds an excessive number of poles or zeros.

1.2.5 Small-Signal Distortion Analysis

The distortion analysis portion of Ngspice computes steady-state harmonic and 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 signals of two frequencies input to the circuit, the analysis finds out the complex values 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. Distortion analysis is supported for the following nonlinear devices:

- Diodes (DIO),

- BJT,

- JFET,

- MOSFETs (levels 1, 2, 3, 6, 9, BSIM1, BSIM2, BSIM3, BSIM4 and BSIMSOI),

- MESFETS.

All linear devices are automatically supported by distortion analysis. If there are switches present in the circuit, the analysis continues to be accurate provided the switches do not change state under the small excitations used for distortion calculations.

1.2.6 Sensitivity Analysis

Ngspice will calculate either the DC operating-point sensitivity or the AC small-signal sensitivity of an output variable with respect to all circuit variables, including model parameters. Ngspice calculates the difference in an output variable (either a node voltage or a branch current) by perturbing each parameter of each device independently. Since the method is a numerical approximation, the results may demonstrate second order affects in highly sensitive parameters, or may fail to show very low but non-zero sensitivity. Further, since each variable is perturb by a small fraction of its value, zero-valued parameters are not analyzed (this has the benefit of reducing what is usually a very large amount of data).

Algorithm 1.1 Instance temperature computation

IF TEMP is specified THEN
instance_temperature = TEMP
ELSE IF
instance_temperature = circuit_temperature + DTEMP
END IF

1.2.7   Noise Analysis

The noise analysis portion of Ngspice does analysis device-generated noise for the given circuit. When provided with an input source and an output port, the 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 the specified input source. This is done for every frequency point in a specified range - the calculated value of the noise corresponds to the spectral density of the circuit variable viewed as a stationary Gaussian stochastic process. After calculating the spectral densities, noise analysis integrates these values over the specified frequency range to arrive at the total noise voltage/current (over this frequency range). This calculated value corresponds to the variance of the circuit variable viewed as a stationary Gaussian process.

1.3   Analysis at Different Temperatures

Temperature, in ngspice, is a property associated to the entire circuit, rather an analysis option. Circuit temperature has a default (nominal) value of 27°C (300.15 K) that can be changed using the TNOM option in an .option control line. All analyses are, thus, performed at circuit temperature, and if you want to simulate circuit behavior at different temperatures you should prepare a netlist for each temperature.

All input data for ngspice is assumed to have been measured at the circuit nominal temperature. This value can further be overridden for any device which models temperature effects by specifying the TNOM parameter on the .model itself. Individual instances may further override the circuit temperature through the specification of TEMP and DTEMP parameters on the instance. The two options are not independent even if you can specify both on the instance line, the TEMP option overrides DTEMP. The algorithm to compute instance temperature is described below:

Temperature dependent support is provided for all devices except voltage and current sources (either independent and controlled) and BSIM models. BSIM MOSFETs have an alternate temperature dependency scheme which adjusts all of the model parameters before input to ngspice.

For details of the BSIM temperature adjustment, see [6] and [7]. Temperature appears explicitly in the exponential terms of the BJT and diode model equations. In addition, saturation currents have a built-in temperature dependence. The temperature dependence of the saturation current in the BJT models is determined by:

$$I_S\left(T_1\right) = I_S\left(T_0\right)\left(\frac{T_1}{T_0}\right)^{XTI}\exp\left(\frac{E_gq\left(T_1T_0\right)}{k\left(T_1-T_0\right)}\right) \tag{1.1}$$

where k is Boltzmann's constant, q is the electronic charge, E_g is the energy gap which is a model parameter, and XTI is the saturation current temperature exponent (also a model parameter, and usually equal to 3).

The temperature dependence of forward and reverse beta is according to the formula:

$$B\left(T_1\right) = B\left(T_0\right)\left(\frac{T_1}{T_0}\right)^{XTB} \tag{1.2}$$

where T_0 and T_1 are in degrees Kelvin, and XTB is a user-supplied model parameter. Temperature effects on beta are carried out by appropriate adjustment to the values of B_F, I_{SE}, B_R, and I_{SC} (spice model parameters BF, ISE, BR, and ISC, respectively).

Temperature dependence of the saturation current in the junction diode model is determined by:

$$I_S\left(T_1\right) = I_S\left(T_0\right)\left(\frac{T_1}{T_0}\right)^{\frac{XTI}{N}}\exp\left(\frac{E_gq\left(T_1T_0\right)}{Nk\left(T_1-T_0\right)}\right) \tag{1.3}$$

where N is the emission coefficient, which is a model parameter, and the other symbols have the same meaning as above. Note that for Schottky barrier diodes, the value of the saturation current temperature exponent, XTI, is usually 2. Temperature appears explicitly in the value of junction potential, U (in Ngspice PHI), for all the device models.

The temperature dependence is determined by:

$$U\left(T\right) = \frac{kT}{q}\ln\left(\frac{N_a N_d}{N_i\left(T\right)^2}\right) \tag{1.4}$$

where k is Boltzmann's constant, q is the electronic charge, N_a is the acceptor impurity density, N_d is the donor impurity density, N_i is the intrinsic carrier concentration, and E_g is the energy gap. Temperature appears explicitly in the value of surface mobility, M_0 (or U_0), for the MOSFET model.

The temperature dependence is determined by:

$$M_0\left(T\right) = \frac{M_0\left(T_0\right)}{\left(\frac{T}{T_0}\right)^{1.5}} \tag{1.5}$$

The effects of temperature on resistors, capacitor and inductors is modeled by the formula:

$$R\left(T\right) = R\left(T_0\right)\left[1 + TC_1\left(T - T_0\right) + TC_2\left(T - T_0\right)^2\right] \tag{1.6}$$

where T is the circuit temperature, T_0 is the nominal temperature, and TC_1 and TC_2 are the first and second order temperature coefficients.

1.4 Convergence

Ngspice use the Newton-Raphson algorithm to solve nonlinear equations arising from circuit description. The NR algorithm is interactive and terminates when both of the following conditions hold:

1. The nonlinear branch currents converge to within a tolerance of 0.1% or 1 picoamp (1.0e-12 Amp), whichever is larger.

2. The node voltages converge to within a tolerance of 0.1% or 1 microvolt (1.0e-6 Volt), whichever is larger.

1.4.1 Voltage convergence criterion

The algorithm has reached convergence if the difference between the last iteration k and the current one $(k + 1)$:

$$\left|v_n^{(k+1)} - v_n^{(k)}\right| \leq \texttt{RELTOL} * v_{n_{max}} + \texttt{VNTOL} \tag{1.7}$$

where

$$v_{n_{max}} = \max\left(\left|v_n^{(k+1)}\right|, \left|v_n^{(k)}\right|\right) \tag{1.8}$$

The `RELTOL` (RELative TOLerance) parameter, which default value is 10^{-3}, specifies how small the solution update must be, relative to the node voltage, to consider the solution to have converged. The `VNTOL` (absolute convergence) parameter, which has $1\mu V$ as default becomes important when node voltages have near zero values. The relative parameter alone, in such case, would need too strict tolerances, perhaps lower than computer round-off error, and thus convergence would never be achieved. `VNTOL` forces the algorithm to consider as converged any node whose solution update is lower than its value.

1.4.2 Current convergence criterion

Ngspice checks the convergence on the non-linear functions that describe the non-linear branches in circuit elements. In semiconductor devices the functions defines currents through the device and thus the name of the criterion.

Ngspice computes the difference between the value of the nonlinear function computed for last voltage and the linear approximation of the same current computed with the actual voltage:

$$\left| \widehat{i_{branch}^{(k+1)}} - i_{branch}^{(k)} \right| \leq \texttt{RELTOL} * i_{br_{max}} + \texttt{ABSTOL} \tag{1.9}$$

where

$$i_{br_{max}} = \max \left(\widehat{i_{branch}^{(k+1)}}, i_{branch}^{(k)} \right) \tag{1.10}$$

In the two expressions above, the $\widehat{i_{branch}}$ indicates the linear approximation of the current.

1.4.3   Convergence failure

Although the algorithm used in ngspice has been found to be very reliable, in some cases it fails to converge to a solution. When this failure occurs, the program terminates the job. Failure to converge in dc analysis is usually due to an error in specifying circuit connections, element values, or model parameter values. Regenerative switching circuits or circuits with positive feedback probably will not converge in the dc analysis unless the `OFF` option is used for some of the devices in the feedback path, `.nodeset` control line is used to force the circuit to converge to the desired state.

Chapter 2

Circuit Description

2.1   General Structure and Conventions

The circuit to be analyzed is described to ngspice by a set of element lines, which define the circuit topology and element values, and a set of control lines, which define the model parameters and the run controls. Two lines are essential:

- The first line in the input file must be the title, which is the only comment line that does not need any special character in the first place.

- The last line must be `.end`.

The order of the remaining lines is arbitrary (except, of course, that continuation lines must immediately follow the line being continued). This feature in the ngspice input language dates back to the punched card times where elements were written on separate cards (and cards frequently fell off). Leading white spaces in a line are ignored, as well as empty lines.

Each element in the circuit is specified by an element line that contains:

- the element name,

- the circuit nodes to which the element is connected,

- and the values of the parameters that determine the electrical characteristics of the element.

The first letter of the element name specifies the element type. The format for the ngspice element types is given in what follows. In the rest of the manual, the strings XXXXXXX, YYYYYYY, and ZZZZZZZ denote arbitrary alphanumeric strings.

For example, a resistor name must begin with the letter R and can contain one or more characters. Hence, R, R1, RSE, ROUT, and R3AC2ZY are valid resistor names. Details of each type of device are supplied in a following section 3.

Fields on a line are separated by one or more blanks, a comma, an equal (=) sign, or a left or right parenthesis; extra spaces are ignored. A line may be continued by entering a "+" (plus) in column 1 of the following line; ngspice continues reading beginning with column 2. A name field must begin with a letter (A through Z) and cannot contain any delimiters. A number field may be an integer field (12, -44), a floating point field (3.14159), either an integer or floating point number followed by an integer exponent (1e-14, 2.65e3), or either an integer or a floating point number followed by one of the following scale factors:

Suffix	Name	Factor
T	Tera	10^{12}
G	Giga	10^{9}
Meg	Mega	10^{6}
K	Kilo	10^{3}
mil	Mil	25.4×10^{-6}
m	milli	10^{-3}
u	micro	10^{-6}
n	nano	10^{-9}
p	pico	10^{-12}
f	femto	10^{-15}

Table 2.1: Ngspice scale factors

Letters immediately following a number that are not scale factors are ignored, and letters immediately following a scale factor are ignored. Hence, 10, 10V, 10Volts, and 10Hz all represent the same number, and M, MA, MSec, and MMhos all represent the same scale factor. Note that 1000, 1000.0, 1000Hz, 1e3, 1.0e3, 1kHz, and 1k all represent the same number.

Nodes names may be arbitrary character strings and are case insensitive. The ground node must be named "0" (zero). For compatibility reason "gnd" is accepted as ground node, and will internally be treated as a global node and be converted to "0". **Each circuit has to have a ground node (gnd or 0)!** Note the difference in ngspice where the nodes are treated as character strings and not evaluated as numbers, thus "0" and "00" are distinct nodes in ngspice but not in SPICE2.

Ngspice requires that the following topological constraints are satisfied:

- The circuit cannot contain a loop of voltage sources and/or inductors and cannot contain a cut-set of current sources and/or capacitors.

- Each node in the circuit must have a dc path to ground.

- Every node must have at least two connections except for transmission line nodes (to permit unterminated transmission lines) and MOSFET substrate nodes (which have two internal connections anyway).

2.2   Basic lines

2.2.1   .TITLE line

Examples:

```
POWER AMPLIFIER CIRCUIT
* additional lines following
*...


Test of CAM cell
* additional lines following
*...
```

The title line must be the first in the input file. Its contents are printed verbatim as the heading for each section of output.

As an alternative you may place a `.TITLE <any title>` line anywhere in your input deck. The first line of your input deck will be overriden by the contents of this line following the .TITLE statement.

.TITLE line example:

```
***************************
* additional lines following
*...
.TITLE Test of CAM cell
* additional lines following
*...
```

will internally be replaced by
Internal input deck:

```
Test of CAM cell
* additional lines following
*...
*TITLE Test of CAM cell
* additional lines following
*...
```

2.2.2   .END Line

Examples:

. end

The ".End" line must always be the last in the input file. Note that the period is an integral part of the name.

2.2.3   Comments

General Form:

* <any comment>

Examples:

```
* RF=1K Gain should be 100
* Check open−loop gain and phase margin
```

The asterisk in the first column indicates that this line is a comment line. Comment lines may be placed anywhere in the circuit description.

2.2.4   End-of-line comments

General Form:

<any command> ;<any comment>

Examples:

```
RF2=1K ;Gain should be 100
C1=10p $ Check open−loop gain and phase margin
```

ngspice supports comments that begin with single characters ';' or double characters '$ ' or '//' or '−'

2.3   Device Models

General form:

. model mname type(pname1=pval1 pname2=pval2 ...)

Examples:

. model MOD1 npn (bf=50 is=1e−13 vbf=50)

Most simple circuit elements typically require only a few parameter values. However, some devices (semiconductor devices in particular) that are included in ngspice require many parameter values. Often, many devices in a circuit are defined by the same set of device model parameters. For these reasons, a set of device model parameters is defined on a separate `.model` line and assigned a unique model name. The device element lines in ngspice then refer to the model name.

For these more complex device types, each device element line contains the device name, the nodes to which the device is connected, and the device model name. In addition, other optional parameters may be specified for some devices: geometric factors and an initial condition (see the following section on Transistors (9 to 12) and Diodes (8) for more details). `mname` in the above is the model name, and type is one of the following fifteen types:

Code	Model Type
R	Semiconductor resistor model
C	Semiconductor capacitor model
L	Inductor model
SW	Voltage controlled switch
CSW	Current controlled switch
URC	Uniform distributed RC model
LTRA	Lossy transmission line model
D	Diode model
NPN	NPN BJT model
PNP	PNP BJT model
NJF	N-channel JFET model
PJF	P-channel JFET model
NMOS	N-channel MOSFET model
PMOS	P-channel MOSFET model
NMF	N-channel MESFET model
PMF	P-channel MESFET model

Table 2.2: Ngspice model types

Parameter values are defined by appending the parameter name followed by an equal sign and the parameter value. Model parameters that are not given a value are assigned the default values given below for each model type. Models are listed in the section on each device along with the description of device element lines. Model parameters and their default values are given in chapter 30.

2.4   Subcircuits

A subcircuit that consists of ngspice elements can be defined and referenced in a fashion similar to device models. Subcircuits are the way ngspice implements hierarchical modeling, but this is not entirely true because each subcircuit instance is flattened during parsing, and thus ngspice is not a hierarchical simulator.

The subcircuit is defined in the input deck by a grouping of element cards delimited by the `.subckt` and the `.ends` cards (or the keywords defined by the substart and subend options (see 18.6)); the program then automatically inserts the defined group of elements wherever the subcircuit is referenced. Instances of subcircuits within a larger circuit are defined through the use of an instance card which begins with the letter "X". A complete example of all three of these cards follows:

```
Example:

* The following is the instance card:
*
xdiv1 10 7 0 vdivide

* The following are the subcircuit definition cards:
*
.subckt vdivide 1 2 3
r1 1 2 10K
r2 2 3 5K
.ends
```

The above specifies a subcircuit with ports numbered "1", "2" and "3":

- Resistor "R1" is connected from port "1" to port "2", and has value 10 kOhms.

- Resistor "R2" is connected from port "2" to port "3", and has value 5 kOhms.

The instance card, when placed in an ngspice deck, will cause subcircuit port "1" to be equated to circuit node "10", while port "2" will be equated to node "7" and port "3" will equated to node "0".

There is no limit on the size or complexity of subcircuits, and subcircuits may contain other subcircuits. An example of subcircuit usage is given in chapter 21.6.

2.4.1   .SUBCKT Line

General form:

.SUBCKT subnam N1 <N2 N3 ... >

Examples:

.SUBCKT OPAMP 1   2   3   4

A circuit definition is begun with a `.SUBCKT` line. SUBNAM is the subcircuit name, and N1, N2, ... are the external nodes, which cannot be zero. The group of element lines which immediately follow the `.SUBCKT` line define the subcircuit. The last line in a subcircuit definition is the `.ENDS` line (see below). Control lines may not appear within a subcircuit definition; however, subcircuit definitions may contain anything else, including other subcircuit definitions, device models, and subcircuit calls (see below). Note that any device models or subcircuit definitions included as part of a subcircuit definition are strictly local (i.e., such models and definitions are not known outside the subcircuit definition). Also, any element nodes not included on the `.SUBCKT` line are strictly local, with the exception of 0 (ground) which is always global. If you use parameters, the `.SUBCKT` line will be extended (see 2.8.3).

2.4.2   .ENDS Line

General form:

.ENDS <SUBNAM>

Examples:

.ENDS OPAMP

The `.ENDS` line must be the last one for any subcircuit definition. The subcircuit name, if included, indicates which subcircuit definition is being terminated; if omitted, all subcircuits being defined are terminated. The name is needed only when nested subcircuit definitions are being made.

2.4.3   Subcircuit Calls

General form:

XYYYYYYY N1 <N2 N3 ... > SUBNAM

Examples:

X1 2 4 17 3 1 MULTI

Subcircuits are used in ngspice by specifying pseudo-elements beginning with the letter X, followed by the circuit nodes to be used in expanding the subcircuit. If you use parameters, the subcircuit call will be modified (see 2.8.3).

2.5   .GLOBAL

General form:

.GLOBAL nodename

Examples:

.GLOBAL gnd vcc

Nodes defined in the .GLOBAL statement are available to all circuit and subcircuit blocks independently from any circuit hierarchy. After parsing the circuit, these nodes are accessible from top level.

2.6   .INCLUDE

General form:

.INCLUDE filename

Examples:

.INCLUDE /users/spice/common/wattmeter.cir

Frequently, portions of circuit descriptions will be reused in several input files, particularly with common models and subcircuits. In any ngspice input file, the .INCLUDE line may be used to copy some other file as if that second file appeared in place of the .INCLUDE line in the original file.

There is no restriction on the file name imposed by ngspice beyond those imposed by the local operating system.

2.7   .LIB

General form:

.LIB filename libname

Examples:

.LIB /users/spice/common/mosfets.lib mos1

The .LIB statement allows to include library descriptions into the input file. Inside the *.lib file a library **libname** may be selected. The statements of each library inside the *.lib file are enclosed in .LIB libname <...> .ENDL statements.

2.8   .PARAM Parametric netlists

Ngspice allows for the definition of parametric attributes in the netlists. This is an enhancement of the ngspice front-end which adds arithmetic functionality to the circuit description language.

2.8.1   .param line

General form:

.param <ident> = <expr> ; <ident> = <expr> ....

Examples:

.param pippo=5
.param pp=6
.param pippp={pippo + pp}
.param p={pp}
.param pap='pp+p'

This line assigns numerical values to identifiers. More than one assignment per line is possible using the ';' separator. The .param lines inside subcircuits are copied per call, like any other line. All assignments are executed sequentially through the expanded circuit. Before its first use, a parameter name must have been assigned a value. Expression defining a parameter have to be put into braces {p+p2}, alternatively into single quotes 'p+p2'.

2.8.2   Brace expressions in circuit elements:

General form:

{ <expr> }

Examples:
These are allowed in .model lines and in device lines. A spice number is a floating point number with an optional scaling suffix, immediately glued to the numeric tokens (see chapt. 2.8.5). Brace expressions

({..}) cannot be used to parametrize node names or parts of names. All identifiers used within an `<expr>` must have known values at the time when the line is evaluated, else an error is flagged.

2.8.3   Subcircuit parameters

General form:

.subckt <identn> node node ...   <ident>=<value> <ident>=<value> ...

Examples:

.subckt myfilter in out rval=100k cval=100nF

`<identn>` is the name of the subcircuit given by the user.   `node` is an integer number or an identifier, for one of the external nodes. The first `<ident>=<value>` introduces an optional section of the line. Each `<ident>` is a formal parameter, and each `<value>` is either a spice number or a brace expression. Inside the ".subckt" ... ".ends" context, each formal parameter may be used like any identifier that was defined on a .param control line. The `<value>` parts are supposed to be default values of the parameters. However, in the current version of , they are not used and each invocation of the subcircuit must supply the _exact_ number of actual parameters.

The syntax of a subcircuit call (invocation) is:
General form:

X<name> node node ... <identn> <ident>=<value> <ident>=<value> ...

Examples:

X1 input output myfilter rval=1k cval=1n

Here `<name>` is the symbolic name given to that instance of the subcircuit, `<identn>` is the name of a subcircuit defined beforehand. `node node ...` is the list of actual nodes where the subcircuit is connected. `<value>` is either a spice number or a brace expression { `<expr>` } . The sequence of `<value>` items on the `X` line must exactly match the number and the order of formal parameters of the subcircuit.

Subcircuit example with parameters:

```
* Param−example
.param amplitude= 1V
*
.subckt myfilter in out rval=100k   cval=100nF
Ra in p1    {2*rval}
Rb p1 out   {2*rval}
C1 p1 0     {2*cval}
Ca in p2    {cval}
Cb p2 out   {cval}
R1 p2 0     {rval}
.ends myfilter
*
X1 input output myfilter rval=1k cval=1n
V1 input 0 AC {amplitude}
.end
```

More text

2.8.4   Symbol scope

All subcircuit and model names are considered global and must be unique. The `.param` symbols that are defined outside of any ".subckt" ... ".ends" section are global. Inside such a section, the pertaining "`params:`" symbols and any .param assignments are considered local: they mask any global identical names, until the `.ends` line is encountered. You cannot reassign to a global number inside a `.subckt`, a local copy is created instead. Scope nesting works up to a level of 10. For example, if the main circuit calls A which has a formal parameter xx, A calls B which has a param. xx, and B calls C which also has a formal param. xx, there will be three versions of 'xx' in the symbol table but only the most local one - belonging to C - is visible.

2.8.5   Syntax of expressions

<expr> (optional parts within [...]):
    An expression may be one of:

    <atom> where <atom> is either a spice number or an identifier
    <unary−operator> <atom>
    <function−name> (<expr> [, <expr> ...])
    <atom> <binary−operator> <expr>
    (<expr>)

As expected, atoms, built-in function calls and stuff within parentheses are evaluated before the other operators. The operators are evaluated following a list of precedence close to the one of the C language. For equal precedence binary ops, evaluation goes left to right.

Operator	Alias	Precedence	Precedence
-		1	unary -
not	!	1	unary not
**	^	2	power
*		3	multiply
/		3	divide
mod	%	3	modulo
div	\	3	integer divide
+		4	add
-		4	subtract
==		5	equality
<>	!=	5	un-equal
<=		5	less or equal
>=		5	greater or equal
<		5	less than
>		5	greater than
and	&&	6	and
or	||	7	or

The result of logical operators is 1 or 0 , for True or False.

Builtin function	Notes
defined	returns 1 if symbol is defined, else 0
sqr	
sqrt	
sin	
cos	
exp	
ln	
arctan	
abs	
pwr	
min	
max	

The scaling suffixes (any decorative alphanumeric string may follow):

suffix	value
g	1e9
meg	1e6
k	1e3
m	1e-3
u	1e-6
n	1e-9
p	1e-12
f	1e-15

Note: there are intentional redundancies in expression syntax, e.g. x^y , x**y and pwr(x,y) all have

nearly the same result.

2.8.6   Reserved words

In addition to the above function names and to the verbose operators (not and or div mod), other words are reserved and cannot be used as parameter names: and, or, not, div, mod, if, else, end, while, macro, funct, defined, include, for, to, downto, is, var, sqr, sqrt, sin, cos, exp, ln, arctan, abs, pwr.

2.8.7   Alternative syntax

The & sign is tolerated to provide some "historical" parameter notation: & as the first character of a line is equivalent to: `.param`.

Inside a line, the notation `&(....)` is equivalent to `{....}`, and `&identifier` means the same thing as `{identifier}` .

Comments in the style of C++ line trailers (//) are detected and erased.

Warning: this is NOT possible in embedded .control parts of a source file, these lines are outside of this scope.

Now, there is some possible confusion in Spice because of multiple numerical expression features. The .param lines and the braces expressions (see next chapter 2.9) are evaluated in the front-end, that is, just after the subcircuit expansion. (Technically, the X lines are kept as comments in the expanded circuit so that the actual parameters can correctly be substituted). So, after the netlist expansion and before the internal data setup, all number attributes in the circuit are known constants. However, there are some circuit elements in Spice which accept arithmetic expressions that are NOT evaluated at this point, but only later during circuit analysis. These are the arbitrary current and voltage sources (B-sources, 6), as well as E- and G-sources and R-, L-, or C-devices. The syntactic difference is that "compile-time" expressions are within braces, but "run-time" expressions have no braces. To make things more complicated, the backend language JDML also accepts arithmetic/logic expressions that operate on its own scalar or vector data sets (18.1). Please see also chapt. 2.10.

It would be desirable to have the same expression syntax, operator and function set, and precedence rules, for the three contexts mentioned above. In the current Numparam implementation, that goal is not yet achieved...

2.9   .func

With this line a function may be defined. The syntax of its expression is equivalent to the expression syntax from the .param line (2.8.5).

General form:

```
. func <ident> { <expr> }
```

Examples:

```
. func  icos(x)  {cos(x) - 1}
. func  f(x,y)  {x*y}
```

Further explanation missing, e.g. sequence of functions, where to use.

2.10   Parameters, functions, expressions, and command scripts

In ngspice there are several ways to describe functional dependencies. In fact there are three independent function parsers, being active before, during, and after the simulation. So it might be due to have a few words on their interdepedencies.

2.10.1   Parameters

Parameters (chapt. 2.8.1) and functions, either defined within the `.param` statement or with the `.func` statement (chapt. 2.9) are evaluated **before** any simulation is started, that is during the setup of the input and the circuit. Therefore these statements may not contain any simulation output (voltage or current vectors), because it is simply not yet available. The syntax is described in chapt. 2.8.5. During the circuit setup all functions are evaluated, all parameters are replaced by their resulting numerical

values. Thus it will not be possible to get feedback from a later stage (during or after simulation) to change any of the parameters.

2.10.2   Nonlinear sources

During the simulation, the B source (chapt. 6) and their associated E and G sources, as well as some devices (R, C, L) may contain expressions. These expressions may contain parameters from above (evaluated immediately upon ngspice start up), numerical data, predined functions, but also node voltages and branch currents which are resulting from the simulation. The source or device values are continuously updated **during** the simulation. Therefore the sources are powerful tools to define non-linear behaviour, you may even create new 'devices' by yourself. Unfortunately the expression syntax (see chapt. 6.1) and the predfined functions may deviate from the ones for parameters listed in 2.8.1.

2.10.3   Control commands, Command scripts

Commands, as decribed in detail in chapt. 18.4, may be used interactively, but also as a command script enclosed in `.control ...  .endc` lines. The scripts may contain expressions (see chapt. 18.1). The expressions may work upon simulation output vectors (of node voltages, branch currents), as well as upon predefined or user defined vectors and variables, and are invoked **after** the simulation. Parameters from 2.8.1 are not allowed in these expressions. Again the expression syntax (see chapt. 18.1) will deviate from the one for parameters or B sources listed in 2.8.1 and 6.1.

If you want to use parameters from 2.8.1 inside your control script, you may apply a trick by defining a voltage source with the parameter as its value, and then have it available as a vector (e.g. after a transient simulation) with a then constant ouput (the parameter). A feedback from here back into parameters (2.10.1) is never possible. Also you cannot access non-linear sources of the preceeding simulation. However you may start a first simulation inside your control script, then evaluate its output using expressions, change some of the element or model parameters with the `alter` and `altermod` statements (see chapt. 18.4.3) and then automatically start a new simulation.

Expressions and scripting are powerful tools within ngspice, and we will enhance the examples given in chapt. 21 continuously to describe these features.

Chapter 3

Circuit Elements and Models

Data fields that are enclosed in less-than and greater-than signs ('$<\ >$') are optional. All indicated punctuation (parentheses, equal signs, etc.) is optional but indicate the presence of any delimiter. Further, future implementations may require the punctuation as stated. A consistent style adhering to the punctuation shown here makes the input easier to understand. With respect to branch voltages and currents, ngspice uniformly uses the associated reference convention (current flows in the direction of voltage drop).

3.1 General options and information

3.1.1 Simulating more devices in parallel

If you need to simulate more devices of the same kind in parallel, you can use the "m" (often called parallel multiplier) option which is available for all instances except transmission lines and sources (both independent and controlled). The parallel multiplier is implemented by multiplying the value of `m` the element's matrix stamp, thus it cannot be used to accurately simulate larger devices in integrated circuits. The netlist below show how to correctly use the parallel multiplier:

Multiple device example:

```
d1  2  0  mydiode m=10
d01 1  0  mydiode
d02 1  0  mydiode
d03 1  0  mydiode
d04 1  0  mydiode
d05 1  0  mydiode
d06 1  0  mydiode
d07 1  0  mydiode
d08 1  0  mydiode
d09 1  0  mydiode
d10 1  0  mydiode
...
```

The `d1` instance connected between nodes 2 and 0 is equivalent to the parallel `d01-d10` connected between 1 and 0.

3.1.2 Technology scaling

Still to be implemented and written.

3.1.3 Model binning

Still to be implemented and written.

3.1.4 Transistors and Diodes

The area factor "m" (often called parallel multiplier) used on the diode, BJT, JFET, and MESFET devices determines the number of equivalent parallel devices of a specified model. The affected parameters are

marked with an asterisk under the heading "area" in the model descriptions (see the various chapters on models below). Several geometric factors associated with the channel and the drain and source diffusions can be specified on the MOSFET device line.

Two different forms of initial conditions may be specified for some devices. The first form is included to improve the dc convergence for circuits that contain more than one stable state. If a device is specified `OFF`, the dc operating point is determined with the terminal voltages for that device set to zero. After convergence is obtained, the program continues to iterate to obtain the exact value for the terminal voltages. If a circuit has more than one dc stable state, the `OFF` option can be used to 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) but more iterations are required since the program must independently converge to two separate solutions.

The `.NODESET` control line (see chapt. 16.2.1) serves a similar purpose as the `OFF` option. The `.NODESET` option is easier to apply and is the preferred means to aid convergence. The second form of initial conditions are specified for use with the transient analysis. These are true "initial conditions" as opposed to the convergence aids above. See the description of the `.IC` control line (chapt. 16.2.2) and the `.TRAN` control line (chapt. 16.3.9) for a detailed explanation of initial conditions.

3.2   Elementary Devices

3.2.1   Resistors

General form:

```
RXXXXXXX n+ n- value <ac=val> <m=val> <scale=val> <temp=val>
+ <dtemp=val> <noisy=0|1>
```

Examples:

```
R1 1 2 100
RC1 12 17 1K
R2 5 7 1K ac=2K
RL 1 4 2K m=2
```

Ngspice has a fairly complex model for resistors. It can simulate both discrete and semiconductor resistors. Semiconductor resistors in ngspice means: resistors described by geometrical parameters. So, do not expect detailed modeling of semiconductor effects.

`n+` and `n-` are the two element nodes, `value` is the resistance (in ohms) and may be positive or negative[1] but not zero.

Simulating small valued resistors: If you need to simulate very small resistors (0.001 Ohm or less), you should use CCVS (transresistance), it is less efficient but improves overall numerical accuracy. Think about that a small resistance is a large conductance.

Ngspice can assign a resistor instance a different value for AC analysis, specified using the `ac` keyword. This value must not be zero as described above. The AC resistance is used in AC analysis only (not Pole-Zero nor noise). If you do not specify the `ac` parameter, it is defaulted to `value`. If you want to simulate temperature dependence of a resistor, you need to specify its temperature coefficients, using a `.model` line, like in the example below:

Example:

```
RE1 1 2 800 newres dtemp=5
```

```
.MODEL newres R tc1=0.001
```

Instance temperature is useful even if resistance does not varies with it, since the thermal noise generated by a resistor depends on its absolute temperature. Resistors in ngspice generates two different noises: thermal and flicker. While thermal noise is always generated in the resistor, to add a flicker noise[2] source you have to add a `.model` card defining the flicker noise parameters. It is possible to simulate resistors that do not generate any kind of noise using the `noisy` keyword and assigning zero to it, as in the following example:

---

[1] A negative resistor modeling an active element can cause convergence problems, please avoid it.
[2] Flicker noise can be used to model carbon resistors.

Example:

Rmd 134 57 1.5k noisy=0

Ngspice calculates the nominal resistance as described below:

$$
\begin{aligned}
R_{nom} &= \frac{\text{VALUE}*\text{scale}}{m} \\
R_{acnom} &= \frac{\text{ac}*\text{scale}}{m}
\end{aligned}
\tag{3.1}
$$

If you are interested in temperature effects or noise equations, read the next section on semiconductor resistors.

3.2.2 Semiconductor Resistors

General form:

RXXXXXXX n+ n− <value> <mname> <l=length> <w=width> <temp=val>
+ <dtemp=val> m=<val> <ac=val> <scale=val> <noisy = 0|1>

Examples:

RLOAD 2 10 10K
RMOD 3 7 RMODEL L=10u W=1u

This is the more general form of the resistor presented before (3.2.1) and allows the modeling of temperature effects and for the calculation of the actual resistance value from strictly geometric information and the specifications of the process. If `value` is specified, it overrides the geometric information and defines the resistance. If `mname` is specified, then the resistance may be calculated from the process information in the model `mname` and the given `length` and `width`. If `value` is not specified, then `mname` and `length` must be specified. If `width` is not specified, then it is taken from the default width given in the model.

The (optional) `temp` value is the temperature at which this device is to operate, and overrides the temperature specification on the `.option` control line and the value specified in `dtemp`.

3.2.3 Semiconductor Resistor Model (R)

The resistor model consists of process-related device data that allow the resistance to be calculated from geometric information and to be corrected for temperature. The parameters available are:

Name	Parameter	Units	Default	Example
TC1	first order temperature coeff.	$\Omega/^\circ C$	0.0	-
TC2	second order temperature coeff.	$\Omega/^\circ C^2$	0.0	-
RSH	sheet resistance	$\Omega/\square$	-	50
DEFW	default width	m	1e-6	2e-6
NARROW	narrowing due to side etching	m	0.0	1e-7
SHORT	shortening due to side etching	m	0.0	1e-7
TNOM	parameter measurement temperature	$^\circ C$	27	50
KF	flicker noise coefficient		0.0	1e-25
AF	flicker noise exponent		0.0	1.0

The sheet resistance is used with the narrowing parameter and `l` and `w` from the resistor device to determine the nominal resistance by the formula:

$$
R_{nom} = \text{rsh}\frac{l - \text{SHORT}}{w - \text{NARROW}}
\tag{3.2}
$$

`DEFW` is used to supply a default value for `w` if one is not specified for the device. If either `rsh` or `l` is not specified, then the standard default resistance value of 1 kOhm is used. `TNOM` is used to override the circuit-wide value given on the `.options` control line where the parameters of this model have been measured at a different temperature. After the nominal resistance is calculated, it is adjusted for temperature by the formula:

$$
R(T) = R(\text{TNOM})\left(1 + TC_1(T - \text{TNOM}) + TC_2(T - \text{TNOM})^2\right)
\tag{3.3}
$$

where $R(\text{TNOM}) = R_{nom}|R_{acnom}$. In the above formula, "T" represents the instance temperature, which can be explicitly set using the `temp` keyword or calculated using the circuit temperature and `dtemp`,

if present. If both `temp` and `dtemp` are specified, the latter is ignored. Ngspice improves spice's resistors noise model, adding flicker noise ($^1/_f$) to it and the `noisy` keyword to simulate noiseless resistors. The thermal noise in resistors is modeled according to the equation:

$$\bar{i_R^2} = \frac{4kT}{R}\Delta f \tag{3.4}$$

where "k" is the Boltzmann's constant, and "T" the instance temperature.

Flicker noise model is:

$$i_{Rfn}^{2^-} = \frac{\text{KF}I_R^{\text{AF}}}{f}\Delta f \tag{3.5}$$

A small list of sheet resistances (in $\Omega/\square$) for conductors is shown below. The table represents typical values for MOS processes in the 0.5 - 1 um

range. The table is taken from: *N. Weste, K. Eshraghian - Principles of CMOS VLSI Design 2nd Edition, Addison Wesley.*

Material	Min.	Typ.	Max.
Intermetal (metal1 - metal2)	0.005	0.007	0.1
Top-metal (metal3)	0.003	0.004	0.05
Polysilicon (poly)	15	20	30
Silicide	2	3	6
Diffusion (n+, p+)	10	25	100
Silicided diffusion	2	4	10
n-well	1000	2000	5000

3.2.4   Resistors, dependent on expressions

General form:

```
RXXXXXXX n+ n− R = ' expression '
RXXXXXXX n+ n− ' expression '
```

Examples:

```
R1  rr  0  r  =  'V( rr ) < {Vt} ? {R0} : {2∗R0}'
```

Expression may be an equation or an expression containing node voltages or branch currents (in the form of i(vm)) and any other terms as given for the B source and described in chapter 6.1. It may contain parameters (2.8.1). An example file is given below.

Example input file for non-linear resistor:

```
Non−linear  resistor
.param  R0=1k  Vi=1  Vt=0.5
∗ resistor  depending  on  control  voltage  V( rr )
R1  rr  0  r  =  'V( rr ) < {Vt} ? {R0} : {2∗R0}'
∗ control  voltage
V1  rr  0 PWL(0  0  100u {Vi})
.control
set  noaskquit
tran  100n  100u  uic
plot  i (V1)
.endc
.end
```

3.2.5 Capacitors

General form:

CXXXXXXX n+ n− <value> <mname> <m=val> <scale=val> <temp=val>
+ <dtemp=val> <ic=init_condition>

Examples:

CBYP 13 0 1UF
COSC 17 23 10U IC=3V

Ngspice provides a detailed model for capacitors. Capacitors in the netlist can be specified giving their capacitance or their geometrical and physical characteristics. Following the original spice3 "convention", capacitors specified by their geometrical or physical characteristics are called "semiconductor capacitors" and are described in the next section.

In this first form n+ and n- are the positive and negative element nodes, respectively and value is the capacitance in Farads.

Capacitance can be specified in the instance line as in the examples above or in a .model line, as in the example below:

    @C1 15 5 cstd
    C2 2 7 cstd
    .model cstd C cap=3n

Both capacitors have a capacitance of 3nF.
If you want to simulate temperature dependence of a capacitor, you need to specify its temperature coefficients, using a @command{.model} line, like in the example below:

    CEB 1 2 1u cap1 dtemp=5
    .MODEL cap1 C tc1=0.001

The (optional) initial condition is the initial (time zero) value of capacitor voltage (in Volts). Note that the initial conditions (if any) apply 'only' if the uic option is specified on the .tran control line.

Ngspice calculates the nominal capacitance as described below:

$$C_{nom} = \text{value} * \text{scale} * m \tag{3.6}$$

3.2.6 Semiconductor Capacitors

General form:

CXXXXXXX n+ n− <value> <mname> <l=length> <w=width> <m=val>
+ <scale=val> <temp=val> <dtemp=val> <ic=init_condition>

Examples:

CLOAD 2 10 10P
CMOD 3 7 CMODEL L=10u W=1u

This is the more general form of the Capacitor presented in section (3.2.5), and allows for the calculation of the actual capacitance value from strictly geometric information and the specifications of the process. If value is specified, it defines the capacitance and both process and geometrical information are discarded. If value is not specified, the capacitance is calculated from information contained model mname and the given length and width (l, w keywords, respectively).

It is possible to specify mname only, without geometrical dimensions and set the capacitance in the .model line (3.2.5).

3.2.7 Semiconductor Capacitor Model (C)

The capacitor model contains process information that may be used to compute the capacitance from strictly geometric information.

Name	Parameter	Units	Default	Example
CAP	model capacitance	F	0.0	1e-6
CJ	junction bottom capacitance	F/m^2	-	5e-5
CJSW	junction sidewall capacitance	F/m	-	2e-11
DEFW	default device width	m	1e-6	2e-6
DEFL	default device length	m	0.0	1e-6
NARROW	narrowing due to side etching	m	0.0	1e-7
SHORT	shortening due to side etching	m	0.0	1e-7
TC1	first order temperature coeff.	$F/°C$	0.0	0.001
TC2	second order temperature coeff.	$F/°C^2$	0.0	0.0001
TNOM	parameter measurement temperature	$°C$	27	50
DI	relative dielectric constant	F/m	-	1
THICK	insulator thickness	m	0.0	1e-9

The capacitor has a capacitance computed as:

If `value` is specified on the instance line then

$$C_{nom} = \text{value} * \text{scale} * m \tag{3.7}$$

If model capacitance is specified then

$$C_{nom} = \text{CAP} * \text{scale} * m \tag{3.8}$$

If neither `value` nor `CAP` are specified, then geometrical and physical parameters are take into account:

$$C_0 = \text{CJ}(l - \text{SHORT})(w - \text{NARROW}) + 2\text{CJSW}(l - \text{SHORT} + w - \text{NARROW}) \tag{3.9}$$

`CJ` can be explicitly given on the `.model` line or calculated by physical parameters. When `CJ` is not given, is calculated as:

If `THICK` is not zero:

$$\begin{aligned} \text{CJ} &= \frac{\text{DI}*\epsilon_0}{\text{THICK}} \quad \text{if DI is specified,} \\ \text{CJ} &= \frac{\epsilon_{SiO_2}}{\text{THICK}} \qquad \text{otherwise.} \end{aligned} \tag{3.10}$$

If the relative dielectric constant is not specified the one for SiO2 is used. The values of the constants are: $\epsilon_0 = 8.854214871e - 12\frac{F}{m}$ and $\epsilon_{SiO_2} = 3.4531479969e - 11\frac{F}{m}$. The nominal capacitance is then computed as:

$$C_{nom} = C_0 * \text{scale} * m \tag{3.11}$$

After the nominal capacitance is calculated, it is adjusted for temperature by the formula:

$$C(T) = C(\text{TNOM})\left(1 + TC_1(T - \text{TNOM}) + TC_2(T - \text{TNOM})^2\right) \tag{3.12}$$

where $C(\text{TNOM}) = C_{nom}$.

In the above formula, "T" represents the instance temperature, which can be explicitly set using the `temp` keyword or calculated using the circuit temperature and `dtemp`, if present.

3.2.8   Capacitors, dependent on expressions

General form:

```
CXXXXXXX n+ n− C = 'expression'
CXXXXXXX n+ n− 'expression'
```

Examples:

```
C1  cc  0  c = 'V(cc) < {Vt} ? {C1} : {Ch}'
```

Expression may be an equation or an expression containing node voltages or branch currents (in the form of i(vm)) and any other terms as given for the B source and described in chapter 6.1. It may contain parameters (2.8.1).

Example input file:

```
Dependent  Capacitor
.param  Cl=5n  Ch=1n  Vt=1m  Il=100n
.ic  v(cc) = 0       v(cc2) = 0
* capacitor  depending  on  control  voltage  V(cc)
C1  cc  0  c = 'V(cc) < {Vt} ? {Cl} : {Ch}'
*C1  cc  0  c ={Ch}
I1  0  1  {Il}
Exxx    n1-copy  n2   n2  cc2   1
Cxxx    n1-copy  n2   1
Bxxx    cc2  n2   I = '(V(cc2) < {Vt} ? {Cl} : {Ch})' * i(Exxx)
I2  n2  22  {Il}
vn2  n2  0 DC 0
* measure  charge  by  integrating  current
aint1 %id(1  cc)  2  time_count
aint2 %id(22  cc2)  3  time_count
.model  time_count  int(in_offset=0.0  gain=1.0
+ out_lower_limit=-1e12  out_upper_limit=1e12
+ limit_range=1e-9  out_ic=0.0)
.control
set  noaskquit
tran  100n  100u
plot  v(2)
plot  v(cc)  v(cc2)
.endc
.end
```

3.2.9 Inductors

General form:

LYYYYYYY n+ n− <value> <mname> <nt=val> <m=val> <scale=val> <temp=val>
+ <dtemp=val> <ic=init_condition>

Examples:

LLINK 42 69 1UH
LSHUNT 23 51 10U IC=15.7MA

The inductor device implemented into ngspice has many enhancements over the original one.n+ and n− are the positive and negative element nodes, respectively. value is the inductance in Henries. Inductance can be specified in the instance line as in the examples above or in a .model line, as in the example below:

```
L1  15  5  indmod1
L2  2  7  indmod1
.model  indmod1  L  ind=3n
```

Both inductors have an inductance of 3nH.
The nt is used in conjunction with a .model line, and is used to specify the number of turns of the inductor. If you want to simulate temperature dependence of an inductor, you need to specify its temperature coefficients, using a .model line, like in the example below:

```
Lload  1  2  1u  ind1  dtemp=5
.MODEL  ind1  L  tc1=0.001
```

The (optional) initial condition is the initial (time zero) value of inductor current (in Amps) that flows from n+, through the inductor, to n−. Note that the initial conditions (if any) apply only if the UIC option is specified on the .tran analysis line.

Ngspice calculates the nominal inductance as described below:

$$L_{nom} = \frac{\text{value} * \text{scale}}{m} \tag{3.13}$$

3.2.10   Inductor model

The inductor model contains physical and geometrical information that may be used to compute the inductance of some common topologies like solenoids and toroids, wound in air or other material with constant magnetic permeability.

Name	Parameter	Units	Default	Example
IND	model inductance	H	0.0	1e-3
CSECT	cross section	m^2	0.0	1e-3
LENGTH	length	m	0.0	1e-2
TC1	first order temperature coeff.	$H/^{\circ}C$	0.0	0.001
TC2	second order temperature coeff.	$H/^{\circ}C^2$	0.0	0.0001
TNOM	parameter measurement temperature	$^{\circ}C$	27	50
NT	number of turns	-	0.0	10
MU	relative magnetic permeability	H/m	0.0	-

The inductor has an inductance computed as:

If `value` is specified on the instance line then

$$L_{nom} = \frac{\text{value} * \text{scale}}{m} \tag{3.14}$$

If model inductance is specified then

$$L_{nom} = \frac{\text{IND} * \text{scale}}{m} \tag{3.15}$$

If neither `value` nor IND are specified, then geometrical and physical parameters are take into account. In the following formulas

NT refers to both instance and model parameter (instance parameter overrides model parameter):

If LENGTH is not zero:

$$\begin{cases} L_{nom} = \frac{\text{MU} * \mu_0 * \text{NT}^2 * \text{CSECT}}{\text{LENGTH}} & \text{if MU is specified,} \\ L_{nom} = \frac{\mu_0 * \text{NT}^2 * \text{CSECT}}{\text{LENGTH}} & \text{otherwise.} \end{cases} \tag{3.16}$$

with:$\mu_0 = 1.25663706143592e - 6\frac{H}{m}$. After the nominal inductance is calculated, it is adjusted for temperature by the formula:

$$L(T) = L(\text{TNOM})\Big(1 + TC_1(T - \text{TNOM}) + TC_2(T - \text{TNOM})^2\Big) \tag{3.17}$$

where $L(\text{TNOM}) = L_{nom}$. In the above formula, "T" represents the instance temperature, which can be explicitly using the `temp` keyword or calculated using the circuit temperature and `dtemp`, if present.

3.2.11   Coupled (Mutual) Inductors

General form:

KXXXXXXX LYYYYYYY LZZZZZZZ `value`

Examples:

K43 LAA LBB `0.999`
KXFRMR L1 L2 `0.87`

LYYYYYYY and LZZZZZZZ are the names of the two coupled inductors, and `value` is the coefficient of coupling, K, which must be greater than 0 and less than or equal to 1. Using the "dot" convention, place a "dot" on the first node of each inductor.

3.2.12 Inductors, dependent on expressions

General form:

```
LXXXXXXX n+ n− L = 'expression'
LXXXXXXX n+ n− 'expression'
```

Examples:

```
L1 l2 lll L = 'i(Vm) < {It} ? {Ll} : {Lh}'
```

Expression may be an equation or an expression containing node voltages or branch currents (in the form of i(vm)) and any other terms as given for the B source and described in chapter 6.1. It may contain parameters (2.8.1).

Example input file:

```
Variable inductor
.param Ll=0.5m Lh=5m It=50u Vi=2m
.ic v(int21) = 0

* variable inductor depending on control current i(Vm)
L1 l2 lll L = 'i(Vm) < {It} ? {Ll} : {Lh}'
* measure current through inductor
vm lll 0 dc 0
* voltage on inductor
V1 l2 0 {Vi}

* fixed inductor
L3 33 331 {Ll}
* measure current through inductor
vm33 331 0 dc 0
* voltage on inductor
V3 33 0 {Vi}

* non linear inductor (discrete setup)
F21 int21 0 B21 −1
L21 int21 0 1
B21 n1 n2 V = '(i(Vm21) < {It} ? {Ll} : {Lh})' * v(int21)
* measure current through inductor
vm21 n2 0 dc 0
V21 n1 0 {Vi}

.control
set noaskquit
tran 1u 100u uic
plot i(Vm) i(vm33)
plot i(vm21) i(vm33)
plot i(vm)−i(vm21)
.endc
.end
```

3.2.13 Capacitor or inductor with initial conditions

The simulator supports the specification of voltage and current initial conditions on capacitor and inductor models, respectively. **These models are not the standard ones supplied with SPICE3, but are in fact code models which can be substituted for the SPICE models when realistic initial conditions are required**. For details please refer to chapt. 13. A XSPICE deck example using these models is shown below:

```
*
* This circuit contains a capacitor and an inductor with
```

```
* initial conditions on them. Each of the components
* has a parallel resistor so that an exponential decay
* of the initial condition occurs with a time constant of
* 1 second.
*
a1 1 0 cap
.model cap capacitor (c=1000uf ic=1)
r1 1 0 1k
*
a2 2 0 ind
.model ind inductor (l=1H ic=1)
r2 2 0 1.0
*
.control
tran 0.01 3
plot v(1) v(2)
.endc
.end
```

3.2.14   Switches

Two types of switches are available: a voltage controlled switch (type SXXXXXX, model SW) and a current controlled switch (type WXXXXXXX, model CSW). A switching hysteresis may be defined, as well as on- and off-resistances ($0 < R < \infty$).

General form:

```
SXXXXXXX N+ N- NC+ NC- MODEL <ON><OFF>
WYYYYYYY N+ N- VNAM MODEL <ON><OFF>
```

Examples:

```
s1 1 2 3 4 switch1 ON
s2 5 6 3 0 sm2 off
Switch1 1 2 10 0 smodel1
w1 1 2 vclock switchmod1
W2 3 0 vramp sm1 ON
wreset 5 6 vclck lossyswitch OFF
```

Nodes 1 and 2 are the nodes between which the switch terminals are connected. The model name is mandatory while the initial conditions are optional. For the voltage controlled switch, nodes 3 and 4 are the positive and negative controlling nodes respectively. For the current controlled switch, the controlling current is that through the specified voltage source. The direction of positive controlling current flow is from the positive node, through the source, to the negative node.

The instance parameters ON or OFF are required, when the controlling voltage (current) starts inside the range of the hysteresis loop (different outputs during forward vs. backward voltage or current ramp). Then ON or OFF detrmine the initial state of the switch.

3.2.15   Switch Model (SW/CSW)

The switch model allows an almost ideal switch to be described in ngspice. The switch is not quite ideal, in that the resistance can not change from 0 to infinity, but must always have a finite positive value. By proper selection of the on and off resistances, they can be effectively zero and infinity in comparison to other circuit elements. The parameters available are:

Name	Parameter	Units	Default	Switch model
VT	threshold voltage	V	0.0	SW
IT	threshold current	A	0.0	CSW
VH	hysteresis voltage	V	0.0	SW
IH	hysteresis current	A	0.0	CSW
RON	on resistance	Ω	1.0	SW,CSW
ROFF	off resistance	Ω	$1/GMIN$[3]	SW,CSW

The use of an ideal element that is highly nonlinear such as a switch can cause large discontinuities to occur in the circuit node voltages. A rapid change such as that associated with a switch changing state can cause numerical round-off or tolerance problems leading to erroneous results or time step difficulties. The user of switches can improve the situation by taking the following steps:

- First, it is wise to set ideal switch impedances just high or low enough to be negligible with respect to other circuit elements. Using switch impedances that are close to "ideal" in all cases aggravates the problem of discontinuities mentioned above. Of course, when modeling real devices such as MOSFETS, the on resistance should be adjusted to a realistic level depending on the size of the device being modeled.

- If a wide range of ON to OFF resistance must be used in the switches (ROFF/RON >1e+12), then the tolerance on errors allowed during transient analysis should be decreased by using the `.OPTIONS` control line and specifying `TRTOL` to be less than the default value of 7.0.

- When switches are placed around capacitors, then the option `CHGTOL` should also be reduced. Suggested values for these two options are 1.0 and 1e-16 respectively. These changes inform ngspice to be more careful around the switch points so that no errors are made due to the rapid change in the circuit.

Example input file:

```
Switch test
.tran 2us 5ms
*switch control voltage
v1 1 0 DC 0.0 PWL(0 0 2e-3 2 4e-3 0)
*switch control voltage starting inside hysteresis window
*please note influence of instance parameters ON, OFF
v2 2 0 DC 0.0 PWL(0 0.9 2e-3 2 4e-3 0.4)
*switch control current
i3 3 0 DC 0.0 PWL(0 0 2e-3 2m 4e-3 0) $ <-- switch control current
*load voltage
v4 4 0 DC 2.0
*input load for current source i3
r3 3 33 10k
vm3 0 33 dc 0 $ <-- measure the current
* ouput load resistors
r10 4 10 10k
r20 4 20 10k
r30 4 30 10k
r40 4 40 10k
*
s1 10 0 1 0 switch1 OFF
s2 20 0 2 0 switch1 OFF
s3 30 0 2 0 switch1 ON
.model switch1 sw vt=1 vh=0.2 ron=1 roff=10k
*
w1 40 0 vm3 wswitch1 off
.model wswitch1 csw  it=1m ih=0.2m ron=1 roff=10k
*
.control
run
plot v(1) v(10)
plot v(10) vs v(1) $ <-- get hysteresis loop
plot v(2) v(20) $ <-- different initial values
plot v(20) vs v(2) $ <-- get hysteresis loop
plot v(2) v(30) $ <-- different initial values
plot v(30) vs v(2) $ <-- get hysteresis loop
plot v(40) vs vm3#branch $ <-- current controlled switch hysteresis
.endc
.end
```

Chapter 4

Voltage and Current Sources

4.1 Arbitrary Phase Sources

The XSPICE simulator supports arbitrary phase independent sources that output at TIME=0.0 a value corresponding to some specified phase shift. Other versions of SPICE use the TD (delay time) parameter to set phase-shifted sources to their time-zero value until the delay time has elapsed. The XSPICE phase parameter is specified in degrees and is included after the SPICE3 parameters normally used to specify an independent source. Partial XSPICE deck examples of usage for pulse and sine waveforms are shown below:

```
* Phase shift is specified after Berkeley defined parameters
* on the independent source cards. Phase shift for both of the
* following is specified as +45 degrees
*
v1 1 0 0.0 sin(0 1 1k 0 0 45.0)
r1 1 0 1k
*
v2 2 0 0.0 pulse(-1 1 0 1e-5 1e-5 5e-4 1e-3 45.0)
r2 2 0 1k
*
```

4.2 Independent Sources

General form:

VXXXXXXX N+ N− <<DC> DC/TRAN VALUE> <AC <ACMAG <ACPHASE>>>
+ <DISTOF1 <F1MAG <F1PHASE>>> <DISTOF2 <F2MAG <F2PHASE>>>
IYYYYYYY N+ N− <<DC> DC/TRAN VALUE> <AC <ACMAG <ACPHASE>>>
+ <DISTOF1 <F1MAG <F1PHASE>>> <DISTOF2 <F2MAG <F2PHASE>>>

Examples:

VCC 10 0 DC 6
VIN 13 2 0.001 AC 1 SIN(0 1 1MEG)
ISRC 23 21 AC 0.333 45.0 SFFM(0 1 10K 5 1K)
VMEAS 12 9
VCARRIER 1 0 DISTOF1 0.1 −90.0
VMODULATOR 2 0 DISTOF2 0.01
IIN1 1 5 AC 1 DISTOF1 DISTOF2 0.001

n+ and n- are the positive and negative nodes, respectively. Note that voltage sources need not be grounded. Positive current is assumed to flow from the positive node, through the source, to the negative node. A current source of positive value forces current to flow out of the n+ node, through the source, and into the n- node. Voltage sources, in addition to being used for circuit excitation, are the "ammeters" for ngspice, that is, zero valued voltage sources may be inserted into the circuit for the purpose of measuring current. They of course have no effect on circuit operation since they represent short-circuits.

DC/TRAN is the dc and transient analysis value of the source. If the source value is zero both for dc and transient analyses, this value may be omitted. If the source value is time-invariant (e.g., a power supply), then the value may optionally be preceded by the letters DC.

ACMAG is the ac magnitude and ACPHASE is the ac phase. The source is set to this value in the ac analysis. If ACMAG is omitted following the keyword AC, a value of unity is assumed. If ACPHASE is omitted, a value of zero is assumed. If the source is not an ac small-signal input, the keyword AC and the ac values are omitted.

DISTOF1 and DISTOF2 are the keywords that specify that the independent source has distortion inputs at the frequencies F1 and F2 respectively (see the description of the .DISTO control line). The keywords may be followed by an optional magnitude and phase. The default values of the magnitude and phase are 1.0 and 0.0 respectively.

Any independent source can be assigned a time-dependent value for transient analysis. If a source is assigned a time-dependent value, the time-zero value is used for dc analysis. There are five independent source functions:

- pulse,

- exponential,

- sinusoidal,

- piece-wise linear,

- and single-frequency FM.

If parameters other than source values are omitted or set to zero, the default values shown are assumed. (TSTEP is the printing increment and TSTOP is the final time (see the .TRAN control line for explanation)).

4.2.1   Pulse

General form:

PULSE(V1 V2 TD TR TF PW PER)

Examples:

VIN 3 0 PULSE(-1 1 2NS 2NS 2NS 50NS 100NS)

Name	Parameter	Default Value	Units
V1	Initial value	-	V, A
V2	Pulsed value	-	V, A
TD	Delay time	0.0	sec
TR	Rise time	TSTEP	sec
TF	Fall time	TSTEP	sec
PW	Pulse width	TSTOP	sec
PER	Period	TSTOP	sec

A single pulse so specified is described by the following table:

Time	Value
0	V1
TD	V1
TD+TR	V2
TD+TR+PW	V2
TD+TR+PW+TF	V1
TSTOP	V1

Intermediate points are determined by linear interpolation.

4.2.2 Sinusoidal

General form:

SIN (VO VA FREQ TD THETA)

Examples:

VIN 3 0 SIN (0 1 100MEG 1NS 1E10)

Name	Parameter	Default Value	Units
VO	Offset	-	V, A
VA	Amplitude	-	V, A
FREQ	Frequency	$1/TSTOP$	Hz
TD	Delay	0.0	sec
THETA	Damping factor	0.0	$1/sec$

The shape of the waveform is described by the following formula:

$$V(t) = \begin{cases} V0 & \text{if } 0 \leq t < TD \\ V0 + VA\mathrm{e}^{-(t-TD)THETA}\sin(2\pi FREQ(t-TD)) & \text{if } TD \leq t < TSTOP \end{cases} \tag{4.1}$$

4.2.3 Exponential

General Form:

EXP (V1 V2 TD1 TAU1 TD2 TAU2)

Examples:

VIN 3 0 EXP(−4 −1 2NS 30NS 60NS 40NS)

Name	Parameter	Default Value	Units
V1	Initial value	-	V, A
V2	pulsed value	-	V, A
TD1	rise delay time	0.0	sec
TAU1	rise time constant	TSTEP	sec
TD2	fall delay time	TD1+TSTEP	sec
TAU2	fall time constant	TSTEP	sec

The shape of the waveform is described by the following formula:
Let $V21 = V2 - V1$ $V12 = V1 - V2$:

$$V(t) = \begin{cases} V1 & \text{if } 0 \leq t < TD1, \\ V1 + V21\left(1 - e^{-\frac{(t-TD1)}{TAU1}}\right) & \text{if } TD1 \leq t < TD2, \\ V1 + V21\left(1 - e^{-\frac{(t-TD1)}{TAU1}}\right) + V12\left(1 - e^{-\frac{(t-TD2)}{TAU2}}\right) & \text{if } TD2 \leq t < TSTOP. \end{cases} \tag{4.2}$$

4.2.4 Piece-Wise Linear

General Form:

PWL(T1 V1 <T2 V2 T3 V3 T4 V4 ... >) <r=value> <td=value>

Examples:

VCLOCK 7 5 PWL(0 −7 10NS −7 11NS −3 17NS −3 18NS −7 50NS −7) r=0 td=15NS

Each pair of values (T_i, V_i) specifies that the value of the source is V_i (in Volts or Amps) at time = T_i. The value of the source at intermediate values of time is determined by using linear interpolation on the input values. The parameter r determines a repaet time point. If r is not given, the whole sequence of values (T_i, V_i) is issued once, then the output stays at its final value. If $r = 0$, the whole sequence from $time = 0$ to $time = Tn$ is repeated forever. If $r = 10ns$, the sequence between 10ns and 50ns is repeated forever. the r value has to be one of the time points T1 to Tn of the PWL sequence. If td is given, the whole PWL sequence is delayed by a delay time $time = td$. The current source still needs to be patched, td and r are not yet available.

4.2.5   Single-Frequency FM

General Form:

SFFM(VO VA FC MDI FS)

Examples:

V1 12 0 SFFM(0 1M 20K 5 1K)

Name	Parameter	Default value	Units
VO	Offset	-	V, A
VA	Amplitude	-	V, A
FC	Carrier frequency	$1/_{TSTOP}$	Hz
MDI	Modulation index	-	
FS	Signal frequency	$1/_{TSTOP}$	Hz

The shape of the waveform is described by the following equation:

$$V(t) = V_O + V_A \sin\left(2\pi FC t + MDI \sin\left(2\pi FS t\right)\right) \tag{4.3}$$

Chapter 5

Linear Dependent Sources

Ngspice allows circuits to contain linear dependent sources characterized by any of the four equations

| $i = gv$ | $v = ev$ | $i = fi$ | $v = hi$ |

where g, e, f, and h are constants representing transconductance, voltage gain, current gain, and transresistance, respectively. Non-linear dependent sources for voltages or currents (B, E, G) are described in chapter 6.

5.1 Linear Voltage-Controlled Current Sources (VCCS)

General form:

GXXXXXXX N+ N− NC+ NC− VALUE

Examples:

G1 2 0 5 0 0.1MMHO

n+ and n- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative
node. nc+ and nc- are the positive and negative controlling nodes, respectively. value is the transconductance (in mhos).

5.2 Linear Voltage-Controlled Voltage Sources (VCVS)

General form:

EXXXXXXX N+ N− NC+ NC− VALUE

Examples:

E1 2 3 14 1 2.0

n+ is the positive node, and n- is the negative node. nc+ and nc- are the positive and negative controlling nodes, respectively. value is the
voltage gain.

5.3 Linear Current-Controlled Current Sources (CCCS)

General form:

FXXXXXXX N+ N− VNAM VALUE

Examples:

F1 13 5 VSENS 5

n+ and n- are the positive and negative nodes, respectively. Current flow is from the positive node, through the source, to the negative node. vnam is the name of a voltage source through which the

Dependent Polynomial Sources	
Source Type	Instance Card
POLYNOMIAL VCVS	EXXXXXXX N+ N- (POLY (ND)) NC1+ NC1- P0 (P1...)
POLYNOMIAL VCCS	GXXXXXXX N+ N- (POLY (ND)) NC1+ NC1- P0 (P1...)
POLYNOMIAL CCCS	FXXXXXXX N+ N- (POLY (ND)) VNAM1 !VNAM2...? P0 (P1...)
POLYNOMIAL CCVS	HXXXXXXX N+ N- (POLY (ND)) VNAM1 !VNAM2...? P0 (P1...)

Table 5.1: Dependent Polynomial Sources

controlling current flows. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of `vnam`. `value` is the current gain.

5.4   Linear Current-Controlled Voltage Sources (CCVS)

General form:

HXXXXXXX n+ n− vnam value

Examples:

HX 5 17 VZ 0.5K

n+ and n- are the positive and negative nodes, respectively. `vnam` is the name of a voltage source through which the controlling current flows. The direction of positive controlling current flow is from the positive node, through the source, to the negative node of `vnam`. `value` is the transresistance (in ohms).

5.5   Polynomial Source Compatibility

Dependent polynomial sources available in SPICE2G6 are fully supported in ngspice using the XSPICE extension. Dependent polynomial sources are not supported in SPICE3 but were reinstated in XSPICE to allow existing third party models to be incorporated readily into XSPICE. The form used to specify these sources is shown in Table 5.1.

Chapter 6

Non-linear Dependent Sources

6.1   B source (ASRC)

General form:

BXXXXXXX n+ n− <i=expr> <v=expr>

Examples:

```
B1 0 1 I=cos(v(1))+sin(v(2))
B2 0 1 V=ln(cos(log(v(1,2)^2)))−v(3)^4+v(2)^v(1)
B3 3 4 I=17
B4 3 4 V=exp(pi^i(vdd))
B5 2 0 V = V(1) < {Vlow} ? {Vlow} : V(1) > {Vhigh} ? {Vhigh} : V(1)
```

   n+ is the positive node, and n- is the negative node. The values of the V and I parameters determine the voltages and currents across and through the device, respectively. If I is given then the device is a current source, and if V is given the device is a voltage source. One and only one of these parameters must be given.

   The small-signal AC behavior of the nonlinear source is a linear dependent source (or sources) with a proportionality constant equal to the derivative (or derivatives) of the source at the DC operating point. The expressions given for V and I may be any function of voltages and currents through voltage sources in the system. In addition, the variables 'time' and 'temper' are available in a transient analysis, reflecting the actual simulation time and circuit temperature. The variable 'hertz' is available in an AC analysis. 'time' is zero in the AC analysis, 'hertz' is zero during transient analysis. Using the variable 'hertz' may cost some CPU time if you have a large circuit, because for each frequency the operating point has to be determined before calculating the AC response.

   The following functions of a single real variable are defined:

Trigonometric functions: cos, sin, tan, acos, asin, atan

Hyperbolic functions: cosh, sinh, acosh, asinh, atanh

Exponential and logarithmic: exp, ln, log

Other: abs, sqrt, u, u2, uramp,

Functions of two variables are: min, max, pow

Functions of three variables are: a ? b:c

The function "u" is the unit step function, with a value of one for arguments greater than zero and a value of zero for arguments less than zero. The function "u2" returns a value of zero for arguments less than zero, one for arguments greater than one and assumes the value of the argument between these limits. The function "uramp" is the integral of the unit step: for an input x, the value is zero if x is less than zero, or if x is greater than zero the value is x. These three functions are useful in synthesizing piece-wise non-linear functions, though convergence may be adversely affected.

   The following standard operators are defined: +, -, *, /, ^, unary -
   Logical operators are !=, >=, <=, ==,>,<, |, &&, ! .

A ternary function is defined as `a ?  b :  c` , which means `IF a, THEN b, ELSE c`. Be sure to place a space in front of '?' to allow the parser distinguishing it from other tokens.

Example: Ternary function

```
* B source test Clamped voltage source
* C. P. Basso "Switched-mode power supplies", New York, 2008
.param Vhigh = 4.6
.param Vlow = 0.4
Vin1 1 0 DC 0 PWL(0 0 1u 5)
Bcl 2 0 V = V(1) < Vlow ? Vlow : V(1) > Vhigh ? Vhigh : V(1)
.control
set noaskquit
tran  5n 1u
plot V(2) vs V(1)
.endc
.end
```

If the argument of log, ln, or sqrt becomes less than zero, the absolute value of the argument is used. If a divisor becomes zero or the argument of log or ln becomes zero, an error will result. Other problems may occur when the argument for a function in a partial derivative enters a region where that function is undefined.

Parameters may be used like {Vlow} shown in the example above. Parameters will be evaluated upon set up of the circuit, vectors like V(1) will be evaluated during the simulation.

To get time into the expression you can integrate the current from a constant current source with a capacitor and use the resulting voltage (don't forget to set the initial voltage across the capacitor).

Non-linear resistors, capacitors, and inductors may be synthesized with the nonlinear dependent source. Nonlinear resistors, capacitors and inductors are implemented with their linear counterparts by a change of variables implemented with the nonlinear dependent source. The following subcircuit will implement a nonlinear capacitor:

Example: Non linear capacitor

```
.Subckt nlcap pos neg
* Bx: calculate f(input voltage)
Bx 1 0 v = f(v(pos,neg))
* Cx: linear capacitance
Cx 2 0 1
* Vx: Ammeter to measure current into the capacitor
Vx 2 1 DC 0Volts
* Drive the current through Cx back into the circuit
Fx pos neg Vx 1
.ends
```

Example for f(v(pos,neg)):

```
Bx 1 0 V = v(pos,neg)*v(pos,neg)
```

Non-linear resistors or inductors may be described in a similar manner. An example for a nonlinear resistor using this template is shown below.

Example: Non linear resistor

```
* use of 'hertz' variable in nonlinear resistor
*.param rbase=1k
* some tests
B1   1  0   V = hertz*v(33)
B2   2  0  V = v(33)*hertz
b3   3  0  V = 6.283e3/(hertz+6.283e3)*v(33)
V1 33 0 DC 0 AC 1
*** Translate R1 10 0 R='1k/sqrt(HERTZ)' to B source ***
.Subckt nlres pos neg rb=rbase
* Bx: calculate f(input voltage)
Bx    1      0      v = -1 / {rb} / sqrt(HERTZ) * v(pos, neg)
* Rx: linear resistance
Rx    2      0      1
* Vx: Ammeter to measure current into the resistor
Vx    2      1      DC 0Volts
* Drive the current through Rx back into the circuit
Fx    pos    neg   Vx 1
.ends
Xres 33 10 nlres rb=1k
*Rres 33 10 1k
Vres 10 0 DC 0
.control
define check(a,b) vecmax(abs(a - b))
ac lin 10 100 1k
* some checks
print v(1) v(2) v(3)
if check(v(1), frequency) < 1e-12
echo "INFO: ok"
end
plot vres#branch
.endc
.end
```

par('expression'): The B source syntax may also be used in output lines like `.plot` as algebraic expressions for output (see chapt.).

Piecewise Linear Function: pwl

Both B source types may contain a piece-wise linear dependency of one network variable:

Example: pwl_current

```
Bdio 1 0 I = pwl(v(A), 0,0, 33,10m, 100,33m, 200,50m)
```

v(A) is the independent variable x. Each pair of values following describes the x,y functional relation: In this example at node A voltage of 0V the current of 0A is generated - next pair gives 10mA flowing from ground to node 1 at 33V on node A and so forth.

The same is possible for voltage sources:

Example: pwl_voltage

```
Blimit b 0 V = pwl(v(1), -4,0, -2,2, 2,4, 4,5, 6,5)
```

Monotony of the independent variable in the pwl definiton is checked - nonmonotonic x entries will stop the program execution. v(1) may be replaced by a controlling current source. v(1) may also be replaced by an expression, e.g. -2*i(Vin). The value pairs may also be parameters, which have to be defined before by a `.param` statement. An example for the pwl function using all of these options is shown below:

Example: pwl function in B source

Demonstrates usage of the pwl function in an B source (ASRC)
* Also emulates the TABLE function with limits

.param x0=−4 y0=0
.param x1=−2 y1=2
.param x2=2 y2=−2
.param x3=4 y3=1
.param xx0=x0−1
.param xx3=x3+1

Vin   1 0   DC=0V
R 1 0 2

* no limits outside of the tabulated x values (continues linearily)
Btest2 2 0   I = pwl(v(1), 'x0','y0', 'x1','y1', 'x2','y2', 'x3','y3')

* like TABLE function with limits:
Btest3 3 0   I = (v(1) < 'x0') ? 'y0' :   (v(1) < 'x3') ?
+ pwl(v(1), 'x0','y0', 'x1','y1', 'x2','y2', 'x3','y3') : 'y3'

* more efficient and elegant TABLE function with limits:
Btest4 4 0   I = pwl(v(1),
+ 'xx0','y0', 'x0','y0',
+   'x1','y1',
+   'x2','y2',
+   'x3','y3', 'xx3','y3')

* controlled by current
* more efficient and elegant TABLE function with limits:
Btest5 5 0   I = pwl(−2*i(Vin),
+ 'xx0','y0', 'x0','y0',
+   'x1','y1',
+   'x2','y2',
+   'x3','y3', 'xx3','y3')

Rint2 2 0 1
Rint3 3 0 1
Rint4 4 0 1
Rint5 5 0 1
.control
dc Vin −6 6   0.2
plot v(2) v(3) v(4)−0.5 v(5)+0.5
.endc

.end

6.2   E source (non-linear voltage source)*

General form:

EXXXXXXX n+ n− vol='expr '

Examples:

E41 4 0 vol = 'V(3)*V(3)−Offs '

Expression may be an equation or an expression containing node voltages or branch currents (in the form of i(vm)) and any other terms as given for the B source and described in chapter 6.1. It may

contain parameters (2.8.1).

6.3 G source (non-linear current source)*

General form:

GXXXXXXX n+ n− cur='expr '

Examples:

G51 55 225 cur = 'V(3)∗V(3)−Offs '

Expression may be an equation or an expression containing node voltages or branch currents (in the form of i(vm)) and any other terms as given for the B source and described in chapter 6.1. It may contain parameters (2.8.1). An example file is given below.
Example input file:

```
VCCS, VCVS, non−linear  dependency
.param  Vi=1
.param  Offs='0.01∗Vi'
∗ VCCS depending on V(3)
B21 int1  0  V = V(3)∗V(3)
G1 21 22 int1 0 1
∗ measure current through VCCS
vm 22 0 dc 0
R21 21 0 1
∗ new VCCS depending on V(3)
G51 55 225 cur = 'V(3)∗V(3)−Offs '
∗ measure current through VCCS
vm5 225 0 dc 0
R51 55 0 1
∗ VCVS depending on V(3)
B31 int2 0 V = V(3)∗V(3)
E1 1 0 int2 0 1
R1 1 0 1
∗ new VCVS depending on V(3)
E41 4 0 vol = 'V(3)∗V(3)−Offs '
R4 4 0 1
∗ control voltage
V1 3 0 PWL(0 0 100u {Vi})
.control
set noaskquit
tran 10n 100u uic
plot i(E1) i(E41)
plot i(vm) i(vm5)
.endc
.end
```

*) To get this functionality, the compatibility mode has to be set in **spinit** by `set ngbehavior=all`.

Chapter 7

Transmission Lines

Ngspice implements both the original spice3f5 transmission lines models and the one introduced with kspice. The latter provide an improved transient analysis of lossy transmission lines. Unlike spice models, which uses the state-based approach to simulate lossy transmission lines, kspice simulates lossy transmission lines and coupled multiconductor line systems using the recursive convolution method. The impulse response of an arbitrary transfer function can be determined by deriving a recursive convolution from the Pade approximations of the function. We use this approach for simulating each transmission line's characteristics and each multiconductor line's modal functions. This method of lossy transmission line simulation has been proved to give a speedup of one to two orders of magnitude over spice3f5.

7.1 Lossless Transmission Lines

General form:

TXXXXXXX N1 N2 N3 N4 Z0=VALUE <TD=VALUE> <F=FREQ <NL=NRMLEN>>
+ <IC=V1, I1, V2, I2>

Examples:

T1 1 0 2 0 Z0=50 TD=10NS

`n1` and `n2` are the nodes at port 1; `n3` and `n4` are the nodes at port 2. `z0` is the characteristic impedance. The length of the line may be expressed in either of two forms. The transmission delay, `td`, may be specified directly (as td=10ns, for example). Alternatively, a frequency `f` may be given, together with `nl`, the normalized electrical length of the transmission line with respect to the wavelength in the line at the frequency "f". If a frequency is specified but `nl` is omitted, 0.25 is assumed (that is, the frequency is assumed to be the quarter-wave frequency). Note that although both forms for expressing the line length are indicated as optional, one of the two must be specified.

Note that this element models only one propagating mode. If all four nodes are distinct in the actual circuit, then two modes may be excited. To simulate such a situation, two transmission-line elements are required. (see the example in chapt. 21.7 for further clarification.) The (optional) initial condition specification consists of the voltage and current at each of the transmission line ports. Note that the initial conditions (if any) apply "only" if the `UIC` option is specified on the `.TRAN` control line.

Note that a lossy transmission line (see below) with zero loss may be more accurate than than the lossless transmission line due to implementation details.

7.2 Lossy Transmission Lines

General form:

OXXXXXXX n1 n2 n3 n4 mname

Examples:

O23 1 0 2 0 LOSSYMOD
OCONNECT 10 5 20 5 INTERCONNECT

This is a two-port convolution model for single conductor lossy transmission lines. **n1** and **n2** are the nodes at port 1; **n3** and **n4** are the nodes at port 2. Note that a lossy transmission line with zero loss may be more accurate than than the lossless transmission line due to implementation details.

7.2.1 Lossy Transmission Line Model (LTRA)

The uniform RLC/RC/LC/RG transmission line model (referred to as the LTRA model henceforth) models a uniform constant-parameter distributed transmission line. The RC and LC cases may also be modelled using the URC and TRA models; however, the newer LTRA model is usually faster and more accurate than the others. The operation of the LTRA model is based on the convolution of the transmission line's impulse responses with its inputs (see [8]). The LTRA model takes a number of parameters, some of which must be given and some of which are optional.

Name	Parameter	Units/Type	Default	Example
R	resistance/length	$\Omega/unit$	0.0	0.2
L	inductance/length	$H/unit$	0.0	9.13e-9
G	conductance/length	$mhos/unit$	0.0	0.0
C	capacitance/length	$F/unit$	0.0	3.65e-12
LEN	length of line		no default	1.0
REL	breakpoint control	arbitrary unit	1	0.5
ABS	breakpoint control		1	5
NOSTEPLIMIT	don't limit timestep to less than line delay	flag	not set	set
NOCONTROL	don't do complex timestep control	flag	not set	set
LININTERP	use linear interpolation	flag	not set	set
MIXEDINTERP	use linear when quadratic seems bad	flag	not set	set
COMPACTREL	special reltol for history compaction		RELTOL	1.0e-3
COMPACTABS	special abstol for history compaction		ABSTOL	1.0e-9
TRUNCNR	use Newton-Raphson method for timestep control	flag	not set	set
TRUNCDONTCUT	don't limit timestep to keep impulse-response errors low	flag	not set	set

The following types of lines have been implemented so far:

- RLC (uniform transmission line with series loss only),

- RC (uniform RC line),

- LC (lossless transmission line),

- RG (distributed series resistance and parallel conductance only).

Any other combination will yield erroneous results and should not be tried. The length `LEN` of the line must be specified. `NOSTEPLIMIT` is a flag that will remove the default restriction of limiting time-steps to less than the line delay in the RLC case. `NOCONTROL` is a flag that prevents the default limiting of the time-step based on convolution error criteria in the RLC and RC cases. This speeds up simulation but may in some cases reduce the accuracy of results. `LININTERP` is a flag that, when specified, will use linear interpolation instead of the default quadratic interpolation for calculating delayed signals. `MIXEDINTERP` is a flag that, when specified, uses a metric for judging whether quadratic interpolation is not applicable and if so uses linear interpolation; otherwise it uses the default quadratic interpolation. `TRUNCDONTCUT` is a flag that removes the default cutting of the time-step to limit errors in the actual calculation of impulse-response related quantities. `COMPACTREL` and `COMPACTABS` are quantities that control the compaction of the past history of values stored for convolution. Larger values of these lower accuracy but usually increase simulation speed. These are to be used with the `TRYTOCOMPACT` option, described in the `.OPTIONS` section. `TRUNCNR` is a flag that turns on the use of Newton-Raphson iterations to determine an appropriate timestep in the timestep control routines. The default is a trial and error procedure by cutting the previous timestep in half. `REL` and `ABS` are quantities that control the setting of breakpoints.

The option most worth experimenting with for increasing the speed of simulation is `REL`. The default value of 1 is usually safe from the point of view of accuracy but occasionally increases computation time. A value greater than 2 eliminates all breakpoints and may be worth trying depending on the nature of the rest of the circuit, keeping in mind that it might not be safe from the viewpoint of accuracy.

Breakpoints may usually be entirely eliminated if it is expected the circuit will not display sharp discontinuities. Values between 0 and 1 are usually not required but may be used for setting many breakpoints.

`COMPACTREL` may also be experimented with when the option `TRYTOCOMPACT` is specified in a `.OPTIONS` card. The legal range is between 0 and 1. Larger values usually decrease the accuracy of the simulation but in some cases improve speed. If `TRYTOCOMPACT` is not specified on a `.OPTIONS` card, history compaction is not attempted and accuracy is high.

`NOCONTROL`, `TRUNCDONTCUT` and `NOSTEPLIMIT` also tend to increase speed at the expense of accuracy.

7.3 Uniform Distributed RC Lines

General form:

UXXXXXXX n1 n2 n3 mname l=len <n=lumps>

Examples:

U1 1 2 0 URCMOD L=50U
URC2 1 12 2 UMODL l=1MIL N=6

`n1` and `n2` are the two element nodes the RC line connects, while `n3` is the node to which the capacitances are connected. `mname` is the model name, `len` is the length of the RC line in meters. `lumps`, if specified, is the number of lumped segments to use in modelling the RC line (see the model description for the action taken if this parameter is omitted).

7.3.1 Uniform Distributed RC Model (URC)

The URC model is derived from a model proposed by L. Gertzberrg in 1974. The model is accomplished by a subcircuit type expansion of the URC line into a network of lumped RC segments with internally generated nodes. The RC segments are in a geometric progression, increasing toward the middle of the URC line, with K as a proportionality constant. The number of lumped segments used, if not specified for the URC line device, is determined by the following formula:

$$N = \frac{\log \left| F_{\max} \frac{R}{L} \frac{C}{L} 2\pi L^2 \left| \frac{(K-1)}{K} \right|^2 \right|}{\log K} \tag{7.1}$$

The URC line is made up strictly of resistor and capacitor segments unless the `ISPERL` parameter is given a nonzero value, in which case the capacitors are replaced with reverse biased diodes with a zero-bias junction capacitance equivalent to the capacitance replaced, and with a saturation current of `ISPERL` amps per meter of transmission line and an optional series resistance equivalent to `RSPERL` ohms per meter.

Name	Parameter	Units	Default	Example	Area
K	Propagation Constant	-	2.0	1.2	-
FMAX	Maximum Frequency of interest	Hz	1.0 G	6.5 Meg	-
RPERL	Resistance per unit length	Ω/m	1000	10	-
CPERL	Capacitance per unit length	F/m	10e-15	1 pF	-
ISPERL	Saturation Current per unit length	A/m	0	-	-
RSPERL	Diode Resistance per unit length	Ω/m	0	-	-

7.4 KSPICE Lossy Transmission Lines

Unlike SPICE3, which uses the state-based approach to simulate lossy transmission lines, KSPICE simulates lossy transmission lines and coupled multiconductor line systems using the recursive convolution method. The impulse response of an arbitrary transfer function can be determined by deriving a recursive

convolution from the Pade approximations of the function. NGSPICE is using this approach for simulating each transmission line's characteristics and each multiconductor line's modal functions. This method of lossy transmission line simulation has proven to give a speedup of one to two orders of magnitude over SPICE3E.

Additional Documentation Available:

- S. Lin and E. S. Kuh, "Pade Approximation Applied to Transient Simulation of Lossy Coupled Transmission Lines," Proc. IEEE Multi-Chip Module Conference, 1992, pp. 52-55.

- S. Lin, M. Marek-Sadowska, and E. S. Kuh, "SWEC: A StepWise Equivalent Conductance Timing Simulator for CMOS VLSI Circuits," European Design Automation Conf., February 1991, pp. 142-148.

- S. Lin and E. S. Kuh, "Transient Simulation of Lossy Interconnect," Proc. Design Automation Conference, Anaheim, CA, June 1992, pp. 81-86.

7.4.1 Single Lossy Transmission Line (TXL)

General form:

YXXXXXXX N1 0 N2 0 mname <LEN=LENGTH>

Example:

Y1 1 0 2 0 ymod LEN=2
.MODEL ymod txl R=12.45 L=8.972e−9 G=0 C=0.468e−12 length=16

n1 and n2 are the nodes of the two ports; Optional instance parameter len is the length of the line may be expressed in [m].

The TXL model takes a number of parameters:

Name	Parameter	Units/Type	Default	Example
R	resistance/length	$\Omega/_{unit}$	0.0	0.2
L	inductance/length	$H/_{unit}$	0.0	9.13e-9
G	conductance/length	$mhos/_{unit}$	0.0	0.0
C	capacitance/length	$F/_{unit}$	0.0	3.65e-12
LENGTH	length of line		no default	1.0

Model parameter length must be specified.

7.4.2 Coupled Multiconductor Line (CPL)

The CPL multiconductor line model, which in theory should be similar to the RLGC model, but without frequency dependent loss (neither skin effect and nor frequency dependent dielectric loss). Up to 8 coupled lines are supported in NGSPICE.

General form:

PXXXXXXX NI1 NI2 ... NIX GND1 NO1 NO2 ... NOX GND2 mname <LEN=LENGTH>

Example:

P1 in1 in2 0 b1 b2 0 PLINE
.model PLINE CPL length={Len}
+R=1 0 1
+L={L11} {L12} {L22}
+G=0 0 0
+C={C11} {C12} {C22}
.param Len=1 Rs=0
+ C11=9.143579E−11 C12=−9.78265E−12 C22=9.143578E−11
+ L11=3.83572E−7 L12=8.26253E−8 L22=3.83572E−7

ni1 ... nix are the nodes at port 1 with gnd1; no1 ... nox are the nodes at port 2 with gnd2. Optional instance parameter len is the length of the lines may be expressed in [m].

The CPL model takes a number of parameters:

Name	Parameter	Units/Type	Default	Example
R	resistance/length	$\Omega/unit$	0.0	0.2
L	inductance/length	$H/unit$	0.0	9.13e-9
G	conductance/length	$mhos/unit$	0.0	0.0
C	capacitance/length	$F/unit$	0.0	3.65e-12
LENGTH	length of line		no default	1.0

All RLGC parameter are given in Maxwell matrix form. For R and G matrix the diagonal elements must be specified, for L and C matrix the lower or upper-triangular elements must specified. Model parameter LENGTH is a scalar and is mandatory.

Chapter 8

DIODEs

8.1 Junction Diodes

General form:

DXXXXXXX n+ n− mname <area=val> <m=val> <pj=val> <off> <ic=vd> <temp=val>
+ <dtemp=val>

Examples:

DBRIDGE 2 10 DIODE1
DCLMP 3 7 DMOD 3.0 IC=0.2

The pn junction (diode) implemented in ngspice expands the one in spice3f5. Perimetral effects and high injection level have been introduced into the original model and temperature dependence of some parameters has been added. n+ and n- are the positive and negative nodes, respectively. mname is the model name, area is the area factor, pj is the perimeter factor, and off indicates an (optional) starting condition on the device for dc analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification using ic is intended for use with the uic option on the .tran control line, when a transient analysis is desired starting from other than the quiescent operating point. You should supply the inital voltage across the diode there. The (optional) temp value is the temperature at which this device is to operate, and overrides the temperature specification on the .option control line. The temperatire of each istance can be can be specified as an offset to the circuit temperature with the dtemp option.

8.2 Diode Model (D)

The dc characteristics of the diode are determined by the parameters is and n. An ohmic resistance, rs, is included. Charge storage effects are modeled by a transit time, tt, and a nonlinear depletion layer capacitance which is determined by the parameters cjo, vj, and m. The temperature dependence of the saturation current is defined by the parameters eg, the energy and xti, the saturation current temperature exponent. The nominal temperature at which these parameters were measured is tnom, which defaults to the circuit-wide value specified on the .options control line. Reverse breakdown is modelled by an exponential increase in the reverse diode current and is determined by the parameters bv and ibv (both of which are positive numbers).

Junction DC parameters

Name	Parameter	Units	Default	Example	Scale factor
BV	Reverse breakdown voltage	V	∞	40	
IBV	Current at breakdown voltage	A	1.0e-3	1.0e-4	
IK (IKF)	Forward knee current	A	1.0e-3	1.0e-6	
IKR	Reverse knee current	A	1.0e-3	1.0e-6	
IS (JS)	Saturation current	A	1.0e-14	1.0e-16	area
JSW	Sidewall saturation current	A	1.0e-14	1.0e-15	perimeter
N	Emission coefficient	-	1	1.5	
RS	Ohmic resistance	Ω	0.0	100	$^1/area$

Junction capacitance parameters

Name	Parameter	Units	Default	Example	Scale factor
CJO (CJ0)	Zero-bias junction bottowall capacitance	F	0.0	2pF	area
CJP (CJSW)	Zero-bias junction sidewall capacitance	F	0.0	.1pF	perimeter
FC	Coefficient for forward-bias depletion bottomwall capacitance formula	-	0.5	-	
FCS	Coefficient for forward-bias depletion sidewall capacitance formula	-	0.5	-	
M (MJ)	Area junction grading coefficient	-	0.5	0.5	
MJSW	Periphery junction grading coefficient	-	0.33	0.5	
VJ	Junction potential	V	1	0.6	
PHP	Periphery junction potential	V	1	0.6	
TT	Transit-time	sec	0	0.1ns	

Temperature effects

Name	Parameter	Units	Default	Example	Scale factor
EG	Activation energy	eV	1.11	1.11 Si 0.69 Sbd 0.67 Ge	
TM1	1st order tempco for MJ	$^1/^\circ C$	0.0	-	
TM2	2nd order tempco for MJ	$^1/^\circ C^2$	0.0	-	
TNOM	Parameter measurement temperature	$^\circ C$	27	50	
TRS	1st order tempco for RS	$^1/^\circ C$	0.0	-	
TRS2	2nd order tempco for RS	$^1/^\circ C^2$	0.0	-	
TTT1	1st order tempco for TT	$^1/^\circ C$	0.0	-	
TTT2	2nd order tempco for TT	$^1/^\circ C^2$	0.0	-	
XTI	Saturation current temperature exponent	-	3.0	3.0 pn 2.0 Sbd	

Noise modeling

Name	Parameter	Units	Default	Example	Scale factor
KF	Flicker noise coefficient	-	0		
AF	Flicker noise exponent	-	1		

8.3 Diode Equations

The junction diode is the the basic semiconductor device and the simplest one modeled in ngspice, but it's model is quite complex, even if not all the physical phenomena affecting a pn junction are modelled. The diode is modeled in three different regions:

- *Forward bias*: the anode is more positive than the cathode, the diode is "on" and can conduct large currents. To avoid convergence problems and unrealistic high current, it is better to specify a series resistance to limit current with `rs` model parameter.

Algorithm 8.1 Diode breakdown current calculation

---

if $IBV_{eff} < I_{bdwn}$ then
$\quad IBV_{eff} = I_{bdwn}$
$\quad\quad BV_{eff} = \text{BV}$
else
$\quad BV_{eff} = \text{BV} - NV_t \ln(\frac{IBV_{eff}}{I_{bdwn}})$

- *Reverse bias*: the cathode is more positive than the anode and the diode is "off". A reverse bias diode conducts a small leakage current.

- *Breakdown*: the breakdown region is modelled only if the `bv` model parameter is given. When a diode enters breakdown the current increase exponentially (remember to limit it); `bv` is a positive value.

Parameters Scaling

Model parameters are scaled using the unitless parameters `area` and `pj` and the multiplier `m` as depicted below:

$AREA_{eff} = \text{AREA} \cdot \text{M}$
$PJ_{eff} = \text{PJ} \cdot \text{M}$
$IS_{eff} = \text{IS} \cdot AREA_{eff} + \text{JSW} * PJ_{eff}$
$IBV_{eff} = \text{IBV} \cdot AREA_{eff}$
$IK_{eff} = \text{IK} \cdot AREA_{eff}$
$IKR_{eff} = \text{IKR} \cdot AREA_{eff}$
$CJ_{eff} = \text{CJ0} \cdot AREA_{eff}$
$CJP_{eff} = \text{CJP} \cdot PJ_{eff}$

Diode DC, Transient and AC model equations

$$I_D = \begin{cases} IS_{eff}(e^{\frac{qV_D}{NkT}} - 1) + V_D * GMIN, & \text{if } V_D \geq -3\frac{NkT}{q} \\ -IS_{eff}[1 + (\frac{3NkT}{qV_De})^3] + V_D * GMIN, & \text{if } -BV_{eff} < V_D < -3\frac{NkT}{q} \\ -IS_{eff}(e^{\frac{-q(BV_{eff}+V_D)}{NkT}}) + V_D * GMIN, & \text{if } V_D \leq -BV_{eff} \end{cases} \tag{8.1}$$

The breakdown region must be described with more depth since the breakdown is not modelled in physically. As written before, the breakdown modelling is based on two model parameters: the "nominal breakdown voltage" `bv` and the current at the onset of breakdown `ibv`. For the diode model to be consistent, the current value cannot be arbitrary chosen, since the reverse bias and breakdown regions must match. When the diode enters breakdown region from reverse bias, the current is calculated using the formula[1]:

$$I_{bdwn} = -IS_{eff}(e^{\frac{-q\text{BV}}{NkT}} - 1) \tag{8.2}$$

The computed current is necessary to adjust the breakdown voltage making the two regions match. The algorithm is a little bit convoluted and only a brief description is given here:

Most real diodes shows a current increase that, at high current levels, does not follow the exponential relationship given above. This behaviour is due to high level of carriers injected into the junction. High injection effects (as they are called) are modelled with `ik` and `ikr`.

$$I_{Deff} = \begin{cases} \frac{I_D}{1+\sqrt{\frac{I_D}{IK_{eff}}}}, & \text{if } V_D \geq -3\frac{NkT}{q} \\ \frac{I_D}{1+\sqrt{\frac{I_D}{IKR_{eff}}}}, & \text{otherwise.} \end{cases} \tag{8.3}$$

Diode capacitance is divided into two different terms:

- Depletion capacitance

---

[1]if you look at the source code in file `diotemp.c` you will discoverthat the exponential relation is replaced with a first order taylor series expansion.

- Diffusion capacitance

Depletion capacitance is composed by two different contributes, one associated to the bottom of the junction (bottowall depletion capacitance) and the other to the periphery (sidewall depletion capacitance). The basic equations are:

$$C_{Diode} = C_{diffusion} + C_{depletion}$$

Where the depletion capacitance i defined as:

$$C_{depletion} = C_{depl_{bw}} + C_{depl_{sw}}$$

The diffusion capacitance, due to the injected minority carriers is modeled with the transit time `tt`:

$$C_{diffusion} = \text{TT} \frac{\partial I_{Deff}}{\partial V_D}$$

The depletion capacitance is more complex to model, since the function used to approximate it diverges when the diode voltage become greater than the junction built-in potential. To avoid function divergence, the capacitance function is approximated with a linear extrapolation for applied voltage greater than a fraction of the junction built-in potential.

$$C_{depl_{bw}} = \begin{cases} CJ_{eff} \cdot (1 - \frac{V_D}{\text{VJ}})^{-\text{MJ}}, & \text{if } V_D < \text{FC} \cdot \text{VJ} \\ CJ_{eff} \cdot \frac{1 - \text{FC} \cdot (1 + \text{MJ}I) + \text{MJ} \cdot \frac{V_D}{\text{VJ}}}{(1 - \text{FC})^{(1 + \text{MJ})}}, & otherwise. \end{cases} \tag{8.4}$$

$$C_{depl_{sw}} = \begin{cases} CJP_{eff} \cdot (1 - \frac{V_D}{\text{PHP}})^{-\text{MJSW}}, & \text{if } V_D < \text{FCS} \cdot \text{PHP} \\ CJP_{eff} \cdot \frac{1 - \text{FCS} \cdot (1 + \text{MJSW}) + \text{MJSW} \cdot \frac{V_D}{\text{PHP}}}{(1 - \text{FCS})^{(1 + \text{MJSW})}}, & otherwise. \end{cases} \tag{8.5}$$

Temperature dependence

The temperature affects many of the parameters in the equations above, the following equations show how. One of the most significant parameter that varies with the temperature for a semiconductor is the band-gap energy:

$$EG_{nom} = 1.16 - 7.02e^{-4} \cdot \frac{\text{TNOM}^2}{\text{TNOM} + 1108.0} \tag{8.6}$$

$$EG(T) = 1.16 - 7.02e^{-4} \cdot \frac{T^2}{\text{TNOM} + 1108.0} \tag{8.7}$$

The leakeage currents temperature dependence is:

$$IS(T) = \text{IS} \cdot e^{\frac{logfactor}{\text{N}}} \tag{8.8}$$

$$JSW(T) = \text{JSW} \cdot e^{\frac{logfactor}{\text{N}}} \tag{8.9}$$

where "logfactor" is defined:

$$logfactor = \frac{\text{EG}}{V_t(\text{TNOM})} - \frac{\text{EG}}{V_t(T)} + \text{XTI} \cdot \ln(\frac{T}{\text{TNOM}}) \tag{8.10}$$

The contact potentials (bottowall an sidewall) temperature dependence is:

$$VJ(T) = \text{VJ} \cdot (\frac{T}{\text{TNOM}}) - V_t(T) \cdot \left[3 \cdot \ln(\frac{T}{\text{TNOM}}) + \frac{\text{EG}_{nom}}{V_t(\text{TNOM})} - \frac{\text{EG(T)}}{V_t(T)} \right] \tag{8.11}$$

$$PHP(T) = \text{PHP} \cdot (\frac{T}{\text{TNOM}}) - V_t(T) \cdot \left[3 \cdot \ln(\frac{T}{\text{TNOM}}) + \frac{\text{EG}_{nom}}{V_t(\text{TNOM})} - \frac{\text{EG(T)}}{V_t(T)} \right] \tag{8.12}$$

The depletion capacitances temperature dependence is:

$$CJ(T) = \text{CJ} \cdot \left[1 + \text{MJ} \cdot (4.0e^{-4} \cdot (T - \text{TNOM}) - \frac{VJ(T)}{\text{VJ}} + 1) \right] \tag{8.13}$$

$$CJSW(T) = \text{CJSW} \cdot \left[1 + \text{MJSW} \cdot (4.0e^{-4} \cdot (T - \text{TNOM}) - \frac{PHP(T)}{\text{PHP}} + 1) \right] \qquad (8.14)$$

The transit time temperature dependence is:

$$TT(T) = \text{TT} \cdot (1 + \text{TTT1} \cdot (T - \text{TNOM}) + \text{TTT2} \cdot (T - \text{TNOM})^2) \qquad (8.15)$$

The junction grading coefficient temperature dependence is:

$$MJ(T) = \text{MJ} \cdot (1 + \text{TM1} \cdot (T - \text{TNOM}) + \text{TM2} \cdot (T - \text{TNOM})^2) \qquad (8.16)$$

The series resistance temperature dependence is:

$$RS(T) = \text{RS} \cdot (1 + \text{TRS} \cdot (T - \text{TNOM}) + \text{TRS2} \cdot (T - \text{TNOM})^2) \qquad (8.17)$$

Noise model

The diode has three noise contribution, one due to the presence of the parasitic resistance `rs` and the other two (shot and flicker) due to the pn junction.

The thermal noise due to the parasitic resistance is:

$$\overline{i_{RS}^2} = \frac{4kT\Delta f}{RS} \qquad (8.18)$$

The shot and flicker noise contributions are:

$$\overline{i_d^2} = 2qI_D\Delta f + \frac{KF * I_D^{AF}}{f}\Delta f \qquad (8.19)$$

Chapter 9

BJTs

9.1 Bipolar Junction Transistors (BJTs)

General form:

```
QXXXXXXX nc nb ne <ns> mname <area=val> <areac=val> <areab=val>
+ <m=val> <off> <ic=vbe,vce> <temp=val> <dtemp=val>
```

Examples:

```
Q23 10 24 13 QMOD IC=0.6, 5.0
Q50A 11 26 4 20 MOD1
```

`nc`, `nb`, and `ne` are the collector, base, and emitter nodes, respectively. `ns` is the (optional) substrate node. If unspecified, ground is used. `mname` is the model name, `area`, `areab`, `areac` are the area factors (emitter, base and collector respectively), and `off` indicates an (optional) initial condition on the device for the dc analysis. If the area factor is omitted, a value of 1.0 is assumed.

The (optional) initial condition specification using `ic=vbe,vce` is intended for use with the `uic` option on a `.tran` control line, when a transient analysis is desired starting from other than the quiescent operating point. See the `.ic` control line description for a better way to set transient initial conditions. The (optional) `temp` value is the temperature at which this device is to operate, and overrides the temperature specification on the `.option` control line. Using `dtemp` option you can specify instance's temperature relative to the circuit temperature.

9.2 BJT Models (NPN/PNP)

Ngspice provides three BJT device models. The `level` keyword specifies the model to be used:

- level=1 : This is the original spice BJT model, and it is the default model if the `level` keyword is not specified on the `.model` line.

- level=2 : This is a modified version of the original spice BJT that models both vertical and lateral devices and includes temperature corrections of collector, emitter and base resistors.

- level=4: Advanced VBIC model (see http://www.designers-guide.org/VBIC/ for details)

The bipolar junction transistor model in ngspice is an adaptation of the integral charge control model of Gummel and Poon. This modified Gummel-Poon model extends the original model to include several effects at high bias levels. The model automatically simplifies to the simpler Ebers-Moll model when certain parameters are not specified. The parameter names used in the modified Gummel-Poon model have been chosen to be more easily understood by the program user, and to reflect better both physical and circuit design thinking.

The dc model is defined by the parameters `is`, `bf`, `nf`, `ise`, `ikf`, amd `ne` which determine the forward current gain characteristics, `is`, `br`, `nr`, `isc`, `ikr`, and `nc` which determine the reverse current gain characteristics, and `vaf` and `var` which determine the output conductance for forward and reverse regions.

Level 2 model includes substrate saturation current `iss`. Three ohmic resistances `rb`, `rc`, and `re` are included, where `rb` can be high current dependent. Base charge storage is modelled by forward

and reverse transit times, `tf` and `tr`, the forward transit time `tf` being bias dependent if desired, and nonlinear depletion layer capacitances which are determined by `cje`, `vje`, and `nje` for the B-E junction, `cjc`, `vjc`, and `njc` for the B-C junction and `cjs`, `vjs`, and `mjs` for the C-S (Collector-Substrate) junction.

Level 2 model defines a substrate capacitance that will be connected to device's base or collector, to model lateral or vertical devices. The temperature dependence of the saturation currents, `is` and `iss` (for level 2 model), is determined by the energy-gap, `eg`, and the saturation current temperature exponent, `xti`.

Additionally base current temperature dependence is modelled by the beta temperature exponent `xtb` in the new model. The values specified are assumed to have been measured at the temperature `tnom`, which can be specified on the `.options` control line or overridden by a specification on the `.model` line.

Level 4 model (VBIC) has the following improvements beyond the GP models: Improved Early effect modeling, Quasi-saturation modeling, Parasitic substrate transistor modeling, Parasitic fixed (oxide) capacitance modeling, Includes an avalanche multiplication model, Improved temperature modeling, Base current is decoupled from collector current, Electrothermal modeling, Smooth, continuous mode.

The BJT parameters used in the modified Gummel-Poon model are listed below. The parameter names used in earlier versions of spice2 are still accepted.

Modified Gummel-Poon BJT Parameters

Name	Parameters	Units	Default	Example	Scale factor
SUBS	Substrate connection: for vertical geometry, -1 for lateral geometry (level 2 only).		1		
IS	Transport saturation current.	A	1.0e-16	1.0e-15	area
ISS	Reverse saturation current, substrate-to-collector for vertical device or substrate-to-base for lateral (level 2 only).	A	1.0e-16	1.0e-15	area
BF	Ideal maximum forward beta.	-	100	100	
NF	Forward current emission coefficient.	-	1.0	1	
VAF	Forward Early voltage.	V	∞	200	
IKF	Corner for forward beta current roll-off.	A	∞	0.01	area
ISE	B-E leakage saturation current.	A	0.0	1e-13	area
NE	B-E leakage emission coefficient.	-	1.5	2	
BR	Ideal maximum reverse beta.	-	1	0.1	
NR	Reverse current emission coefficient.	-	1	1	
VAR	Reverse Early voltage.	V	∞	200	
IKR	Corner for reverse beta high current roll-off.	A	∞	0.01	area
ISC	B-C leakage saturation current (area is "areab" for vertical devices and "areac" for lateral).	A	0.0	1e-13	area
NC	B-C leakage emission coefficient.	-	2	1.5	
RB	Zero bias base resistance.	Ω	0	100	area
IRB	Current where base resistance falls halfway to its min value.	A	∞	0.1	area
RBM	Minimum base resistance at high currents.	Ω	RB	10	area
RE	Emitter resistance.	Ω	0	1	area
RC	Collector resistance.	Ω	0	10	area
CJE	B-E zero-bias depletion capacitance.	F	0	2pF	area
VJE	B-E built-in potential.	V	0.75	0.6	
MJE	B-E junction exponential factor.	-	0.33	0.33	
TF	Ideal forward transit time.	sec	0	0.1ns	
XTF	Coefficient for bias dependence of TF.	-	0		

VTF	Voltage describing VBC dependence of TF.	V	∞		
ITF	High-current parameter for effect on TF.	A	0	-	area
PTF	Excess phase at freq=1.0/(TF*2PI) Hz.	deg	0		
CJC	B-C zero-bias depletion capacitance (area is "areab" for verticaldevices and "areac" for lateral).	F	0	2pF	area
VJC	B-C built-in potential.	V	0.75	0.5	
MJC	B-C junction exponential factor.	-	0.33	0.5	
XCJC	Fraction of B-C depletion capacitance connected to internal base node.	-	1		
TR	Ideal reverse transit time.	sec	0	10ns	
CJS	Zero-bias collector-substrate capacitance (area is "areac" for vertical devices and"areab" for lateral).	F	0	2pF	area
VJS	Substrate junction built-in potential.	V	0.75		
MJS	Substrate junction exponential factor.	-	0	0.5	
XTB	Forward and reverse beta temperature exponent.	-	0		
EG	Energy gap for temperature effect on IS.	eV	1.11		
XTI	Temperature exponent for effect on IS.	-	3		
KF	Flicker-noise coefficient.	-	0		
AF	Flicker-noise exponent.	-	1		
FC	Coefficient for forward-bias depletion capacitance formula.	-	0.5	0	
TNOM	Parameter measurement temperature.	$^\circ C$	27	50	
TRE1	1st order temperature coefficient for RE (level 2 only).	$1/^\circ C$	0.0	1e-3	
TRE2	2nd order temperature coefficient for RE (level 2 only).	$1/^\circ C^2$	0.0	1e-5	
TRC1	1st order temperature coefficient for RC (level 2 only).	$1/^\circ C$	0.0	1e-3	
TRC2	2nd order temperature coefficient for RC (level 2 only).	$1/^\circ C^2$	0.0	1e-5	
TRB1	1st order temperature coefficient for RB (level 2 only).	$1/^\circ C$	0.0	1e-3	
TRB2	2nd order temperature coefficient for RB (level 2 only).	$1/^\circ C^2$	0.0	1e-5	
TRBM1	1st order temperature coefficient for RBM	$1/^\circ C$	0.0	1e-3	
TRBM2	2nd order temperature coefficient for RBM	$1/^\circ C^2$	0.0	1e-5	

Chapter 10

JFETs

10.1 Junction Field-Effect Transistors (JFETs)

General form:

JXXXXXXX nd ng ns mname <area> <off> <ic=vds ,vgs> <temp=t>

Examples:

J1 7 2 3 JM1 OFF

nd, ng, and ns are the drain, gate, and source nodes, respectively. mname is the model name, area is the area factor, and off indicates an (optional) initial condition on the device for dc analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification, using ic=VDS,VGS is intended for use with the uic option on the .TRAN control line, when a transient analysis is desired starting from other than the quiescent operating point. See the .ic control line for a better way to set initial conditions. The (optional) temp value is the temperature at which this device is to operate, and overrides the temperature specification on the .option control line.

10.2 JFET Models (NJF/PJF)

10.2.1 Model by Parker and Skellern

The **level 1** JFET model is derived from the FET model of Shichman and Hodges. The dc characteristics are defined by the parameters vto and beta, which determine the variation of drain current with gate voltage, lambda, which determines the output conductance, and is, the saturation current of the two gate junctions. Two ohmic resistances, rd and rs, are included. Charge storage is modelled by nonlinear depletion layer capacitances for both gate junctions which vary as the $-1/2$ power of junction voltage and are defined by the parameters cgs, cgd, and pb.

Note that in Spice3f and later, a fitting parameter "b" has been added. For details, see [9].

Name	Parameter	Units	Default	Example	Scaling factor
VTO	Threshold voltage V_{T0}	V	-2.0	-2.0	
BETA	Transconductance parameter (β)	$^A/_{V^"}$	1.0e-4	1.0e-3	area
LAMBDA	Channel-length modulation parameter (λ)	$^1/_V$	0	1.0e-4	
RD	Drain ohmic resistance	Ω	0	100	area
RS	Source ohmic resistance	Ω	0	100	area
CGS	Zero-bias G-S junction capacitance C_{gs}	F	0	5pF	area
CGD	Zero-bias G-D junction capacitance C_{gd}	F	0	1pF	area
PB	Gate junction potential	V	1	0.6	
IS	Gate saturation current I_S	A	1.0e-14	1.0e-14	area
B	Doping tail parameter	-	1	1.1	
KF	Flicker noise coefficient	-	0		
AF	Flicker noise exponent	-	1		
FC	Coefficient for forward-bias depletion capacitance formula		0.5		
TNOM	Parameter measurement temperature	$^\circ C$	27	50	

10.2.2   Modified Parker Skellern model

The level 2 model is an improvement to level 1. Details are available from Macquarie University. Some important items are:

- The description maintains strict continuity in its high-order derivatives, which is essential for prediction of distortion and intermodulation.

- Frequency dependence of output conductance and transconductance is described as a function of bias.

- Both drain-gate and source-gate potentials modulate the pinch-off potential, which is consistent with S-parameter and pulsed-bias measurements.

- Self-heating varies with frequency.

- Extreme operating regions - subthreshold, forward gate bias, controlled resistance, and breakdown regions - are included.

- Parameters provide independent fitting to all operating regions. It is not necessary to compromise one region in favour of another.

- Strict drain-source symmetry is maintained. The transition during drain-source potential reversal is smooth and continuous.

The model equations are described in this pdf document and in [19].

Name	Description	Unit Type	Default
ID	Device IDText	Text	PF1
ACGAM	Capacitance modulation	None	0
BETA	Linear-region transconductance scale	None	10^{-4}
CGD	Zero-bias gate-source capacitance	Capacitance	0 F
CGS	Zero-bias gate-drain capacitance	Capacitance	0 F
DELTA	Thermal reduction coefficient	None	0 W
FC	Forward bias capacitance parameter	None	0.5
HFETA	High-frequency VGS feedback parameter	None	0
HFE1	HFGAM modulation by VGD	None	$0V^{-1}$
HFE2	HFGAM modulation by VGS	None	0 V−1
HFGAM	High-frequency VGD feedback parameter	None	0
HFG1	HFGAM modulation by VSG	None	0 V−1
HFG2	HFGAM modulation by VDG	None	0 V−1
IBD	Gate-junction breakdown current	Current	0 A
IS	Gate-junction saturation current	Current	10−14A
LFGAM	Low-frequency feedback parameter	None	0
LFG1	LFGAM modulation by VSG	None	0 V−1
LFG2	LFGAM modulation by VDG	None	0 V−1
MVST	Subthreshold modulation	None	0 V−1
N	Gate-junction ideality factor	None	1
P	Linear-region power-law exponent	None	2
Q	Saturated-region power-law exponent	None	2
RS	Source ohmic resistance	Resistance	0 Ohm
RD	Drain ohmic resistance	Resistance	0 Ohm
TAUD	Relaxation time for thermal reduction	Time	0 s
TAUG	Relaxation time for gamma feedback	Time	0 s
VBD	Gate-junction breakdown potential	Voltage	1 V
VBI	Gate-junction potential	Voltage	1 V
VST	Subthreshold potential	Voltage	0 V
VTO	Threshold voltage	Voltage	-2.0 V
XC	Capacitance pinch-off reduction factor	None	0
XI	Saturation-knee potential factor	None	1000
Z	Knee transition parameter	None	0.5
RG	Gate ohmic resistance	Resistance	0 Ohm
LG	Gate inductance	Inductance	0 H
LS	Source inductance	Inductance	0 H
LD	Drain inductance	Inductance	0 H
CDSS	Fixed Drain-source capacitance	Capacitance	0 F
AFAC	Gate-width scale factor	None	1
NFING	Number of gate fingers scale factor	None	1
TNOM	Nominal Temperature (Not implemented)	Temperature	300 K
TEMP	Temperature	Temperature	300 K

Chapter 11

MESFETs

11.1 MESFETs

General form:

ZXXXXXXX ND NG NS MNAME <AREA> <OFF> <IC=VDS, VGS>

Examples:

Z1 7 2 3 ZM1 OFF

11.2 MESFET Models (NMF/PMF)

11.2.1 Model by Statz e.a.

The MESFET model **level 1** is derived from the GaAs FET model of Statz et al. as described in [11]. The dc characteristics are defined by the parameters VTO, B, and BETA, which determine the variation of drain current with gate voltage, ALPHA, which determines saturation voltage, and LAMBDA, which determines the output conductance. The formula are given by:

$$I_d = \begin{cases} \frac{B(V_{gs}-V_T)^2}{1+b(V_{gs}-V_T)} \left| 1 - \left| 1 - A\frac{V_{ds}}{3} \right|^3 \right| (1+LV_{ds}) & \text{for } 0 < V_{ds} < \frac{3}{A} \\ \frac{B(V_{gs}-V_T)^2}{1+b(V_{gs}-V_T)}(1+LV_{ds}) & \text{for } V > \frac{3}{A} \end{cases} \qquad (11.1)$$

Two ohmic resistances, `rd` and `rs`, are included. Charge storage is modeled by total gate charge as a function of gate-drain and gate-source voltages and is defined by the parameters `cgs`, `cgd`, and `pb`.

Name	Parameter	Units	Default	Example	Area
VTO	Pinch-off voltage	V	-2.0	-2.0	
BETA	Transconductance parameter	A/V^2	1.0e-4	1.0e-3	*
B	Doping tail extending parameter	$1/V$	0.3	0.3	*
ALPHA	Saturation voltage parameter	$1/V$	2	2	*
LAMBDA	Channel-length modulation parameter	$1/V$	0	1.0e-4	
RD	Drain ohmic resistance	Ω	0	100	*
RS	Source ohmic resistance	Ω	0	100	*
CGS	Zero-bias G-S junction capacitance	F	0	5pF	*
CGD	Zero-bias G-D junction capacitance	F	0	1pF	*
PB	Gate junction potential	V	1	0.6	
KF	Flicker noise coefficient	-	0		
AF	Flicker noise exponent	-	1		
FC	Coefficient for forward-bias depletion capacitance formula	-	0.5		

Device instance:

```
z1 2 3 0 mesmod area=1.4
```

Model:

```
.model mesmod nmf level=1 rd=46 rs=46 vt0=−1.3
+ lambda=0.03 alpha=3 beta=1.4e−3
```

11.2.2   Model by Ytterdal e.a.

level 2 (and levels 3,4) Copyright 1993: T. Ytterdal, K. Lee, M. Shur and T. A. Fjeldly
to be written
M. Shur, T.A. Fjeldly, T. Ytterdal, K. Lee, "Unified GaAs MESFET Model for Circuit Simulation",
Int. Journal of High Speed Electronics, vol. 3, no. 2, pp. 201-233, 1992

11.2.3   hfet1

level 5
to be written
no documentation available

11.2.4   hfet2

level6
to be written
no documentation available

Chapter 12

MOSFETs

Ngspice supports all the original mosfet models present in spice3f5 and almost all the newer ones that have been published and made open-source. Both bulk and SOI (Silicon on Insulator) models are available. When compiled with the cider option, ngspice implements the four terminals numerical model that can be used to simulate a MOSFET (please refer to numerical modeling documentation for additional information and examples).

12.1 MOSFET devices

General form:

```
MXXXXXXX nd ng ns nb mname <m=val> <l=val> <w=val>
+ <ad=val> <as=val> <pd=val> <ps=val> <nrd=val>
+ <nrs=val> <off> <ic=vds, vgs, vbs> <temp=t>
```

Examples:

```
M1 24 2 0 20 TYPE1
M31 2 17 6 10 MODM L=5U W=2U
M1 2 9 3 0 MOD1 L=10U W=5U AD=100P AS=100P PD=40U PS=40U
```

Note the suffixes in the example: the suffix "u" specifies microns (1e-6 m) and "p" sq-microns (1e-12 m^2).

In the instance card, `nd`, `ng`, `ns`, and `nb` are the drain, gate, source, and bulk (substrate) nodes, respectively. `mname` is the model name and `m` is the multiplicity parameter, which simulates "m" paralleled devices. All MOS models support the "m" parameter. Instance parameters `l` and `w`, channel length and width respectively, are expressed in meters. The areas of drain and source diffusions: `ad` and `as`, in squared mters (m^2).

If any of `l`, `w`, `ad`, or `as` are not specified, default values are used. The use of defaults simplifies input file preparation, as well as the editing required if device geometries are to be changed. `pd` and `ps` are the perimeters of the drain and source junctions, in meters. `nrd` and `nrs` designate the equivalent number of squares of the drain and source diffusions; these values multiply the sheet resistance `rsh` specified on the `.model` control line for an accurate representation of the parasitic series drain and source resistance of each transistor. `pd` and `ps` default to 0.0 while `nrd` and `nrs` to 1.0. `off` indicates an (optional) initial condition on the device for dc analysis. The (optional) initial condition specification using `ic=vds,vgs,vbs` is intended for use with the `uic` option on the `.tran` control line, when a transient analysis is desired starting from other than the quiescent operating point. See the `.ic` control line for a better and more convenient way to specify transient initial conditions. The (optional) `temp` value is the temperature at which this device is to operate, and overrides the temperature specification on the `.option` control line.

The temperature specification is ONLY valid for level 1, 2, 3, and 6 MOSFETs, not for level 4 or 5 (BSIM) devices.

12.2   MOSFET models (NMOS/PMOS)

MOSFET models are the central part of ngspice, probably because they are the most widely used devices in the electronics world. Ngspice provides all the MOSFETs implemented in the original Spice3f and adds several models developed by Berkeley's Device Group and other independent groups. The variable `level` specifies the model to be used and a short summary of available device is show in 12.1.

Note: not all models below are included in the standard ngspice distribution because of copyright restrictions.

Ngspice provides four MOSFET device models, which differ in the formulation of the I-V characteristic.

12.2.1   MOS Level 1

This model is also known as the "Schichman-Hodges" model. This is the first model written and the one often described in the introductory textbooks of electronics. This model i applicable only to long channel devices and, the use of Meyer's model for the C-V part makes it non charge conserving.

12.2.2   MOS Level 2

This model tries to overcome the limitations of the Level 1 model addressing several short-channel effect, like velocity saturation. The implementation of this model is complicated and this leads to many convergence problems. C-V calculations can be done with the original Meyer model (non conserving).

12.2.3   MOS Level 3

This is a semiempirical model derived from the Level 2 one. This model is often used for digital design and, in the years, has proven to be robust. A discontinuity in the model with respect to the KAPPA parameter has been detected (see [10]). The supplied fix has been implemented in Spice3f2 and later. Since this fix may affect parameter fitting, the option "`badmos3`" may be set to use the old implementation (see the section on simulation variables and the "`.options`" line). Ngspice level 3 implementation takes into account length and width mask adjustments (`xl` and `xw`) and device width narrowing due to diffusion (`wd`).

12.2.4   MOS Level 6

This model is described in [2]. The model can express the current characteristics of short-channel MOS-FETs at least down to 0. 25 μm channel-length, GaAs FET, and resistance inserted MOSFETs. The model evaluation time is about $1/3$ of the evaluation time of the spice3 mos level 3 model. The model also enables analytical treatments of circuits in short-channel region and makes up for a missing link between a complicated MOSFET current characteristics and circuit behaviors in the deep submicron region.

12.2.5   Notes on Level 1-6 models

The dc characteristics of the level 1 through level 3 MOSFETs are defined by the device parameters `vto`, `kp`, `lambda`, `phi` and `gamma`. These parameters are computed by ngspice if process parameters (`nsub`, `tox`, ...) are given, but users specified values always override. `vto` is positive (negative) for enhancement mode and negative (positive) for depletion mode N-channel (P-channel) devices.

Charge storage is modeled by three constant capacitors, `cgso`, `cgdo`, and `cgbo` which represent overlap capacitances, by the nonlinear thin-oxide capacitance which is distributed among the gate, source, drain, and bulk regions, and by the nonlinear depletion-layer capacitances for both substrate junctions divided into bottom and periphery, which vary as the `mj` and `mjsw` power of junction voltage respectively, and are determined by the parameters `cbd`, `cbs`, `cj`, `cjsw`, `mj`, `mjsw` and `pb`.

Charge storage effects are modelled by the piecewise linear voltages-dependent capacitance model proposed by Meyer. The thin-oxide charge-storage effects are treated slightly different for the level 1 model. These voltage-dependent capacitances are included only if `tox` is specified in the input description and they are represented using Meyer's formulation.

There is some overlap among the parameters describing the junctions, e.g. the reverse current can be input either as `is` (in A) or as `js` (in A/m^2). Whereas the first is an absolute value the second is multiplied by `ad` and `as` to give the reverse current of the drain and source junctions respectively.

Level	Name	Model	Version	Developer	References	Notes
1	MOS1	Shichman-Hodges	-	Berkeley		This is the classical quadratic model.
2	MOS2	Grove-Frhoman	-	Berkeley		Described in [2]
3	MOS3			Berkeley		A semi-empirical model (see [1])
4	BSIM1			Berkeley		Described in [3]
5	BSIM2			Berkeley		Described in [5]
6	MOS6			Berkeley		Described in [2]
8	BSIM3		3.3.0	Berkeley		Described in [13]
9	MOS9			Alan Gillespie		
10	B4SOI		4.3.1	Berkeley		
14	BSIM4		4.6.5	Berkeley		
17	HISIM1					
29	B3SOIPD			Berkeley		
30	B3SOIFD			Berkeley		
31	B3SOIDD			Berkeley		
44	EKV			EPFL		not in the standard distribution
49	BSIM3V1S			Serban Popescu		
50	BSIM3V1			Berkeley		
51	BSIM3V1A			Alan Gillespie		
52	BSIM3V0			Berkeley		
62	STAG			Southampton		

Table 12.1: MOSFET model summary

This methodology has been chosen since there is no sense in relating always junction characteristics with `ad` and `as` entered on the device line; the areas can be defaulted. The same idea applies also to the zero-bias junction capacitances `cbd` and `cbs` (in F) on one hand, and `cj` (in F/m^2) on the other.

The parasitic drain and source series resistance can be expressed as either `rd` and `rs` (in ohms) or `rsh` (in ohms/sq.), the latter being multiplied by the number of squares `nrd` and `nrs` input on the device line.

NGSPICE level 1, 2, 3 and 6 parameters

Name	Parameter	Units	Default	Example
LEVEL	Model index	-	1	
VTO	Zero-bias threshold voltage (V_{T0})	V	0.0	1.0
KP	Transconductance parameter	A/V^2	2.0e-5	3.1e-5
GAMMA	Bulk treshold parameter	$\sqrt{V}$	0.0	0.37
PHI	Sufrace potential (U)	V	0.6	0.65
LAMBDA	Channel length modulation (MOS1 and MOS2 only) (λ)	$1/V$	0.0	0.02
RD	Drain ohmic resistance	Ω	0.0	1.0
RS	Source ohmic resistance	Ω	0.0	1.0
CBD	Zero-bias B-D junction capacitance	F	0.0	20fF
CBS	Zero-bias B-S junction capacitance	F	0.0	20fF
IS	Bulk junction saturation current (I_S)	A	1.0e-14	1.0e-15
PB	Bulk junction potential	V	0.8	0.87
CGSO	Gate-source overlap capacitance per meter channel width	F/m	0.0	4.0e-11
CGDO	Gate-drain overlap capacitance per meter channel width	F/m	0.0	4.0e-11
CGBO	Gate-bulk overlap capacitance per meter channel width	F/m	0.0	2.0e-11
RSH	Drain and source diffusion sheet resistance	$\Omega/\square$	0.0	10
CJ	Zero-bias bulk junction bottom cap. per sq-meter of junction area	F/m^2	0.0	2.0e-4
MJ	Bulk junction bottom grading coeff.	-	0.5	0.5
CJSW	Zero-bias bulk junction sidewall cap. per meter of junction perimeter	F/m	0.0	1.0e-9
MJSW	Bulk junction sidewall grading coeff.	-	0.50 (level1) 0.33 (level2, 3)	
JS	Bulk junction saturation current			
TOX	Oxide thickness	m	1.0e-7	1.0e-7
NSUB	Substrate doping	cm^{-3}	0.0	4.0e15
NSS	Surface state density	cm^{-2}	0.0	1.0e10
NFS	Fast surface state density	cm^{-2}	0.0	1.0e10
TPG	Type of gate material: +1 opp. to substrate, -1 same as substrate, 0 Al gate	-	1.0	
XJ	Metallurgical junction depth	m	0.0	1M

Name	Parameter	Units	Default	Example
LD	Lateral diffusion	m	0.0	0.8M
UO	Surface mobility	$cm^2/V{\cdot}sec$	600	700
UCRIT	Critical field for mobility degradation (MOS2 only)	V/cm	1.0e4	1.0e4
UEXP	Critical field exponent in mobility degradation (MOS2 only)	-	0.0	0.1
UTRA	Transverse field coeff. (mobility) (deleted for MOS2)	-	0.0	0.3
VMAX	Maximum drift velocity of carriers	m/s	0.0	5.0e4
NEFF	Total channel-charge (fixed and mobile) coefficient (MOS2 only)	-	1.0	5.0
KF	Flicker noise coefficient	-	0.0	1.0e-26
AF	Flicker noise exponent	-	1.0	1.2
FC	Coefficient for forward-bias depletion capacitance formula	-	0.5	
DELTA	Width effect on threshold voltage (MOS2 and MOS3)	-	0.0	1.0
THETA	Mobility modulation (MOS3 only)	$1/V$	0.0	0.1
ETA	Static feedback (MOS3 only)	-	0.0	1.0
KAPPA	Saturation field factor (MOS3 only)	-	0.2	0.5
TNOM	Parameter measurement temperature	$°C$	27	50

12.2.6 BSIM Models

Ngspice implments many of the BSIM models developer by Berkeley's device group. BSIM stands for Berkeley Short-Channel IGFET Model and groups a class of models that are continuosly updated. In general, all parameters of BSIM model are obtained from process characterization, in particulat level 4 and level 5 (BSIM1 and BSIM2) parameters are can be generated automatically. J. Pierret [4] describes a means of generating a "process" file, and the program `ngproc2mod` provided with ngspice converts this file into a sequence of BSIM1 ".model" lines suitable for inclusion in an ngspice input file.

Parameters marked below with an `*` in the `l/w` column also have corresponding parameters with a length and width dependency. For example, `vfb` is the basic parameter with units of Volts, and `lvfb` and `wvfb` also exist and have units of Volt-meter.

The formula

$$P = P_0 + \frac{P_L}{L_{\text{effective}}} + \frac{P_W}{W_{\text{effective}}} \tag{12.1}$$

is used to evaluate the parameter for the actual device specified with

$$L_{\text{effective}} = L_{\text{input}} - DL \tag{12.2}$$

$$W_{\text{effective}} = W_{\text{input}} - DW \tag{12.3}$$

Note that unlike the other models in ngspice, the BSIM models are designed for use with a process characterization system that provides all the parameters, thus there are no defaults for the parameters, and leaving one out is considered an error. For an example set of parameters and the format of a process file, see the SPICE2 implementation notes[3]. For more information on BSIM2, see reference [5].

12.2.7 BSIM1 model (level 4)

BSIM1 model (the first is a long series) is an empirical model. Developers placed less emphasis on device physics and based the model on parametrical polynomial equations to model the various physical effects.

This approach pays in terms of circuit simulation behaviour but the accuracy degrades in tghe submicron region. A k nown problem of this model is the negative output conductance and the convergence problems, both related to poor behavior of the polynomial equations.

Ngspice BSIM (level 4) parameters

Name	Parameter	Units	l/w
VFB	Flat-band voltage	V	*
PHI	Surface inversion potential	V	*
K1	Body effect coefficient	$\sqrt{V}$	*
K2	Drain/source depletion charge-sharing coefficient	-	*
ETA	Zero-bias drain-induced barrier-lowering coefficient	-	*
MUZ	Zero-bias mobility	$cm^2/V \cdot sec$	
DL	Shortening of channel	μm	
DW	Narrowing of channel	μm	
U0	Zero-bias transverse-field mobility degradation coefficient	$1/V$	*
U1	Zero-bias velocity saturation coefficient	μ/V	*
X2MZ	Sens. of mobility to substrate bias at v=0	$cm^2/V^2 \cdot sec$	*
X2E	Sens. of drain-induced barrier lowering effect to substrate bias	$1/V$	*
X3E	Sens. of drain-induced barrier lowering effect to drain bias at $V_{ds} = V_{dd}$	$1/V$	*
X2U0	Sens. of transverse field mobility degradation effect to substrate bias	$1/V^2$	*
X2U1	Sens. of velocity saturation effect to substrate bias	$\mu m/V^2$	*
MUS	Mobility at zero substrate bias and at $V_{ds} = V_{dd}$	$cm^2/V^2 sec$	
X2MS	Sens. of mobility to substrate bias at $V_{ds} = V_{dd}$	$cm^2/V^2 sec$	*
X3MS	Sens. of mobility to drain bias at $V_{ds} = V_{dd}$	$cm^2/V^2 sec$	*
X3U1	Sens. of velocity saturation effect on drain bias at Vds=Vdd	$\mu m/V^2$	*
TOX	Gate oxide thickness	μm	
TEMP	Temperature at which parameters were measured	$^\circ C$	
VDD	Measurement bias range	V	
CGDO	Gate-drain overlap capacitance per meter channel width	F/m	
CGSO	Gate-source overlap capacitance per meter channel width	F/m	
CGBO	Gate-bulk overlap capacitance per meter channel length	F/m	
XPART	Gate-oxide capacitance-charge model flag	-	
N0	Zero-bias subthreshold slope coefficient	-	*
NB	Sens. of subthreshold slope to substrate bias	-	*
ND	Sens. of subthreshold slope to drain bias	-	*
RSH	Drain and source diffusion sheet resistance	$\Omega/\square$	
JS	Source drain junction current density	A/m^2	
PB	Built in potential of source drain junction	V	
MJ	Grading coefficient of source drain junction	-	
PBSW	Built in potential of source, drain junction sidewall	V	
MJSW	Grading coefficient of source drain junction sidewall	-	
CJ	Source drain junction capacitance per unit area	F/m^2	
CJSW	source drain junction sidewall capacitance per unit length	F/m	
WDF	Source drain junction default width	m	
DELL	Source drain junction length reduction	m	

`xpart` = 0 selects a 40/60 drain/source charge partition in saturation, while `xpart`=1 selects a 0/100 drain/source charge partition. `nd`, `ng`, and `ns` are the drain, gate, and source nodes, respectively. `mname` is the model name, `area` is the area factor, and `off` indicates an (optional) initial condition on the device for dc analysis. If the area factor is omitted, a value of 1.0 is assumed. The (optional) initial condition specification, using `ic=vds,vgs` is intended for use with the `uic` option on the `.tran` control line, when a transient analysis is desired starting from other than the quiescent operating point. See the `.ic` control line for a better way to set initial conditions.

12.2.8   BSIM2 model (level 5)

This model contains many improvements over BSIM1 and is suitable for analog simulation. Nevertheless, even BSIM2 breaks transistor operation into several distinct regions and this leads to discontinuities in the first derivative in C-V and I-V chracteristics that can cause numerical problems during simulation.

12.2.9   BSIM3 model (levels 8, 49)

BSIM3 solves the numerical problems of previous models with the introduction of smoothing functions. It adopts a single equation to describe device characteristics in the operating regions. This approach eliminates the discontinuities in the I-V and C-V characteristics. The original model, BSIM3 evolved through three versions: BSIM3v1, BSIM3v2 and BSIM3v3. Both BSIM3v1 and BSIM3v2 had suffered from many mathematical problems and were replaced by BSIM3v3. The latter is the only surviving release and has itself a long revision history

The following table summarizes the story of this model:

Release	Date	Notes
BSIM3v3.0	10/30/1995	
BSIM3v3.1	12/09/1996	
BSIM3v3.2	06/16/1998	Two minor revisions available: BSIM3v3.2.1 and BSIM3v3.2.2
BSIM3v3.3	07/29/2005	Parallel processing with OpenMP is available for this model.

BSIM3v2 and 3v3 models has proven for accurate use in 0.18 μm technologies. The model is publicy available in source code form from University of California, Berkeley at

http://www-device.eecs.berkeley.edu/~bsim3/get.html.

A detailed description is given in the user's manual available at

http://www-device.eecs.berkeley.edu/~bsim3/ftpv330/Mod_doc/b3v33manu.tar.

12.2.10   BSIM4 model (levels 14, 54)

This is the newest class of the BSIM family and introduces noise modeling and extrinsic parasitics.. BSIM4, as the extension of BSIM3 model, addresses the MOSFET physical effects into sub-100nm regime. It is a physics-based, accurate, scalable, robust and predictive MOSFET SPICE model for circuit simulation and CMOS technology development. It is developed by the BSIM Research Group in the Department of Electrical Engineering and Computer Sciences (EECS) at the University of California, Berkeley (see http://www-device.eecs.berkeley.edu/~bsim3/bsim4_get.html). BSIM4 has a long revision history, which is summarized below.

Release	Date	Notes
BSIM4.0.0	03/24/2000	
BSIM4.1.0	10/11/2000	
BSIM4.2.0	04/06/2001	
BSIM4.2.1	10/05/2001	*
BSIM4.3.0	05/09/2003	*
BSIM4.4.0	03/04/2004	*
BSIM4.5.0	07/29/2005	*
BSIM4.6.0	12/13/2006	
...		
BSIM4.6.5	09/09/2009	* **

*) supported in ngspice, using e.g. the `version=4.6` flag in the parameter file.

**) Parallel processing using OpenMP support is available for this model.

Details of any revision are to be found in the user's manual, a pdf download from
http://www-device.eecs.berkeley.edu/∼bsim3/BSIM4/BSIM464/BSIM464_Manual.pdf.

12.2.11   EKV model

Level 44 model (EKV) is not available in the standard distribution since it is not released in source form
by the EKV group. To obtain the code please refer to the (EKV model home page, EKV group home
page).

12.2.12   BSIMSOI models (levels 10, 58, 55, 56, 57)

BSIMSOI is a SPICE compact model for SOI (Silicon-On-Insulator) circuit design. This model is formu-
lated on top of the BSIM3 framework. It shares the same basic equations with the bulk model so that the
physical nature and smoothness of BSIM3v3 are retained. Four models are supported in ngspice, those
based on BSIM3 and modelling fully depleted (FD, level 55), partially depleted (PD, level 57) and both
(DD, level 56), as well as the modern BSIMSOI version 4 model (levels 10, 58). Detailed descriptions
are beyond the scope of this manual, but see e.g. BSIMSOI_4.3.1_Users_manual for a very extensive
description of the recent model version. OpenMP support is available for levels 10, 58, version 4.3.1.

12.2.13   SOI3 model (level 62)

see literature citation [18] for a decription.

Chapter 13

Behavioral Modeling

Ngspice implements XSPICE extensions for behavioral modeling. In the XSPICE framework this is referred to as code level modeling. This chapter describes the predefined models available in ngspice, stemming from the original XSPICE simulator. The instructions for writing new code models are given in chapter 27.

To make use of the XSPICE extensions, you need to compile them in. LINX, CYGWIN, MINGW and other users may add the flag `--enable-xspice` to their `./configure` command and then recompile. The prebuild ngspice for Windows distribution has XSPICE already enabled. For detailed compiling instructions see chapter 31.1.

13.1 Code Model Element & .MODEL Cards

Ngspice includes a library of predefined "Code Models" that can be placed within any circuit description in a manner similar to that used to place standard device models. Code model instance cards always begin with the letter "A", and always make use of a .MODEL card to describe the code model desired. Section of this document goes into greater detail as to how a code model similar to the predefined models may be developed, but once any model is created and linked into the simulator it may be placed using one instance card and one .MODEL card (note here we conform to the SPICE custom of referring to a single logical line of information as a "card"). As an example, the following uses the predefined "gain" code model which takes as an input some value on node 1, multiplies it by a gain of 5.0, and outputs the new value to node 2. Note that, by convention, input ports are specified first on code models. Output ports follow the inputs.

```
Example:

a1 1 2 amp
.model amp gain(gain=5.0)
```

In this example the numerical values picked up from single-ended (i.e. ground referenced) input node 1 and output to single-ended output node 2 will be voltages, since in the Interface Specification File for this code model (i.e., gain), the default port type is specified as a voltage (more on this later). However, if you didn't know this, the following modifications to the instance card could be used to insure it:

```
Example:

a1 %v(1) %v(2) amp
.model amp gain(gain=5.0)
```

The specification "%v" preceding the input and output node numbers of the instance card indicate to the simulator that the inputs to the model should be single-ended voltage values. Other possibilities exist, as described later.

Some of the other features of the instance and .MODEL cards are worth noting. Of particular interest is the portion of the .MODEL card which specifies `gain=5.0`. This portion of the card assigns a value to a parameter of the "gain" model. There are other parameters which can be assigned values for this model, and in general code models will have several. In addition to numeric values, code model parameters can take non-numeric values (such as TRUE and FALSE), and even vector values. All of these topics will be discussed at length in the following pages. In general, however, the instance and .MODEL cards which define a code model will follow the abstract form described below. This form illustrates that the number of inputs and outputs and the number of parameters which can be specified is relatively open-ended and can be interpreted in a variety of ways (note that angle-brackets "<" and ">" enclose optional inputs):

```
Example:
AXXXXXXX <%v,%i,%vd,%id,%g,%gd,%h,%hd, or %d>
+ <[> <~><%v,%i,%vd,%id,%g,%gd,%h,%hd, or %d>
+ <NIN1 or +NIN1 -NIN1 or "null">
+ <~>...<NIN2.. <]> >
+ <%v,%i,%vd,%id,%g,%gd,%h,%hd,%d or %vname>
+ <[> <~><%v,%i,%vd,%id,%g,%gd,%h,%hd, or %d><NOUT1 or +NOUT1 -NOUT1>
+ <~>...<NOUT2.. <]>>
+ MODELNAME

.MODEL MODELNAME MODELTYPE
+ <( PARAMNAME1= <[> VAL1 <VAL2... <]>> PARAMNAME2..>)>
```

Square brackets ([ ]) are used to enclose vector input nodes. In addition, these brackets are used to delineate vectors of parameters.

The literal string "null", when included in a node list, is interpreted as no connection at that input to the model. "Null" is not allowed as the name of a model's input or output if the model only has one input or one output. Also, "null" should only be used to indicate a missing connection for a code model; use on other XSPICE component is not interpreted as a missing connection, but will be interpreted as an actual node name.

The tilde, "~", when prepended to a digital node name, specifies that the logical value of that node be inverted prior to being passed to the code model. This allows for simple inversion of input and output polarities of a digital model in order to handle logically equivalent cases and others that frequently arise in digital system design. The following example defines a NAND gate, one input of which is inverted:

```
a1 [~1 2] 3 nand1
.model nand1 d_nand (rise_delay=0.1 fall_delay=0.2)
```

The optional symbols %v, %i, %vd, etc. specify the type of port the simulator is to expect for the subsequent port or port vector. The meaning of each symbol is given in Table 13.1.

The symbols described in Table 13.1 may be omitted if the default port type for the model is desired. Note that non-default port types for multi-input or multi-output (vector) ports must be specified by placing one of the symbols in front of EACH vector port. On the other hand, if all ports of a vector port are to be declared as having the same non-default type, then a symbol may be specified immediately prior to the opening bracket of the vector. The following examples should make this clear:

```
Example 1: - Specifies two differential voltage connections, one
             to nodes 1 & 2, and one to nodes 3 & 4.

   %vd [1 2 3 4]
Example 2: - Specifies two single-ended connections to node 1 and
             at node 2, and one differential connection to
             nodes 3 & 4.

   %v [1 2 %vd 3 4]
```

Port Type Modifiers	
Modifier	Interpretation
%v	represents a single-ended voltage port - one node name or number is expected for each port.
%i	represents a single-ended current port - one node name or number is expected for each port.
%g	represents a single-ended voltage-input, current-output (VCCS) port - one node name or number is expected for each port. This type of port is automatically an input/output.
%h	represents a single-ended current-input, voltage-output (CCVS) port - one node name or number is expected for each port. This type of port is automatically an input/output.
%d	represents a digital port - one node name or number is expected for each port. This type of port may be either an input or an output.
%vnam	epresents the name of a voltage source, the current through which is taken as an input. This notation is provided primarily in order to allow models defined using SPICE2G6 syntax to operate properly in XSPICE.
%vd	represents a differential voltage port - two node names or numbers are expected for each port.
%id	represents a differential current port - two node names or numbers are expected for each port.
%gd	represents a differential VCCS port - two node names or numbers are expected for each port.
%hd	represents a differential CCVS port - two node names or numbers are expected for each port.

Table 13.1: Port Type Modifiers

```
Example 3: - Identical to the previous example...parenthesis
              are added for additional clarity.
    %v [1 2 %vd(3 4)]
Example 4: - Specifies that the node numbers are to be treated in the
              default fashion for the particular model.
              If this model had "%v" as a default for this
              port, then this notation would represent four single-ended
              voltage connections.


    [1 2 3 4]
```

The parameter names listed on the .MODEL card must be identical to those named in the code model itself. The parameters for each predefined code model are described in detail in Sections 13.2 (analog), 13.3 (Hybrid, A/D) and 13.4 (digital) . The steps required in order to specify parameters for user-defined models are described in Chapter 27.

   The following is a list of instance card and associated .MODEL card examples showing use of predefined models within an XSPICE deck:

```
a1 1 2 amp
.model amp gain(in_offset=0.1 gain=5.0 out_offset=-0.01)
a2 %i[1 2] 3 sum1
.model sum1 summer(in_offset=[0.1 -0.2] in_gain=[2.0 1.0]
+ out_gain=5.0 out_offset=-0.01)
a21 %i[1 %vd(2 5) 7 10] 3 sum2
.model sum2 summer(out_gain=10.0)
a5 1 2 limit5 .model limit5 limit(in_offset=0.1 gain=2.5 out_lower.limit=-5.0
+ out_upper_limit=5.0 limit_domain=0.10
+ fraction=FALSE)
a7 2 %id(4 7) xfer.cntl1
.model xfer_cntl1 pwl(x_array=[-2.0 -1.0 2.0 4.0 5.0]
+ y_array=[-0.2 -0.2 0.1 2.0 10.0]
+ input_domain=0.05 fraction=TRUE)
a8 3 %gd(6 7) switch3
.model switch3 aswitch(cntl_off=0.0 cntl_on=5.0 r_off=1e6
+ r_on=10.0 log=TRUE)
```

13.2   Analog Models

The following analog models are supplied with XSPICE. The descriptions included consist of the model Interface Specification File and a description of the model's operation. This is followed by an example of a simulator-deck placement of the model, including the .MODEL card and the specification of all available parameters.

13.2.1   Gain

```
NAME_TABLE:
C_Function_Name:     cm_gain
Spice_Model_Name:    gain
Description:         "A simple gain block"

PORT_TABLE:
Port Name:           in            out
Description:         "input"       "output"
Direction:           in            out
Default_Type:        v             v
Allowed_Types:       [v,vd,i,id]   [v,vd,i,id]
Vector:              no            no
Vector.Bounds:       -             -
Null.Allowed:        no            no
```

```
PARAMETER_TABLE:
Parameter_Name:      in_offset         gain       out_offset
Description:         "input offset"    "gain"     "output offset"
Data_Type:           real              real       real
Default_Value:       0.0               1.0        0.0
Limits:              -                 -          -
Vector:              no                no         no
Vector_Bounds:       -                 -          -
Null_Allowed:        yes               yes        yes
```

Description: This function is a simple gain block with optional offsets on the input and the output. The input offset is added to the input, the sum is then multiplied by the gain, and the result is produced by adding the output offset. This model will operate in DC, AC, and Transient analysis modes.

Example:

```
a1 1 2 amp
.model amp gain(in_offset=0.1 gain=5.0
+ out_offset=-0.01)
```

13.2.2 Summer

```
NAME_TABLE:
C_Function_Name:     cm_summer
Spice_Model_Name:    summer
Description:         "A summer block"

PORT_TABLE:
Port Name:           in                out
Description:         "input vector"    "output"
Direction:           in                out
Default_Type:        v                 v
Allowed_Types:       [v,vd,i,id]       [v,vd,i,id]
Vector:              yes               no
Vector_Bounds:       -                 -
Null_Allowed:        no                no

PARAMETER_TABLE:
Parameter_Name:      in_offset              in_gain
Description:         "input offset vector"  "input gain vector"
Data_Type:           real                   real
Default_Value:       0.0                    1.0
Limits:              -                      -
Vector:              yes                    yes
Vector_Bounds:       in                     in
Null_Allowed:        yes                    yes

PARAMETER_TABLE:
Parameter_Name:      out_gain               out_offset
Description:         "output gain"          "output offset"
Data_Type:           real                   real
Default_Value:       1.0                    0.0
Limits:              -                      -
Vector:              no                     no
```

```
Vector_Bounds:          -                          -
Null_Allowed:           yes                        yes
```

Description: This function is a summer block with 2-to-N input ports. Individual gains and offsets can be applied to each input and to the output. Each input is added to its respective offset and then multiplied by its gain. The results are then summed, multiplied by the output gain and added to the output offset. This model will operate in DC, AC, and Transient analysis modes.

Example usage:

```
a2 [1 2] 3 sum1
.model sum1 summer(in_offset=[0.1 -0.2] in_gain=[2.0 1.0]
+ out_gain=5.0 out_offset=-0.01)
```

13.2.3   Multiplier

```
NAME_TABLE:
C_Function_Name:     cm_mult
Spice_Model_Name:    mult
Description:         "multiplier block"
PORT_TABLE:
Port_Name:           in                     out
Description:         "input vector"         "output"
Direction:           in                     out
Default_Type:        v                      v
Allowed_Types:       [v,vd,i,id]            [v,vd,i,id]
Vector:              yes                    no
Vector_Bounds:       [2 -]                  -
Null_Allowed:        no                     no
PARAMETER_TABLE:
Parameter_Name:      in_offset              in_gain
Description:         "input offset vector"  "input gain vector"
Data_Type:           real                   real
Default_Value:       0.0                    1.0
Limits:              -                      -
Vector:              yes                    yes
Vector_Bounds:       in                     in
Null_Allowed:        yes                    yes
PARAMETER_TABLE:
Parameter_Name:      out_gain               out_offset
Description:         "output gain"          "output offset"
Data_Type:           real                   real
Default_Value:       1.0                    0.0
Limits:              -                      -
Vector:              no                     no
Vector_Bounds:       -                      -
Null_Allowed:        yes                    yes
```

Description: This function is a multiplier block with 2-to-N input ports. Individual gains and offsets can be applied to each input and to the output. Each input is added to its respective offset and then multiplied by its gain. The results are multiplied along with the output gain and are added to the output offset. This model will operate in DC, AC, and Transient analysis modes. However, in ac analysis it is important to remember that results are invalid unless only ONE INPUT of the multiplier is connected to a node which bears an AC signal (this is exemplified by the use of a multiplier to perform a potentiometer function: one input is DC, the other carries the AC signal).

Example SPICE Usage:

```
a3 [1 2 3] 4 sigmult
.model sigmult mult(in_offset=[0.1 0.1 -0.1] in_gain=[10.0 10.0
+ 10.0] out_gain=5.0 out_offset=0.05)
```

13.2.4 Divider

```
NAME_TABLE:
C_Function_Name:      cm_divide
Spice_Model_Name:     divide
Description:          "divider block"
PORT_TABLE:
Port_Name:           num                  den                  out
Description:         "numerator"          "denominator"        "output"
Direction:           in                   in                   out
Default_Type:        v                    v                    v
Allowed_Types:       [v,vd,i,id,vnam]     [v,vd,i,id,vnam]     [v,vd,i,id]
Vector:              no                   no                   no
Vector_Bounds:       -                    -                    -
Null_Allowed:        no                   no                   no
PARAMETER_TABLE:
Parameter_Name:      num_offset           num_gain
Description:         "numerator offset"   "numerator gain"
Data_Type:           real                 real
Default_Value:       0.0                  1.0
Limits:              -                    -
Vector:              no                   no
Vector_Bounds:       -                    -
Null_Allowed:        yes                  yes
PARAMETER_TABLE:
Parameter_Name:      den_offset           den_gain
Description:         "denominator offset" "denominator gain"
Data_Type:           real                 real
Default_Value:       0.0                  1.0
Limits:              -                    -
Vector:              no                   no
Vector_Bounds:       -                    -
Null_Allowed:        yes                  yes
PARAMETER_TABLE:
Parameter_Name:      den_lower_limit
Description:         "denominator lower limit"
Data_Type:           real
Default_Value:       1.0e-10
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
PARAMETER_TABLE:
Parameter_Name:      den_domain
Description:         "denominator smoothing domain"
Data_Type:           real
Default_Value:       1.0e-10
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
```

```
PARAMETER_TABLE:
Parameter_Name:      fraction
Description:         "smoothing fraction/absolute value switch"
Data_Type:           boolean
Default_Value:       false
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
PARAMETER_TABLE:
Parameter_Name:      out_gain              out_offset
Description:         "output gain"        "output offset"
Data_Type:           real                  real
Default_Value:       1.0                   0.0
Limits:              -                     -
Vector:              no                    no
Vector_Bounds:       -                     -
Null_Allowed:        yes                   yes
```

Description: This function is a two-quadrant divider. It takes two inputs; num (numerator) and den (denominator). Divide offsets its inputs, multiplies them by their respective gains, divides the results, multiplies the quotient by the output gain, and offsets the result. The denominator is limited to a value above zero via a user specified lower limit. This limit is approached through a quadratic smoothing function, the domain of which may be specified as a fraction of the lower limit value (default), or as an absolute value. This model will operate in DC, AC and Transient analysis modes. However, in ac analysis it is important to remember that results are invalid unless only ONE INPUT of the divider is connected to a node which bears an AC signal (this is exemplified by the use of the divider to perform a potentiometer function: one input is DC, the other carries the AC signal).

```
Example SPICE Usage:
a4 1 2 4 divider
.model divider divide(num_offset=0.1 num_gain=2.5 den_offset=-0.1
+ den_gain=5.0 den_lower.limit=1e-5 den_domain=1e-6
+ fraction=FALSE out_gain=1.0 out_offset=0.0)
```

13.2.5   Limiter

```
NAME_TABLE:
C_Function_Name:     cm_limit
Spice_Model_Name:    limit
Description:         "limit block"
PORT_TABLE:
Port Name:           in                    out
Description:         "input"              "output"
Direction:           in                    out
Default_Type:        v                     v
Allowed_Types:       [v,vd,i,id]           [v,vd,i,id]
Vector:              no                    no
Vector_Bounds:       -                     -
Null_Allowed:        no                    no
PARAMETER_TABLE:
Parameter_Name:      in_offset             gain
Description:         "input offset"       "gain"
Data_Type:           real                  real
Default_Value:       0.0                   1.0
Limits:              -                     -
Vector:              no                    no
Vector_Bounds:       -                     -
```

```
Null_Allowed:       yes                 yes
PARAMETER_TABLE:
Parameter_Name:     out_lower_limit     out_upper_limit
Description:        "output lower limit" "output upper limit"
Data_Type:          real                real
Default_Value:      0.0                 1.0
Limits:             -                   -
Vector:             no                  no
Vector_Bounds:      -                   -
Null_Allowed:       yes                 yes
PARAMETER_TABLE:
Parameter_Name:     limit_range
Description:        "upper & lower smoothing range"
Data_Type:          real
Default_Value:      1.0e-6
Limits:             -
Vector:             no
Vector_Bounds:      -
Null_Allowed:       yes
PARAMETER_TABLE:
Parameter_Name:     fraction
Description:        "smoothing fraction/absolute value switch"
Data_Type:          boolean
Default_Value:      FALSE
Limits:             -
Vector:             no
Vector_Bounds:      -
Null_Allowed:       yes
```

Description: The Limiter is a single input, single output function similar to the Gain Block. However, the output of the Limiter function is restricted to the range specified by the output lower and upper limits. This model will operate in DC, AC and Transient analysis modes. Note that the limit range is the value BELOW THE UPPER LIMIT AND ABOVE THE LOWER LIMIT at which smoothing of the output begins. For this model, then, the limit range represents the delta WITH RESPECT TO THE OUTPUT LEVEL at which smoothing occurs. Thus, for an input gain of 2.0 and output limits of 1.0 and -1.0 volts, the output will begin to smooth out at ± 0.9 volts, which occurs when the input value is at ± 0.4.

```
Example SPICE Usage:
a5 1 2 limit5
.model limit5 limit(in_offset=0.1 gain=2.5 out_lower_limit=-5.0
+ out_upper_limit=5.0 limit_range=0.10 fraction=FALSE)
```

13.2.6   Controlled Limiter

```
NAME_TABLE:
C_Function_Name:    cm_climit
Spice_Model_Name:   climit
Description:        "controlled limiter block"
PORT_TABLE:
Port_Name:          in                  cntl_upper
Description:        "input"             "upper lim. control input"
Direction:          in                  in
Default_Type:       v                   v
Allowed_Types:      [v,vd,i,id,vnam]    [v,vd,i,id,vnam]
Vector:             no                  no
Vector_Bounds:      -                   -
Null_Allowed:       no                  no
PORT_TABLE:
```

```
Port_Name:           cntl_lower                      out
Description:         "lower limit control input"  "output"
Direction:           in                              out
Default_Type:        v                               v
Allowed_Types:       [v,vd,i,id,vnam]                [v,vd,i,id]
Vector:              no                              no
Vector_Bounds:       -                               -
Null_Allowed:        no                              no
PARAMETER_TABLE:
Parameter_Name:      in_offset           gain
Description:         "input offset"      "gain"
Data_Type:           real                real
Default_Value:       0.0                 1.0
Limits:              -                   -
Vector:              no                  no
Vector_Bounds:       -                   -
Null_Allowed:        yes                 yes
PARAMETER_TABLE:
Parameter_Name:      upper_delta         lower_delta
Description:         "output upper delta"  "output lower delta"
Data_Type:           real                real
Default_Value:       0.0                 0.0
Limits:              -                   -
Vector:              no                  no
Vector_Bounds:       -                   -
Null_Allowed:        yes                 yes
PARAMETER_TABLE:
Parameter_Name:      limit_range                     fraction
Description:         "upper & lower sm. range"     "smoothing %/abs switch"
Data_Type:           real                            boolean
Default_Value:       1.0e-6                          FALSE
Limits:              -                               -
Vector:              no                              no
Vector_Bounds:       -                               -
Null_Allowed:        yes                             yes
```

Description: The Controlled Limiter is a single input, single output function similar to the Gain Block. However, the output of the Limiter function is restricted to the range specified by the output lower and upper limits. This model will operate in DC, AC, and Transient analysis modes. Note that the limit range is the value BELOW THE CNTL_UPPER LIMIT AND ABOVE THE CNTL_LOWER LIMIT at which smoothing of the output begins (minimum positive value of voltage must exist between the CNTL_UPPER input and the CNTL_LOWER input at all times). For this model, then, the limit range represents the delta WITH RESPECT TO THE OUTPUT LEVEL at which smoothing occurs. Thus, for an input gain of 2.0 and output limits of 1.0 and -1.0 volts, the output will begin to smooth out at ±0.9 volts, which occurs when the input value is at ±0.4. Note also that the Controlled Limiter code tests the input values of cntl_lower and cntl_upper to make sure that they are spaced far enough apart to guarantee the existence of a linear range between them. The range is calculated as the difference between (cntl_upper - upper_delta - limit_range) and (cntl_lower + lower_delta + limit_range) and must be greater than or equal to zero. Note that when the limit range is specified as a fractional value, the limit range used in the above is taken as the calculated fraction of the difference between cntl upper and cntl lower. Still, the potential exists for too great a limit range value to be specified for proper operation, in which case the model will return an error message.

```
Example SPICE Usage:
a6 3 6 8 4 varlimit
.
.
.model varlimit climit(in_offset=0.1 gain=2.5 upper_delta=0.0
```

```
+ lower_delta=0.0 limit_range=0.10 fraction=FALSE)
```

13.2.7 PWL Controlled Source

```
NAME_TABLE:
C_Function_Name:      cm_pwl
Spice_Model_Name:     pwl
Description:          "piecewise linear controlled source"
PORT_TABLE:
Port_Name:            in                     out
Description:          "input"                "output"
Direction:            in                     out
Default_Type:         v                      v
Allowed_Types:        [v,vd,i,id,vnam]       [v,vd,i,id]
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         no                     no
PARAMETER_TABLE:
Parameter_Name:       x_array                y_array
Description:          "x-element array"      "y-element array"
Data_Type:            real                   real
Default_Value:        -                      -
Limits:               -                      -
Vector:               yes                    yes
Vector_Bounds:        [2 -]                  [2 -]
Null_Allowed:         no                     no
PARAMETER_TABLE:
Parameter_Name:       input_domain           fraction
Description:          "input sm. domain"     "smoothing %/abs switch"
Data_Type:            real                   boolean
Default_Value:        0.01                   TRUE
Limits:               [1e-12 0.5]            -
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
STATIC_VAR_TABLE:
Static_Var_Name:      last_x_value
Data_Type:            pointer Description:   "iteration holding variable for limiting"
```

Description: The Piece-Wise Linear Controlled Source is a single input, single output function similar to the Gain Block. However, the output of the PWL Source is not necessarily linear for all values of input. Instead, it follows an I/O relationship specified by you via the x_array and y_array coordinates. This is detailed below.

The x_array and y_array values represent vectors of coordinate points on the x and y axes, respectively. The x_array values are progressively increasing input coordinate points, and the associated y_array values represent the outputs at those points. There may be as few as two (x_array[n], y_array[n]) pairs specified, or as many as memory and simulation speed allow. This permits you to very finely approximate a non-linear function by capturing multiple input-output coordinate points. Two aspects of the PWL Controlled Source warrant special attention. These are the handling of endpoints and the smoothing of the described transfer function near coordinate points.

In order to fully specify outputs for values of "in" outside of the bounds of the PWL function (i.e., less than x_array[0] or greater than x_array[n], where n is the largest userspecified coordinate index), the PWL Controlled Source model extends the slope found between the lowest two coordinate pairs and the highest two coordinate pairs. This has the effect of making the transfer function completely linear for "in" less than x_array[0] and "in" greater than x_array[n]. It also has the potentially subtle effect of unrealistically causing an output to reach a very large or small value for large inputs. You should thus keep in mind that the PWL Source does not inherently provide a limiting capability.

In order to diminish the potential for nonconvergence of simulations when using the PWL block, a

form of smoothing around the x_array, y_array coordinate points is necessary. This is due to the iterative nature of the simulator and its reliance on smooth first derivatives of transfer functions in order to arrive at a matrix solution. Consequently, the "input_domain" and "fraction" parameters are included to allow you some control over the amount and nature of the smoothing performed.

"Fraction" is a switch that is either TRUE or FALSE. When TRUE (the default setting), the simulator assumes that the specified input domain value is to be interpreted as a fractional figure. Otherwise, it is interpreted as an absolute value. Thus, if fraction=TRUE and input_domain=0.10, The simulator assumes that the smoothing radius about each coordinate point is to be set equal to 10% of the length of either the x_array segment above each coordinate point, or the x_array segment below each coordinate point. The specific segment length chosen will be the smallest of these two for each coordinate point.

On the other hand, if fraction=FALSE and input=0.10, then the simulator will begin smoothing the transfer function at 0.10 volts (or amperes) below each x_array coordinate and will continue the smoothing process for another 0.10 volts (or amperes) above each x_array coordinate point. Since the overlap of smoothing domains is not allowed, checking is done by the model to ensure that the specified input domain value is not excessive.

One subtle consequence of the use of the fraction=TRUE feature of the PWL Controlled Source is that, in certain cases, you may inadvertently create extreme smoothing of functions by choosing inappropriate coordinate value points. This can be demonstrated by considering a function described by three coordinate pairs, such as (-1,-1), (1,1), and (2,1). In this case, with a 10% input_domain value specified (fraction=TRUE, input domain=0.10), you would expect to see rounding occur between in=0.9 and in=1.1, and nowhere else. On the other hand, if you were to specify the same function using the coordinate pairs (-100,-100), (1,1) and (201,1), you would find that rounding occurs between in=-19 and in=21. Clearly in the latter case the smoothing might cause an excessive divergence from the intended linearity above and below in=1.

```
Example SPICE Usage:
a7 2 4 xfer_cntl1
.
.
.model xfer_cntl1 pwl(x_array=[-2.0 -1.0 2.0 4.0 5.0]
+                     y_array=[-0.2 -0.2 0.1 2.0 10.0]
+                     input_domain=0.05 fraction=TRUE)
```

13.2.8   Analog Switch

```
NAME_TABLE:
C_Function_Name:    cm_aswitch
Spice_Model_Name:   aswitch
Description:        "analog switch"
PORT_TABLE:
Port Name:          cntl_in              out
Description:        "input"              "resistive output"
Direction:          in                   out
Default_Type:       v                    gd
Allowed_Types:      [v,vd,i,id]          [gd]
Vector:             no                   no
Vector_Bounds:      -                    -
Null_Allowed:       no                   no
PARAMETER_TABLE:
Parameter_Name:     cntl_off             cntl_on
Description:        "control 'off' value" "control 'on' value"
Data_Type:          real                 real
Default_Value:      0.0                  1.0
Limits:             -                    -
Vector:             no                   no
Vector_Bounds:      -                    -
Null_Allowed:       yes                  yes
PARAMETER_TABLE:
```

```
Parameter_Name:        r_off                   log
Description:           "off resistance"        "log/linear switch"
Data_Type:             real                    boolean
Default_Value:         1.0e12                  TRUE
Limits:                -                       -
Vector:                no                      no
Vector_Bounds:         -                       -
Null_Allowed:          yes                     yes
PARAMETER_TABLE:
Parameter_Name:        r_on
Description:           "on resistance"
Data_Type:             real
Default_Value:         1.0
Limits:                -
Vector:                no
Vector_Bounds:         -
Null_Allowed:          yes
```

Description: The Analog Switch is a resistor that varies either logarithmically or linearly between specified values of a controlling input voltage or current. Note that the input is not internally limited. Therefore, if the controlling signal exceeds the specified OFF state or ON state value, the resistance may become excessively large or excessively small (in the case of logarithmic dependence), or may become negative (in the case of linear dependence). For the experienced user, these excursions may prove valuable for modeling certain devices, but in most cases you are advised to add limiting of the controlling input if the possibility of excessive control value variation exists.

```
Example SPICE Usage:
a8 3 (6 7) switch3
.
.
.model switch3 aswitch(cntl_off=0.0 cntl_on=5.0 r_off=1e6
+                      r_on=10.0 log=TRUE)
```

13.2.9   Zener Diode

```
NAME_TABLE:
C_Function_Name:       cm_zener
Spice_Model_Name:      zener
Description:           "zener diode"
PORT_TABLE:
Port Name:             z
Description:           "zener"
Direction:             inout
Default_Type:          gd
Allowed_Types:         [gd]
Vector:                no
Vector_Bounds:         -
Null_Allowed:          no
PARAMETER_TABLE:
Parameter_Name:        v_breakdown             i_breakdown
Description:           "breakdown voltage"     "breakdown current"
Data_Type:             real                    real
Default_Value:         -                       2.0e-2
Limits:                [1.0e-6 1.0e6]          [1.0e-9 -]
Vector:                no                      no
Vector_Bounds:         -                       -
Null_Allowed:          no                      yes
PARAMETER_TABLE:
Parameter_Name:        i_sat                   n_forward
```

```
Description:            "saturation current"  "forward emission coefficient"
Data_Type:              real                  real
Default_Value:          1.0e-12               1.0
Limits:                 [1.0e-15 -]           [0.1 10]
Vector:                 no                    no
Vector_Bounds:          -                     -
Null_Allowed:           yes                   yes
PARAMETER_TABLE:
Parameter_Name:         limit_switch
Description:            "switch for on-board limiting (convergence aid)"
Data_Type:              boolean
Default_Value:          FALSE
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
STATIC_VAR_TABLE:
Static_Var_Name:        previous_voltage
Data_Type:              pointer
Description:            "iteration holding variable for limiting"
```

Description: The Zener Diode models the DC characteristics of most zeners. This model differs from
the Diode/Rectifier by providing a user-defined dynamic resistance in the reverse breakdown region.
The forward characteristic is defined by only a single point, since most data sheets for zener diodes
do not give detailed characteristics in the forward region.

The first three parameters define the DC characteristics of the zener in the breakdown region and
are usually explicitly given on the data sheet.

The saturation current refers to the relatively constant reverse current that is produced when the
voltage across the zener is negative, but breakdown has not been reached. The reverse leakage
current determines the slight increase in reverse current as the voltage across the zener becomes
more negative. It is modeled as a resistance parallel to the zener with value v breakdown / i rev.

Note that the limit switch parameter engages an internal limiting function for the zener. This can,
in some cases, prevent the simulator from converging to an unrealistic solution if the voltage across
or current into the device is excessive. If use of this feature fails to yield acceptable results, the
convlimit option should be tried (add the following statement to the SPICE input deck: .options
convlimit)

```
Example SPICE Usage:
a9 3 4 vref10
.
.
.model vref10 zener(v_breakdown=10.0 i_breakdown=0.02
+                   r_breakdown=1.0 i_rev=1e-6 i_sat=1e-12)
```

13.2.10   Current Limiter

```
NAME_TABLE:
C_Function_Name:        cm_ilimit
Spice_Model_Name:       ilimit
Description:            "current limiter block"
PORT_TABLE:
Port Name:              in                    pos_pwr
Description:            "input"               "positive power supply"
Direction:              in                    inout
Default_Type:           v                     g
Allowed_Types:          [v,vd]                [g,gd]
Vector:                 no                    no
Vector_Bounds:          -                     -
Null_Allowed:           no                    yes
```

```
PORT_TABLE:
Port Name:           neg_pwr                 out
Description:          "negative power supply" "output"
Direction:           inout                   inout
Default_Type:        g                       g
Allowed_Types:       [g,gd]                  [g,gd]
Vector:              no                      no
Vector_Bounds:       -                       -
Null_Allowed:        yes                     no
PARAMETER_TABLE:
Parameter_Name:      in_offset               gain
Description:         "input offset"          "gain"
Data_Type:           real                    real
Default_Value:       0.0                     1.0
Limits:              -                       -
Vector:              no                      no
Vector_Bounds:       -                       -
Null_Allowed:        yes                     yes
PARAMETER_TABLE:
Parameter_Name:      r_out_source            r_out_sink
Description:         "sourcing resistance"   "sinking resistance"
Data_Type:           real                    real
Default_Value:       1.0                     1.0
Limits:              [1.0e-9 1.0e9]          [1.0e-9 1.0e9]
Vector:              no                      no
Vector_Bounds:       -                       -
Null_Allowed:        yes                     yes
PARAMETER_TABLE:
Parameter_Name:      i_limit_source
Description:         "current sourcing limit"
Data_Type:           real
Default_Value:       -
Limits:              [1.0e-12 -]
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
PARAMETER_TABLE:
Parameter_Name:      i_limit_sink
Description:         "current sinking limit"
Data_Type:           real
Default_Value:       -
Limits:              [1.0e-12 -]
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
PARAMETER_TABLE:
Parameter_Name:      v_pwr_range             i_source_range
Description:         "upper & lower power     "sourcing current
                      supply smoothing range"  smoothing range"
Data_Type:           real                    real
Default_Value:       1.0e-6                  1.0e-9
Limits:              [1.0e-15 -]             [1.0e-15 -]
Vector:              no                      no
Vector_Bounds:       -                       -
Null_Allowed:        yes                     yes
PARAMETER_TABLE:
Parameter_Name:      i_sink_range
Description:         "sinking current smoothing range"
```

```
Data_Type:              real
Default_Value:          1.0e-9
Limits:                 [1.0e-15 -]
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
PARAMETER_TABLE:
Parameter_Name:         r_out_domain
Description:            "internal/external voltage delta smoothing range"
Data_Type:              real
Default_Value:          1.0e-9
Limits:                 [1.0e-15 -]
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
```

Description: The Current Limiter models the behavior of an operational amplifier or comparator device at a high level of abstraction. All of its pins act as inputs; three of the four also act as outputs. The model takes as input a voltage value from the "in" connector. It then applies an offset and a gain, and derives from it an equivalent internal voltage (veq), which it limits to fall between pos pwr and neg pwr. If veq is greater than the output voltage seen on the "out" connector, a sourcing current will flow from the output pin. Conversely, if the voltage is less than vout, a sinking current will flow into the output pin.

Depending on the polarity of the current flow, either a sourcing or a sinking resistance value (r_out_source, r_out_sink) is applied to govern the vout/i_out relationship. The chosen resistance will continue to control the output current until it reaches a maximum value specified by either i_limit_source or i_limit_sink. The latter mimics the current limiting behavior of many operational amplifier output stages.

During all operation, the output current is reflected either in the pos_pwr connector current or the neg_pwr current, depending on the polarity of i_out. Thus, realistic power consumption as seen in the supply rails is included in the model.

The user-specified smoothing parameters relate to model operation as follows: v_pwr_range controls the voltage below vpos_pwr and above vneg_pwr inputs beyond which veq [= gain * (vin + voffset)] is smoothed; i_source_range specifies the current below i_limit_source at which smoothing begins, as well as specifying the current increment above i_out=0.0 at which i_pos_pwr begins to transition to zero; i_sink_range serves the same purpose with respect to i_limit_sink and i_neg_pwr that i_source_range serves for i_limit_source & i_pos_pwr; r_out_domain specifies the incremental value above and below (veq-vout)=0.0 at which r_out will be set to r_out_source and r_out_sink, respectively. For values of (veq- vout) less than r_out_domain and greater than -r_out_domain, r_out is interpolated smoothly between r_out_source & r_out_sink.

```
Example SPICE Usage:
a10 3 10 20 4 amp3
.
.
.model amp3 ilimit(in_offset=0.0 gain=16.0 r_out_source=1.0
+               r_out_sink=1.0 i_limit_source=1e-3
+               i_limit_sink=10e-3 v_pwr_range=0.2
+               i_source_range=1e-6 i_sink_range=1e-6
+               r_out_domain=1e-6)
```

13.2.11   Hysteresis Block

```
NAME_TABLE:
C_Function_Name:        cm_hyst
Spice_Model_Name:       hyst
Description:            "hysteresis block"
PORT_TABLE:
Port Name:              in                      out
```

```
Description:            "input"                 "output"
Direction:              in                      out
Default_Type:           v                       v
Allowed_Types:          [v,vd,i,id]             [v,vd,i,id]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           no                      no
PARAMETER_TABLE:
Parameter_Name:         in_low                  in_high
Description:            "input low value"       "input high value"
Data_Type:              real                    real
Default_Value:          0.0                     1.0
Limits:                 -                       -
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         hyst                    out_lower_limit
Description:            "hysteresis"            "output lower limit"
Data_Type:              real                    real
Default_Value:          0.1                     0.0
Limits:                 [0.0 -]                 -
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         out_upper_limit         input_domain
Description:            "output upper limit"    "input smoothing domain"
Data_Type:              real                    real
Default_Value:          1.0                     0.01
Limits:                 -                       -
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         fraction
Description:            "smoothing fraction/absolute value switch"
Data_Type:              boolean
Default_Value:          TRUE
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
```

Description: The Hysteresis block is a simple buffer stage that provides hysteresis of the output with respect to the input. The in low and in high parameter values specify the center voltage or current inputs about which the hysteresis effect operates. The output values are limited to out lower limit and out upper limit. The value of "hyst" is added to the in low and in high points in order to specify the points at which the slope of the hysteresis function would normally change abruptly as the input transitions from a low to a high value. Likewise, the value of "hyst" is subtracted from the in high and in low values in order to specify the points at which the slope of the hysteresis function would normally change abruptly as the input transitions from a high to a low value. In fact, the slope of the hysteresis function is never allowed to change abruptly but is smoothly varied whenever the input domain smoothing parameter is set greater than zero.

```
Example SPICE Usage:
a11 1 2 schmitt1
.
.
```

```
.model schmitt1 hyst(in_low=0.7 in_high=2.4 hyst=0.5
+                    out_lower_limit=0.5 out_upper_limit=3.0
+                    input_domain=0.01 fraction=TRUE)
```

13.2.12   Differentiator

```
NAME_TABLE:
C_Function_Name:      cm_d_dt
Spice_Model_Name:     d_dt
Description:          "time-derivative block"
PORT_TABLE:
Port Name:            in                       out
Description:          "input"                  "output"
Direction:            in                       out
Default_Type:         v                        v
Allowed_Types:        [v,vd,i,id]              [v,vd,i,id]
Vector:               no                       no
Vector_Bounds:        -                        -
Null_Allowed:         no                       no
PARAMETER_TABLE:
Parameter_Name:       gain                     out_offset
Description:          "gain"                   "output offset"
Data_Type:            real                     real
Default_Value:        1.0                      0.0
Limits:               -                        -
Vector:               no                       no
Vector_Bounds:        -                        -
Null_Allowed:         yes                      yes
PARAMETER_TABLE:
Parameter_Name:       out_lower_limit          out_upper_limit
Description:          "output lower limit"     "output upper limit"
Data_Type:            real                     real
Default_Value:        -                        -
Limits:               -                        -
Vector:               no                       no
Vector_Bounds:        -                        -
Null_Allowed:         yes                      yes
PARAMETER_TABLE:
Parameter_Name:       limit_range
Description:          "upper & lower limit smoothing range"
Data_Type:            real
Default_Value:        1.0e-6
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
```

Description: The Differentiator block is a simple derivative stage that approximates the time derivative of an input signal by calculating the incremental slope of that signal since the previous timepoint. The block also includes gain and output offset parameters to allow for tailoring of the required signal, and output upper and lower limits to prevent convergence errors resulting from excessively large output values. The incremental value of output below the output upper limit and above the output lower limit at which smoothing begins is specified via the limit range parameter. In AC analysis, the value returned is equal to the radian frequency of analysis multiplied by the gain.

Note that since truncation error checking is not included in the d_dt block, it is not recommended that the model be used to provide an integration function through the use of a feedback loop. Such an arrangement could produce erroneous results. Instead, you should make use of the "integrate" model, which does include truncation error checking for enhanced accuracy.

Example SPICE Usage:

```
a12 7 12 slope_gen
.
.
.model slope_gen d_dt(out_offset=0.0 gain=1.0
+                      out_lower_limit=1e-12 out_upper_limit=1e12
+                      limit_range=1e-9)
```

13.2.13   Integrator

```
NAME_TABLE:
C_Function_Name:      cm_int
Spice_Model_Name:     int
Description:          "time-integration block"
PORT_TABLE:
Port Name:            in                     out
Description:          "input"                "output"
Direction:            in                     out
Default_Type:         v                      v
Allowed_Types:        [v,vd,i,id]            [v,vd,i,id]
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         no                     no
PARAMETER_TABLE:
Parameter_Name:       in_offset              gain
Description:          "input offset"         "gain"
Data_Type:            real                   real
Default_Value:        0.0                    1.0
Limits:               -                      -
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       out_lower_limit        out_upper_limit
Description:          "output lower limit"   "output upper limit"
Data_Type:            real                   real
Default_Value:        -                      -
Limits:               -                      -
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       limit_range
Description:          "upper & lower limit smoothing range"
Data_Type:            real
Default_Value:        1.0e-6
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
PARAMETER_TABLE:
Parameter_Name:       out_ic
Description:          "output initial condition"
Data_Type:            real
Default_Value:        0.0
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
```

Description: The Integrator block is a simple integration stage that approximates the integral with respect to time of an input signal. The block also includes gain and input offset parameters to allow for tailoring of the required signal, and output upper and lower limits to prevent convergence errors resulting from excessively large output values. Note that these limits specify integrator behavior similar to that found in an operational amplifier-based integration stage, in that once a limit is reached, additional storage does not occur. Thus, the input of a negative value to an integrator which is currently driving at the out upper limit level will immediately cause a drop in the output, regardless of how long the integrator was previously summing positive inputs. The incremental value of output below the output upper limit and above the output lower limit at which smoothing begins is specified via the limit range parameter. In AC analysis, the value returned is equal to the gain divided by the radian frequency of analysis.

Note that truncation error checking is included in the "int" block. This should provide for a more accurate simulation of the time integration function, since the model will inherently request smaller time increments between simulation points if truncation errors would otherwise be excessive.

```
Example SPICE Usage:
a13 7 12 time_count
.
.
.model time_count int(in_offset=0.0 gain=1.0
+                     out_lower_limit=-1e12 out_upper_limit=1e12
+                     limit_range=1e-9 out_ic=0.0)
```

13.2.14   S-Domain Transfer Function

```
NAME_TABLE:
C_Function_Name:      cm_s_xfer
Spice_Model_Name:     s_xfer
Description:          "s-domain transfer function"
PORT_TABLE:
Port Name:            in                        out
Description:          "input"                   "output"
Direction:            in                        out
Default_Type:         v                         v
Allowed_Types:        [v,vd,i,id]               [v,vd,i,id]
Vector:               no                        no
Vector_Bounds:        -                         -
Null_Allowed:         no                        no
PARAMETER_TABLE:
Parameter_Name:       in_offset                 gain
Description:          "input offset"            "gain"
Data_Type:            real                      real
Default_Value:        0.0                       1.0
Limits:               -                         -
Vector:               no                        no
Vector_Bounds:        -                         -
Null_Allowed:         yes                       yes
PARAMETER_TABLE:
Parameter_Name:       num_coeff
Description:          "numerator polynomial coefficients"
Data_Type:            real
Default_Value:        -
Limits:               -
Vector:               yes
Vector_Bounds:        [1 -]
Null_Allowed:         no
PARAMETER_TABLE:
Parameter_Name:       den_coeff
Description:          "denominator polynomial coefficients"
```

```
          Data_Type:            real
          Default_Value:        -
          Limits:               -
          Vector:               yes
          Vector_Bounds:        [1 -]
          Null_Allowed:         no
          PARAMETER_TABLE:
          Parameter_Name:       int_ic
          Description:           "integrator stage initial conditions"
          Data_Type:            real
          Default_Value:        0.0
          Limits:               -
          Vector:               yes
          Vector_Bounds:        den_coeff
          Null_Allowed:         yes
          PARAMETER_TABLE:
          Parameter_Name:       denormalized_freq
          Description:           "denorm. corner freq.(radians) for 1 rad/s coeffs"
          Data_Type:            real
          Default_Value:        1.0
          Limits:               -
          Vector:               no
          Vector_Bounds:        -
          Null_Allowed:         yes
```

Description: The s-domain transfer function is a single input, single output transfer function in the Laplace transform variable "s" that allows for flexible modulation of the frequencydomain characteristics of a signal. The code model may be configured to produce an arbitrary s-domain transfer function with the following restrictions:

```
   1. The degree of the numerator polynomial cannot exceed that
      of the denominator polynomial in the variable "s".
   2. The coefficients for a polynomial must be stated
      explicitly. That is, if a coefficient is zero, it must be
      included as an input to the num coeff or den coeff vector.
```

The order of the coefficient parameters is from that associated with the highest-powered term decreasing to that of the lowest. Thus, for the coefficient parameters specified below, the equation in "s" is shown:

```
   .model filter s_xfer(gain=0.139713 num_coeff=[1.0 0.0 0.07464102]
   +                             den_coeff=[1.0 0.998942 0.01170077])
          ...specifies a transfer function of the form...
```

$$N(s) = 0.139713 \cdot \left\{ \frac{s^2+0.7464102}{s^2+0.998942s+0.00117077} \right\}$$

The s-domain transfer function includes gain and input offset parameters to allow for tailoring of the required signal. There are no limits on the internal signal values or on the output value of the s-domain transfer function, so you are cautioned to specify gain and coefficient values that will not cause the model to produce excessively large values. In AC analysis, the value returned is equal to the real and imaginary components of the total s-domain transfer function at each frequency of interest.

The denormalized freq term allows you to specify coefficients for a normalized filter (i.e. one in which the frequency of interest is 1 rad/s). Once these coefficients are included, specifying the denormalized frequency value "shifts" the corner frequency to the actual one of interest. As an example, the following transfer function describes a Chebyshev lowpass filter with a corner (passband) frequency of 1 rad/s:

$$N(s) = 0.139713 \cdot \left\{ \frac{1.0}{s^2+1.09773s+1.10251} \right\}$$

In order to define an s_xfer model for the above, but with the corner frequency equal to 1500 rad/s (9425 Hz), the following instance and model lines would be needed:

```
a12 cheby1
.model cheby1 s_xfer(num_coeff=[1] den_coeff=[1 1.09773 1.10251]
+                    denormalized_freq=1500)
```

In the above, you add the normalized coefficients and scales the filter through the use of the denormalized freq parameter. Similar results could have been achieved by performing the denormalization prior to specification of the coefficients, and setting denormalized freq to the value 1.0 (or not specifying the frequency, as the default is 1.0 rad/s) Note in the above that frequencies are ALWAYS SPECIFIED AS RADIANS/SECOND.

Truncation error checking is included in the s-domain transfer block. This should provide for more accurate simulations, since the model will inherently request smaller time increments between simulation points if truncation errors would otherwise be excessive.

```
Example SPICE Usage:
a14 9 22 cheby_LP_3KHz
.
.
.model cheby_LP_3KHz s_xfer(in_offset=0.0 gain=1.0 num_coeff=[1.0]
+                           den_coeff=[1.0 1.42562 1.51620])
```

13.2.15   Slew Rate Block

```
NAME_TABLE:
C_Function_Name:     cm_slew
Spice_Model_Name:    slew
Description:         "A simple slew rate follower block"
PORT_TABLE:
Port Name:           in                    out
Description:         "input"               "output"
Direction:           in                    out
Default_Type:        v                     v
Allowed_Types:       [v,vd,i,id]           [v,vd,i,id]
Vector:              no                    no
Vector_Bounds:       -                     -
Null_Allowed:        no                    no
PARAMETER_TABLE:
Parameter_Name:      rise_slope
Description:         "maximum rising slope value"
Data_Type:           real
Default_Value:       1.0e9
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
PARAMETER_TABLE:
Parameter_Name:      fall_slope
Description:         "maximum falling slope value"
Data_Type:           real
Default_Value:       1.0e9
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
PARAMETER_TABLE:
Parameter_Name:      range
Description:         "smoothing range"
Data_Type:           real
Default_Value:       0.1
Limits:              -
```

```
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
```

Description: This function is a simple slew rate block that limits the absolute slope of the output with respect to time to some maximum or value. The actual slew rate effects of over-driving an amplifier circuit can thus be accurately modeled by cascading the amplifier with this model. The units used to describe the maximum rising and falling slope values are expressed in volts or amperes per second. Thus a desired slew rate of 0.5 V/μs will be expressed as 0.5e+6, etc.

The slew rate block will continue to raise or lower its output until the difference between the input and the output values is zero. Thereafter, it will resume following the input signal, unless the slope again exceeds its rise or fall slope limits. The range input specifies a smoothing region above or below the input value. Whenever the model is slewing and the output comes to within the input + or - the range value, the partial derivative of the output with respect to the input will begin to smoothly transition from 0.0 to 1.0. When the model is no longer slewing (output = input), dout/din will equal 1.0.

```
Example SPICE Usage:
a15 1 2 slew1
.model slew1 slew(rise_slope=0.5e6 fall_slope=0.5e6)
```

13.2.16   Inductive Coupling

```
NAME_TABLE:
C_Function_Name:     cm_lcouple
Spice_Model_Name:    lcouple
Description:         "inductive coupling (for use with 'core' model)"
PORT_TABLE:
Port_Name:           l                    mmf_out
Description:         "inductor"           "mmf output (in ampere-turns)"
Direction:           inout                inout
Default_Type:        hd                   hd
Allowed_Types:       [h,hd]               [hd]
Vector:              no                   no
Vector_Bounds:       -                    -
Null_Allowed:        no                   no
PARAMETER_TABLE:
Parameter_Name:      num_turns
Description:         "number of inductor turns"
Data_Type:           real
Default_Value:       1.0
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
```

Description: This function is a conceptual model which is used as a building block to create a wide variety of inductive and magnetic circuit models. This function is normally used in conjunction with the "core" model, but can also be used with resistors, hysteresis blocks, etc. to build up systems which mock the behavior of linear and nonlinear components.

The lcouple takes as an input (on the "l" port) a current. This current value is multiplied by the num_turns value, N, to produce an output value (a voltage value which appears on the mmf_out port). The mmf_out acts similar to a magnetomotive force in a magnetic circuit; when the lcouple is connected to the "core" model, or to some other resistive device, a current will flow. This current value (which is modulated by whatever the lcouple is connected to) is then used by the lcouple to calculate a voltage "seen" at the "l" port. The voltage is a function of the derivative with respect to time of the current value seen at mmf_out.

The most common use for lcouples will be as a building block in the construction of transformer models. To create a transformer with a single input and a single output, you would require two

lcouple models plus one "core" model. The process of building up such a transformer is described
under the description of the "core" model, below.

```
Example SPICE Usage:
a150 (7 0) (9 10) lcouple1
.model lcouple1 lcouple(num_turns=10.0)
```

13.2.17   Magnetic Core

```
NAME_TABLE:
C_Function_Name:    cm_core
Spice_Model_Name:   core
Description:        "magnetic core"
PORT_TABLE:
Port_Name:          mc
Description:        "magnetic core"
Direction:          inout
Default_Type:       gd
Allowed_Types:      [g,gd]
Vector: no
Vector_Bounds:      -
Null_Allowed:       no
PARAMETER_TABLE:
Parameter_Name:     H_array                 B_array
Description:        "magnetic field array" "flux density array"
Data_Type:          real                    real
Default_Value:      -                       -
Limits:             -                       -
Vector:             yes                     yes
Vector_Bounds:      [2 -]                   [2 -]
Null_Allowed:       no                      no
PARAMETER_TABLE:
Parameter_Name:     area                    length
Description:        "cross-sectional area" "core length"
Data_Type:          real                    real
Default_Value:      -                       -
Limits:             -                       -
Vector:             no                      no
Vector_Bounds:      -                       -
Null_Allowed:       no                      no
PARAMETER_TABLE:
Parameter_Name:     input_domain
Description:        "input sm. domain"
Data_Type:          real
Default_Value:      0.01
Limits:             [1e-12 0.5]
Vector:             no
Vector_Bounds:      -
Null_Allowed:       yes
PARAMETER_TABLE:
Parameter_Name:     fraction
Description:        "smoothing fraction/abs switch"
Data_Type:          boolean
Default_Value:      TRUE
Limits:             -
Vector:             no
Vector_Bounds:      -
Null_Allowed:       yes
PARAMETER_TABLE:
```

```
       Parameter_Name:      mode
       Description:         "mode switch (1 = pwl, 2 = hyst)"
       Data_Type:           int
       Default_Value:       1
       Limits:              [1 2]
       Vector:              no
       Vector_Bounds:       -
       Null_Allowed:        yes
       PARAMETER_TABLE:
       Parameter_Name:      in_low                  in_high
       Description:         "input low value"       "input high value"
       Data_Type:           real                    real
       Default_Value:       0.0                     1.0
       Limits:              -                       -
       Vector:              no                      no
       Vector_Bounds:       -                       -
       Null_Allowed:        yes                     yes
       PARAMETER_TABLE:
       Parameter_Name:      hyst                    out_lower_limit
       Description:         "hysteresis"            "output lower limit"
       Data_Type:           real                    real
       Default_Value:       0.1                     0.0
       Limits:              [0 -]                   -
       Vector:              no                      no
       Vector_Bounds:       -                       -
       Null_Allowed:        yes                     yes
       PARAMETER_TABLE:
       Parameter_Name:      out_upper_limit
       Description:         "output upper limit"
       Data_Type:           real
       Default_Value:       1.0
       Limits:              -
       Vector:              no
       Vector_Bounds:       -
       Null_Allowed:        yes
```

Description: This function is a conceptual model which is used as a building block to create a wide variety of inductive and magnetic circuit models. This function is almost always expected to be used in conjunction with the "lcouple" model to build up systems which mock the behavior of linear and nonlinear magnetic components. There are two fundamental modes of operation for the core model. These are the pwl mode (which is the default, and which is the most likely to be of use to you) and the hysteresis mode. These are detailed below.

<div align="center">PWL Mode (mode = 1)</div>

The core model in PWL mode takes as input a voltage which it treats as a magnetomotive force (mmf) value. This value is divided by the total effective length of the core to produce a value for the Magnetic Field Intensity, H. This value of H is then used to find the corresponding Flux Density, B, using the piecewise linear relationship described by you in the H array / B array coordinate pairs. B is then multiplied by the cross-sectional area of the core to find the Flux value, which is output as a current. The pertinent mathematical equations are listed below:

$$H = mmf = L, \text{ where } L = \text{Length}$$

Here H, the Magnetic Field Intensity, is expressed in ampere-turns/meter.

$$B = f(H)$$

The B value is derived from a piecewise linear transfer function described to the model via the (H_array[],B_array[]) parameter coordinate pairs. This transfer function does not include hysteretic effects; for that, you would need to substitute a HYST model for the core.

$$\phi = \mathrm{BA}, \text{ where A} = \mathrm{Area}$$

The final current allowed to flow through the core is equal to ϕ. This value in turn is used by the "lcouple" code model to obtain a value for the voltage reflected back across its terminals to the driving electrical circuit.

The following example code shows the use of two "lcouple" models and one core model to produce a simple primary/secondary transformer.

```
Example SPICE Usage:
a1 (2 0) (3 0) primary
.model primary lcouple (num_turns = 155)
a2 (3 4) iron_core
.model iron_core core (H_array = [-1000 -500 -375 -250 -188 -125 -63 0
+                                 63 125 188 250 375 500 1000]
+                   B_array = [-3.13e-3 -2.63e-3 -2.33e-3 -1.93e-3
+                               -1.5e-3 -6.25e-4 -2.5e-4 0 2.5e-4
+                               6.25e-4 1.5e-3 1.93e-3 2.33e-3
+                               2.63e-3 3.13e-3]
+                   area = 0.01 length = 0.01)
a3 (5 0) (4 0) secondary
.model secondary lcouple (num_turns = 310)
```

HYSTERESIS Mode (mode = 2)

The core model in HYSTERESIS mode takes as input a voltage which it treats as a magnetomotive force (mmf) value. This value is used as input to the equivalent of a hysteresis code model block. The parameters defining the input low and high values, the output low and high values, and the amount of hysteresis are as in that model. The output from this mode, as in PWL mode, is a current value which is seen across the mc port. An example of the core model used in this fashion is shown below:

```
Example SPICE Usage:
a1 (2 0) (3 0) primary
.model primary lcouple (num_turns = 155)
a2 (3 4) iron_core
.model iron_core core (mode = 2 in_low=-7.0 in_high=7.0
+                     out_lower_limit=-2.5e-4 out_upper_limit=2.5e-4
+                     hyst = 2.3 )
a3 (5 0) (4 0) secondary
.model secondary lcouple (num_turns = 310)
```

One final note to be made about the two core model nodes is that certain parameters are available in one mode, but not in the other. In particular, the in_low, in_high, out_lower_limit, out_upper_limit, and hysteresis parameters are not available in PWL mode. Likewise, the H_array, B_array, area, and length values are unavailable in HYSTERESIS mode. The input domain and fraction parameters are common to both modes (though their behavior is somewhat different; for explanation of the input domain and fraction values for the HYSTERESIS mode, you should refer to the hysteresis code model discussion).

13.2.18   Controlled Sine Wave Oscillator

```
NAME_TABLE:
C_Function_Name:      cm_sine
Spice_Model_Name:     sine
Description:          "controlled sine wave oscillator"
PORT_TABLE:
Port Name:            cntl_in                  out
Description:          "control input"          "output"
Direction:            in                       out
Default_Type:         v                        v
Allowed_Types:        [v,vd,i,id]              [v,vd,i,id]
Vector:               no                       no
```

```
Vector_Bounds:          -                       -
Null_Allowed:           no                      no
PARAMETER_TABLE:
Parameter_Name:         cntl_array              freq_array
Description:            "control array"         "frequency array"
Data_Type:              real                    real
Default_Value:          0.0                     1.0e3
Limits:                 -                       [0 -]
Vector:                 yes                     yes
Vector_Bounds:          [2 -]                   cntl_array
Null_Allowed:           no                      no
PARAMETER_TABLE:
Parameter_Name:         out_low                 out_high
Description:            "output peak low value" "output peak high value"
Data_Type:              real                    real
Default_Value:          -1.0                    1.0
Limits:                 -                       -
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
```

Description: This function is a controlled sine wave oscillator with parameterizable values of low and high peak output. It takes an input voltage or current value. This value is used as the independent variable in the piecewise linear curve described by the coordinate points of the cntl array and freq array pairs. From the curve, a frequency value is determined, and the oscillator will output a sine wave at that frequency. From the above, it is easy to see that array sizes of 2 for both the cntl array and the freq array will yield a linear variation of the frequency with respect to the control input. Any sizes greater than 2 will yield a piecewise linear transfer characteristic. For more detail, refer to the description of the piecewise linear controlled source, which uses a similar method to derive an output value given a control input.

```
Example SPICE Usage:
asine 1 2 in_sine
.model in_sine sine(cntl_array = [-1 0 5 6]
+                   freq_array=[10 10 1000 1000] out_low = -5.0
+                   out_high = 5.0)
```

13.2.19   Controlled Triangle Wave Oscillator

```
NAME_TABLE:
C_Function_Name:        cm_triangle
Spice_Model_Name:       triangle
Description:            "controlled triangle wave oscillator"
PORT_TABLE:
Port Name:              cntl_in                 out
Description:            "control input"         "output"
Direction:              in                      out
Default_Type:           v                       v
Allowed_Types:          [v,vd,i,id]             [v,vd,i,id]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           no                      no
PARAMETER_TABLE:
Parameter_Name:         cntl_array              freq_array
Description:            "control array"         "frequency array"
Data_Type:              real                    real
Default_Value:          0.0                     1.0e3
Limits:                 -                       [0 -]
Vector:                 yes                     yes
```

```
Vector_Bounds:      [2 -]                  cntl_array
Null_Allowed:       no                     no
PARAMETER_TABLE:
Parameter_Name:     out_low                out_high
Description:        "output peak low value" "output peak high value"
Data_Type:          real                   real
Default_Value:      -1.0                   1.0
Limits:             -                      -
Vector:             no                     no
Vector_Bounds:      -                      -
Null_Allowed:       yes                    yes
PARAMETER_TABLE:
Parameter_Name:     rise_duty
Description:        "rise time duty cycle"
Data_Type:          real
Default_Value:      0.5
Limits:             [1e-10 0.999999999]
Vector:             no
Vector_Bounds:      -
Null_Allowed:       yes
```

Description: This function is a controlled triangle/ramp wave oscillator with parameterizable values of low and high peak output and rise time duty cycle. It takes an input voltage or current value. This value is used as the independent variable in the piecewise linear curve described by the coordinate points of the cntl_array and freq_array pairs.

From the curve, a frequency value is determined, and the oscillator will output a triangle wave at that frequency. From the above, it is easy to see that array sizes of 2 for both the cntl_array and the freq_array will yield a linear variation of the frequency with respect to the control input. Any sizes greater than 2 will yield a piecewise linear transfer characteristic. For more detail, refer to the description of the piecewise linear controlled source, which uses a similar method to derive an output value given a control input.

```
Example SPICE Usage:
ain 1 2 ramp1
.model ramp1 triangle(cntl_array = [-1 0 5 6]
+                     freq_array=[10 10 1000 1000] out_low = -5.0
+                     out_high = 5.0 duty_cycle = 0.9)
```

13.2.20   Controlled Square Wave Oscillator

```
NAME_TABLE:
C_Function_Name:    cm_square
Spice_Model_Name:   square
Description:        "controlled square wave oscillator"
PORT_TABLE:
Port Name:          cntl_in                out
Description:        "control input"        "output"
Direction:          in                     out
Default_Type:       v                      v
Allowed_Types:      [v,vd,i,id]            [v,vd,i,id]
Vector:             no                     no
Vector_Bounds:      -                      -
Null_Allowed:       no                     no
PARAMETER_TABLE:
Parameter_Name:     cntl_array             freq_array
Description:        "control array"        "frequency array"
Data_Type:          real                   real
Default_Value:      0.0                    1.0e3
Limits:             -                      [0 -]
```

```
Vector:                 yes                     yes
Vector_Bounds:          [2 -]                   cntl_array
Null_Allowed:           no                      no
PARAMETER_TABLE:
Parameter_Name:         out_low                 out_high
Description:            "output peak low value" "output peak high value"
Data_Type:              real                    real
Default_Value:          -1.0                    1.0
Limits:                 -                       -
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER.TABLE:
Parameter_Name:         duty_cycle              rise_time
Description:            "duty cycle"            "output rise time"
Data_Type:              real                    real
Default_Value:          0.5                     1.0e-9
Limits:                 [1e-6 0.999999]         -
Vector: no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         fall_time
Description:            "output fall time"
Data_Type:              real
Default_Value:          1.0e-9
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
```

Description: This function is a controlled square wave oscillator with parameterizable values of low and high peak output, duty cycle, rise time, and fall time. It takes an input voltage or current value. This value is used as the independent variable in the piecewise linear curve described by the coordinate points of the cntl_array and freq_array pairs. From the curve, a frequency value is determined, and the oscillator will output a square wave at that frequency.

From the above, it is easy to see that array sizes of 2 for both the cntl_array and the freq_array will yield a linear variation of the frequency with respect to the control input. Any sizes greater than 2 will yield a piecewise linear transfer characteristic. For more detail, refer to the description of the piecewise linear controlled source, which uses a similar method to derive an output value given a control input.

```
Example SPICE Usage:
ain 1 2 pulse1
.model pulse1 square(cntl_array = [-1 0 5 6]
+               freq_array=[10 10 1000 1000] out_low = 0.0
+               out_high = 4.5 duty_cycle = 0.2
+               rise_time = 1e-6 fall_time = 2e-6)
```

13.2.21   Controlled One-Shot

```
NAME_TABLE:
C_Function_Name:        cm_oneshot
Spice_Model_Name:       oneshot
Description:            "controlled one-shot"
PORT_TABLE:
Port Name:              clk                     cntl_in
Description:            "clock input"           "control input"
Direction:              in                      in
```

```
Default_Type:         v                      v
Allowed_Types:        [v,vd,i,id]            [v,vd,i,id]
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         no                     yes
PORT_TABLE:
Port Name:            clear                  out
Description:          "clear signal"         "output"
Direction:            in                     out
Default_Type:         v                      v
Allowed_Types:        [v,vd,i,id]            [v,vd,i,id]
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    no
PARAMETER_TABLE:
Parameter_Name:       clk_trig
Description:          "clock trigger value"
Data_Type:            real
Default_Value:        0.5
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         no
PARAMETER_TABLE:
Parameter_Name:       pos_edge_trig
Description:          "positive/negative edge trigger switch"
Data_Type:            boolean
Default_Value:        TRUE
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         no
PARAMETER_TABLE:
Parameter_Name:       cntl_array             pw_array
Description:          "control array"        "pulse width array"
Data_Type:            real                   real
Default_Value:        0.0                    1.0e-6
Limits:               -                      [0.00 -]
Vector:               yes                    yes
Vector_Bounds:        -                      cntl_array
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       out_low                out_high
Description:          "output low value"     "output high value"
Data_Type:            real                   real
Default_Value:        0.0                    1.0
Limits:               -                      -
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       delay                  rise_time
Description:          "output delay from trig." "output rise time"
Data_Type:            real                   real
Default_Value:        1.0e-9                 1.0e-9
Limits:               -                      -
Vector:               no                     no
Vector_Bounds:        -                      -
```

```
Null_Allowed:         yes                          yes
PARAMETER_TABLE:
Parameter_Name:       fall_time
Description:          "output fall time"
Data_Type:            real
Default_Value:        1.0e-9
Limits:               -
Vector:               -
Vector_Bounds:        -
Null_Allowed:         yes
```

Description: This function is a controlled oneshot with parameterizable values of low and high peak output, input trigger value level, delay, and output rise and fall times. It takes an input voltage or current value. This value is used as the independent variable in the piecewise linear curve described by the coordinate points of the cntl_array and pw_array pairs. From the curve, a pulse width value is determined. The one-shot will output a pulse of that width, triggered by the clock signal (rising or falling edge), delayed by the delay value, and with specified rise and fall times. A positive slope on the clear input will immediately terminate the pulse, which resets with its fall time.

From the above, it is easy to see that array sizes of 2 for both the cntl_array and the pw_array will yield a linear variation of the pulse width with respect to the control input. Any sizes greater than 2 will yield a piecewise linear transfer characteristic. For more detail, refer to the description of the piecewise linear controlled source, which uses a similar method to derive an output value given a control input.

```
Example SPICE Usage:
ain 1 2 3 4 pulse2
.model pulse2 oneshot(cntl_array = [-1 0 10 11]
+                     pw_array=[1e-6 1e-6 1e-4 1e-4]
+                     clk_trig = 0.9 pos_edge_trig = FALSE
+                     out_low = 0.0 out_high = 4.5
+                     rise_delay = 20.0-9 fall_delay = 35.0e-9)
```

13.2.22 Capacitance Meter

```
NAME_TABLE:
C_Function_Name:      cm_cmeter
Spice_Model_Name:     cmeter
Description:          "capacitance meter"
PORT_TABLE:
Port Name:            in                           out
Description:          "input"                      "output"
Direction:            in                           out
Default_Type:         v                            v
Allowed_Types:        [v,vd,i,id]                  [v,vd,i,id]
Vector:               no                           no
Vector_Bounds:        -                            -
Null_Allowed:         no                           no
PARAMETER_TABLE:
Parameter_Name:       gain
Description:          "gain"
Data_Type:            real
Default_Value:        1.0
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
```

Description: The capacitance meter is a sensing device which is attached to a circuit node and produces as an output a scaled value equal to the total capacitance seen on its input multiplied by the gain

parameter. This model is primarily intended as a building block for other models which must sense a capacitance value and alter their behavior based upon it.

```
Example SPICE Usage:
atest1 1 2 ctest
.model ctest cmeter(gain=1.0e12)
```

13.2.23   Inductance Meter

```
NAME_TABLE:
C_Function_Name:      cm_lmeter
Spice_Model_Name:     lmeter
Description:          "inductance meter"
PORT_TABLE:
Port Name:            in                    out
Description:          "input"               "output"
Direction:            in                    out
Default_Type:         v                     v
Allowed_Types:        [v,vd,i,id]           [v,vd,i,id]
Vector:               no                    no
Vector_Bounds:        -                     -
Null_Allowed:         no                    no
PARAMETER_TABLE:
Parameter_Name:       gain
Description:          "gain"
Data_Type:            real
Default_Value:        1.0
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
```

Description: The inductance meter is a sensing device which is attached to a circuit node and produces as an output a scaled value equal to the total inductance seen on its input multiplied by the gain parameter. This model is primarily intended as a building block for other models which must sense an inductance value and alter their behavior based upon it.

```
Example SPICE Usage:
atest2 1 2 ltest
.model ltest lmeter(gain=1.0e6)
```

13.3   Hybrid Models

The following hybrid models are supplied with XSPICE. The descriptions included below consist of the model Interface Specification File and a description of the model's operation. This is followed by an example of a simulator-deck placement of the model, including the .MODEL card and the specification of all available parameters.

A note should be made with respect to the use of hybrid models for other than simple digital-to-analog and analog-to-digital translations. The hybrid models represented in this section address that specific need, but in the development of user-defined nodes you may find a need to translate not only between digital and analog nodes, but also between real and digital, real and int, etc. In most cases such translations will not need to be as involved or as detailed as shown in the following.

13.3.1   Digital-to-Analog Node Bridge

```
NAME_TABLE:
C_Function_Name:      cm_dac_bridge
Spice_Model_Name:     dac_bridge
Description:          "digital-to-analog node bridge"
```

```
PORT_TABLE:
Port Name:            in                        out
Description:          "input"                   "output"
Direction:            in                        out
Default_Type:         d                         v
Allowed_Types:        [d]                       [v,vd,i,id,d]
Vector:               yes                       yes
Vector_Bounds:        -                         -
Null_Allowed:         no                        no
PARAMETER_TABLE:
Parameter_Name:       out_low
Description:          "0-valued analog output"
Data_Type:            real
Default_Value:        0.0
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
PARAMETER_TABLE:
Parameter_Name:       out.high
Description:          "1-valued analog output"
Data_Type:            real
Default_Value:        1.0
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
PARAMETER_TABLE:
Parameter_Name:       out_undef                 input_load
Description:          "U-valued analog output"   "input load (F)"
Data_Type:            real                      real
Default_Value:        0.5                       1.0e-12
Limits:               -                         -
Vector:               no                        no
Vector_Bounds:        -                         -
Null_Allowed:         yes                       yes
PARAMETER_TABLE:
Parameter_Name:       t_rise                    t_fall
Description:          "rise time 0->1"          "fall time 1->0"
Data_Type:            real                      real
Default_Value:        1.0e-9                    1.0e-9
Limits:               -                         -
Vector:               no                        no
Vector_Bounds:        -                         -
Null_Allowed:         yes                       yes
```

Description: The dac_bridge is the first of two node bridge devices designed to allow for the ready transfer of digital information to analog values and back again. The second device is the adc_bridge (which takes an analog value and maps it to a digital one). The dac_bridge takes as input a digital value from a digital node. This value by definition may take on only one of the values "0", "1" or "U". The dac_bridge then outputs the value "out_low", "out_high" or "out_undef", or ramps linearly toward one of these "final" values from its current analog output level. The speed at which this ramping occurs depends on the values of "t_rise" and "t_fall". These parameters are interpreted by the model such that the rise or fall slope generated is always constant. **Note that the dac_bridge includes test code in its cfunc.mod file for determining the presence of the out_undef parameter. If this parameter is not specified by you, and if out_high and out_low values are specified, then out_undef is assigned the value of the arithmetic mean of out_high and out_low.** This simplifies coding of output buffers, where typically a logic family will include an out_low and out_high voltage, but not an out_undef value. This model also posts

an input load value (in farads) based on the parameter input load.

```
Example SPICE Usage:
abridge1 7 2 dac1
.model dac1 dac_bridge(out_low = 0.7 out_high = 3.5 out_undef = 2.2
+                      input_load = 5.0e-12 t_rise = 50e-9
+                      t_fall = 20e-9)
```

13.3.2   Analog-to-Digital Node Bridge

```
NAME_TABLE:
C_Function_Name:      cm_adc_bridge
Spice_Model_Name:     adc_bridge
Description:          "analog-to-digital node bridge"
PORT_TABLE:
Port Name:            in                      out
Description:          "input"                 "output"
Direction:            in                      out
Default_Type:         v                       d
Allowed_Types:        [v,vd,i,id,d]           [d]
Vector:               yes                     yes
Vector_Bounds:        -                       -
Null_Allowed:         no                      no
PARAMETER_TABLE:
Parameter_Name:       in_low
Description:          "maximum 0-valued analog input"
Data_Type:            real
Default_Value:        1.0
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
PARAMETER_TABLE:
Parameter_Name:       in_high
Description:          "minimum 1-valued analog input"
Data_Type:            real
Default_Value:        2.0
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
PARAMETER_TABLE:
Parameter_Name:       rise_delay              fall_delay
Description:          "rise delay"            "fall delay"
Data_Type:            real                    real
Default_Value:        1.0e-9                  1.0e-9
Limits:               [1.0e-12 -]             [1.0e-12 -]
Vector:               no                      no
Vector_Bounds:        -                       -
Null_Allowed:         yes                     yes
```

Description: The adc_bridge is one of two node bridge devices designed to allow for the ready transfer of analog information to digital values and back again. The second device is the dac_bridge (which takes a digital value and maps it to an analog one). The adc_bridge takes as input an analog value from an analog node. This value by definition may be in the form of a voltage, or a current. If the input value is less than or equal to in_low, then a digital output value of "0" is generated. If the input is greater than or equal to in_high, a digital output value of "1" is generated. If neither of these is true, then a digital "UNKNOWN" value is output. Note that unlike the case of the dac_bridge, no ramping time or delay is associated with the adc_bridge. Rather, the continuous ramping of the input value provides for any associated delays in the digitized signal.

```
Example SPICE Usage:
abridge2 1 8 adc_buff
.model adc_buff adc_bridge(in_low = 0.3 in_high = 3.5)
```

13.3.3 Controlled Digital Oscillator

```
NAME_TABLE:
C_Function_Name:     cm_d_osc
Spice_Model_Name:    d_osc
Description:         "controlled digital oscillator"
PORT_TABLE:
Port Name:           cntl_in                out
Description:         "control input"        "output"
Direction:           in                     out
Default_Type:        v                      d
Allowed_Types:       [v,vd,i,id]            [d]
Vector:              no                     no
Vector_Bounds:       -                      -
Null_Allowed:        no                     no
PARAMETER_TABLE:
Parameter_Name:      cntl_array             freq_array
Description:         "control array"        "frequency array"
Data_Type:           real                   real
Default_Value:       0.0                    1.0e6
Limits:              -                      [0 -]
Vector:              yes                    yes
Vector_Bounds:       [2 -]                  cntl_array
Null_Allowed:        no                     no
PARAMETER_TABLE:
Parameter_Name:      duty_cycle             init_phase
Description:         "duty cycle"           "initial phase of output"
Data_Type:           real                   real
Default_Value:       0.5                    0
Limits:              [1e-6 0.999999]        [-180.0 +360.0]
Vector:              no                     no
Vector_Bounds:       -                      -
Null_Allowed:        yes                    yes
PARAMETER_TABLE:
Parameter_Name:      rise_delay             fall_delay
Description:         "rise delay"           "fall delay"
Data_Type:           real                   real
Default_Value:       1e-9                   1e-9
Limits:              [0 -]                  [0 -]
Vector:              no                     no
Vector_Bounds:       -                      -
Null_Allowed:        yes                    yes
```

Description: The digital oscillator is a hybrid model which accepts as input a voltage or current. This input is compared to the voltage-to-frequency transfer characteristic specified by the cntl_array/freq_array coordinate pairs, and a frequency is obtained which represents a linear interpolation or extrapolation based on those pairs. A digital time-varying signal is then produced with this fundamental frequency.

The output waveform, which is the equivalent of a digital clock signal, has rise and fall delays which can be specified independently. In addition, the duty cycle and the phase of the waveform are also variable and can be set by you.

```
Example SPICE Usage:
a5 1 8 var_clock
.model var_clock d_osc(cntl_array = [-2 -1 1 2]
```

```
+                              freq_array = [1e3 1e3 10e3 10e3]
+                              duty_cycle = 0.4 init_phase = 180.0
+                              rise_delay = 10e-9 fall_delay=8e-9)
```

13.4   Digital Models

The following digital models are supplied with XSPICE. The descriptions included below consist of an example model Interface Specification File and a description of the model's operation. This is followed by an example of a simulator-deck placement of the model, including the .MODEL card and the specification of all available parameters. Note that these models have not been finalized at this time.

Some information common to all digital models and/or digital nodes is included here. The following are general rules which should make working with digital nodes and models more straightforward:

1. All digital nodes are initialized to ZERO at the start of a simulation (i.e., when INIT=TRUE). This means that a model need not post an explicit value to an output node upon initialization if its output would normally be a ZERO (although posting such would certainly cause no harm).

13.4.1   Buffer

```
NAME_TABLE:
C_Function_Name:        cm_d_buffer
Spice_Model_Name:       d_buffer
Description:            "digital one-bit-wide buffer"
PORT_TABLE:
Port Name:              in                         out
Description:            "input"                    "output"
Direction:              in                         out
Default_Type:           d                          d
Allowed_Types:          [d]                        [d]
Vector:                 no                         no
Vector_Bounds:          -                          -
Null_Allowed:           no                         no
PARAMETER_TABLE:
Parameter_Name:         rise_delay                 fall_delay
Description:            "rise delay"               "fall delay"
Data_Type:              real                       real
Default_Value:          1.0e-9                     1.0e-9
Limits:                 [1.0e-12 -]                [1.0e-12 -]
Vector:                 no                         no
Vector_Bounds:          -                          -
Null_Allowed:           yes                        yes
PARAMETER_TABLE:
Parameter_Name:         input_load
Description:            "input load value (F)"
Data_Type:              real
Default_Value:          1.0e-12
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
```

Description: The buffer is a single-input, single-output digital buffer which produces as output a time-delayed copy of its input. The delays associated with an output rise and those associated with an output fall may be different. The model also posts an input load value (in farads) based on the parameter input load. The output of this model does NOT, however, respond to the total loading it sees on its output; it will always drive the output strongly with the specified delays.

```
Example SPICE Usage:
a6 1 8 buff1
```

```
.model buff1 d_buffer(rise_delay = 0.5e-9 fall_delay = 0.3e-9
+                     input_load = 0.5e-12)
```

13.4.2 Inverter

```
NAME_TABLE:
C_Function_Name:     cm_d_inverter
Spice_Model_Name:    d_inverter
Description:         "digital one-bit-wide inverter"
PORT_TABLE:
Port Name:           in                    out
Description:         "input"               "output"
Direction:           in                    out
Default_Type:        d                     d
Allowed_Types:       [d]                   [d]
Vector:              no                    no
Vector_Bounds:       -                     -
Null_Allowed:        no                    no
PARAMETER_TABLE:
Parameter_Name:      rise_delay            fall_delay
Description:         "rise delay"          "fall delay"
Data_Type:           real                  real
Default_Value:       1.0e-9                1.0e-9
Limits:              [1.0e-12 -]           [1.0e-12 -]
Vector:              no                    no
Vector_Bounds:       -                     -
Null_Allowed:        yes                   yes
PARAMETER_TABLE:
Parameter_Name:      input_load
Description:         "input load value (F)"
Data_Type:           real
Default_Value:       1.0e-12
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
```

Description: The inverter is a single-input, single-output digital inverter which produces as output an inverted, time- delayed copy of its input. The delays associated with an output rise and those associated with an output fall may be specified independently. The model also posts an input load value (in farads) based on the parameter input load. The output of this model does NOT, however, respond to the total loading it sees on its output; it will always drive the output strongly with the specified delays.

```
Example SPICE Usage:
a6 1 8 inv1
.model inv1 d_inverter(rise_delay = 0.5e-9 fall_delay = 0.3e-9
+                      input_load = 0.5e-12)
```

13.4.3 And

```
NAME_TABLE:
C_Function_Name:     cm_d_and
Spice_Model_Name:    d_and
Description:         "digital 'and' gate"
PORT_TABLE:
Port Name:           in                    out
Description:         "input"               "output"
Direction:           in                    out
```

```
Default_Type:        d                      d
Allowed_Types:       [d]                    [d]
Vector:              yes                    no
Vector_Bounds:       [2 -]                  -
Null_Allowed:        no                     no
PARAMETER_TABLE:
Parameter_Name:      rise_delay             fall_delay
Description:         "rise delay"           "fall delay"
Data_Type:           real                   real
Default_Value:       1.0e-9                 1.0e-9
Limits:              [1.0e-12 -]            [1.0e-12 -]
Vector:              no                     no
Vector_Bounds:       -                      -
Null_Allowed:        yes                    yes
PARAMETER_TABLE:
Parameter_Name:      input_load
Description:         "input load value (F)"
Data_Type:           real
Default_Value:       1.0e-12
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
```

Description: The digital 'and' gate is an n-input, single-output 'and' gate which produces an active "1" value if, and only if, all of its inputs are also "1" values. If ANY of the inputs is a "0", the output will also be a "0"; if neither of these conditions holds, the output will be unknown. The delays associated with an output rise and those associated with an output fall may be specified independently. The model also posts an input load value (in farads) based on the parameter input load. The output of this model does NOT, however, respond to the total loading it sees on its output; it will always drive the output strongly with the specified delays.

```
Example SPICE Usage:
a6 [1 2] 8 and1
.model and1 d_and(rise_delay = 0.5e-9 fall_delay = 0.3e-9
+               input_load = 0.5e-12)
```

13.4.4   Nand

```
NAME_TABLE:
C_Function_Name:     cm_d_nand
Spice_Model_Name:    d_nand
Description:         "digital 'nand' gate"
PORT_TABLE:
Port Name:           in                     out
Description:         "input"                "output"
Direction:           in                     out
Default_Type:        d                      d
Allowed_Types:       [d]                    [d]
Vector:              yes                    no
Vector_Bounds:       [2 -]                  -
Null_Allowed:        no                     no
PARAMETER_TABLE:
Parameter_Name:      rise_delay             fall_delay
Description:         "rise delay"           "fall delay"
Data_Type:           real                   real
Default_Value:       1.0e-9                 1.0e-9
Limits:              [1.0e-12 -]            [1.0e-12 -]
Vector:              no                     no
```

```
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         input_load
Description:            "input load value (F)"
Data_Type:              real
Default_Value:          1.0e-12
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
```

Description: The digital 'nand' gate is an n-input, single-output 'nand' gate which produces an active "0" value if and only if all of its inputs are "1" values. If ANY of the inputs is a "0", the output will be a "1"; if neither of these conditions holds, the output will be unknown. The delays associated with an output rise and those associated with an output fall may be specified independently. The model also posts an input load value (in farads) based on the parameter input load. The output of this model does NOT, however, respond to the total loading it sees on its output; it will always drive the output strongly with the specified delays.

```
Example SPICE Usage:
a6 [1 2 3] 8 nand1
.model nand1 d_nand(rise_delay = 0.5e-9 fall_delay = 0.3e-9
+                  input_load = 0.5e-12)
```

13.4.5   Or

```
NAME_TABLE:
C_Function_Name:        cm_d_or
Spice_Model_Name:       d_or
Description:            "digital 'or' gate"
PORT_TABLE:
Port Name:              in                      out
Description:            "input"                 "output"
Direction:              in                      out
Default_Type:           d                       d
Allowed_Types:          [d]                     [d]
Vector:                 yes                     no
Vector_Bounds:          [2 -]                   -
Null_Allowed:           no                      no
PARAMETER_TABLE:
Parameter_Name:         rise_delay              fall_delay
Description:            "rise delay"            "fall delay"
Data_Type:              real                    real
Default_Value:          1.0e-9                  1.0e-9
Limits:                 [1.0e-12 -]             [1.0e-12 -]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         input_load
Description:            "input load value (F)"
Data_Type:              real
Default_Value:          1.0e-12
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
```

Description: The digital 'or' gate is an n-input, single-output 'or' gate which produces an active "1" value if at least one of its inputs is a "1" value. The gate produces a "0" value if all inputs are "0"; if neither of these two conditions holds, the output is unknown. The delays associated with an output rise and those associated with an output fall may be specified independently. The model also posts an input load value (in farads) based on the parameter input load. The output of this model does NOT, however, respond to the total loading it sees on its output; it will always drive the output strongly with the specified delays.

```
Example SPICE Usage:
a6 [1 2 3] 8 or1
.model or1 d_or(rise_delay = 0.5e-9 fall_delay = 0.3e-9
+               input_load = 0.5e-12)
```

13.4.6   Nor

```
NAME_TABLE:
C_Function_Name:    cm_d_nor
Spice_Model_Name:   d_nor
Description:        "digital 'nor' gate"
PORT_TABLE:
Port Name:          in                      out
Description:        "input"                 "output"
Direction:          in                      out
Default_Type:       d                       d
Allowed_Types:      [d]                     [d]
Vector:             yes                     no
Vector_Bounds:      [2 -]                   -
Null_Allowed:       no                      no
PARAMETER_TABLE:
Parameter_Name:     rise_delay              fall_delay
Description:        "rise delay"            "fall delay"
Data_Type:          real                    real
Default_Value:      1.0e-9                  1.0e-9
Limits:             [1.0e-12 -]             [1.0e-12 -]
Vector:             no                      no
Vector_Bounds:      -                       -
Null_Allowed:       yes                     yes
PARAMETER_TABLE:
Parameter_Name:     input_load
Description:        "input load value (F)"
Data_Type:          real
Default_Value:      1.0e-12
Limits:             -
Vector:             no
Vector_Bounds:      -
Null_Allowed:       yes
```

Description: The digital 'nor' gate is an n-input, single-output 'nor' gate which produces an active "0" value if at least one of its inputs is a "1" value. The gate produces a "0" value if all inputs are "0"; if neither of these two conditions holds, the output is unknown. The delays associated with an output rise and those associated with an output fall may be specified independently. The model also posts an input load value (in farads) based on the parameter input load. The output of this model does NOT, however, respond to the total loading it sees on its output; it will always drive the output strongly with the specified delays.

```
Example SPICE Usage:
anor12 [1 2 3 4] 8 nor12
.model nor12 d_or(rise_delay = 0.5e-9 fall_delay = 0.3e-9
+               input_load = 0.5e-12)
```

13.4.7 Xor

```
NAME_TABLE:
C_Function_Name:     cm_d_xor
Spice_Model_Name:    d_xor
Description:         "digital exclusive-or gate"
PORT_TABLE:
Port Name:           in                    out
Description:         "input"               "output"
Direction:           in                    out
Default_Type:        d                     d
Allowed_Types:       [d]                   [d]
Vector:              yes                   no
Vector_Bounds:       [2 -]                 -
Null_Allowed:        no                    no
PARAMETER_TABLE:
Parameter_Name:      rise_delay            fall_delay
Description:         "rise delay"          "fall delay"
Data_Type:           real                  real
Default_Value:       1.0e-9                1.0e-9
Limits:              [1.0e-12 -]           [1.0e-12 -]
Vector:              no                    no
Vector_Bounds:       -                     -
Null_Allowed:        yes                   yes
PARAMETER_TABLE:
Parameter_Name:      input_load
Description:         "input load value (F)"
Data_Type:           real
Default_Value:       1.0e-12
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
```

Description: The digital 'xor' gate is an n-input, single-output 'xor' gate which produces an active "1" value if an odd number of its inputs are also "1" values. The delays associated with an output rise and those associated with an output fall may be specified independently.

The model also posts an input load value (in farads) based on the parameter input load. The output of this model does NOT, however, respond to the total loading it sees on its output; it will always drive the output strongly with the specified delays. Note also that to maintain the technology-independence of the model, any UNKNOWN input, or any floating input causes the output to also go UNKNOWN.

```
Example SPICE Usage:
a9 [1 2] 8 xor3
.model xor3 d_xor(rise_delay = 0.5e-9 fall_delay = 0.3e-9
+                input_load = 0.5e-12)
```

13.4.8 Xnor

```
NAME_TABLE:
C_Function_Name:     cm_d_xnor
Spice_Model_Name:    d_xnor
Description:         "digital exclusive-nor gate"
PORT_TABLE:
Port Name:           in                    out
Description:         "input"               "output"
Direction:           in                    out
Default_Type:        d                     d
Allowed_Types:       [d]                   [d]
```

```
Vector:                 yes                     no
Vector_Bounds:          [2 -]                   -
Null_Allowed:           no                      no
PARAMETER_TABLE:
Parameter_Name:         rise_delay              fall_delay
Description:            "rise delay"            "fall delay"
Data_Type:              real                    real
Default_Value:          1.0e-9                  1.0e-9
Limits:                 [1.0e-12 -]             [1.0e-12 -]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         input_load
Description:            "input load value (F)"
Data_Type:              real
Default_Value:          1.0e-12
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
```

Description: The digital 'xnor' gate is an n-input, single-output 'xnor' gate which produces an active
"0" value if an odd number of its inputs are also "1" values. It produces a "1" output when an even
number of "1" values occurs on its inputs. The delays associated with an output rise and those
associated with an output fall may be specified independently. The model also posts an input load
value (in farads) based on the parameter input load. The output of this model does NOT, however,
respond to the total loading it sees on its output; it will always drive the output strongly with
the specified delays. Note also that to maintain the technology-independence of the model, any
UNKNOWN input, or any floating input causes the output to also go UNKNOWN.

```
Example SPICE Usage:
a9 [1 2] 8 xnor3
.model xnor3 d_xnor(rise_delay = 0.5e-9 fall_delay = 0.3e-9
+                   input_load = 0.5e-12)
```

13.4.9   Tristate

```
NAME_TABLE:
C_Function_Name:        cm_d_tristate
Spice_Model_Name:       d_tristate
Description:            "digital tristate buffer"
PORT_TABLE:
Port Name:              in        enable        out
Description:            "input"   "enable"      "output"
Direction:              in        in            out
Default_Type:           d         d             d
Allowed_Types:          [d]       [d]           [d]
Vector:                 no        no            no
Vector_Bounds:          -         -             -
Null_Allowed:           no        no            no
PARAMETER_TABLE:
Parameter_Name:         delay
Description:            "delay"
Data_Type:              real
Default_Value:          1.0e-9
Limits:                 [1.0e-12 -]
Vector:                 no
Vector_Bounds:          -
```

```
Null_Allowed:          yes
PARAMETER_TABLE:
Parameter_Name:        input_load
Description:           "input load value (F)"
Data_Type:             real
Default_Value:         1.0e-12
Limits:                -
Vector:                no
Vector_Bounds:         -
Null_Allowed:          yes
PARAMETER_TABLE:
Parameter_Name:        enable_load
Description:           "enable load value (F)"
Data_Type:             real
Default_Value:         1.0e-12
Limits:                -
Vector:                no
Vector_Bounds:         -
Null_Allowed:          yes
```

Description: The digital tristate is a simple tristate gate which can be configured to allow for open-collector behavior, as well as standard tristate behavior. The state seen on the input line is reflected in the output. The state seen on the enable line determines the strength of the output. Thus, a ONE forces the output to its state with a STRONG strength. A ZERO forces the output to go to a HI_IMPEDANCE strength. The delays associated with an output state or strength change cannot be specified independently, nor may they be specified independently for rise or fall conditions; other gate models may be used to provide such delays if needed. The model posts input and enable load values (in farads) based on the parameters input load and enable.The output of this model does NOT, however, respond to the total loading it sees on its output; it will always drive the output with the specified delay. Note also that to maintain the technology-independence of the model, any UNKNOWN input, or any floating input causes the output to also go UNKNOWN. Likewise, any UNKNOWN input on the enable line causes the output to go to an UNDETERMINED strength value.

```
Example SPICE Usage:
a9 1 2 8 tri7
.model tri7 d_tristate(delay = 0.5e-9 input_load = 0.5e-12
+                      enable_load = 0.5e-12)
```

13.4.10   Pullup

```
NAME_TABLE:
C_Function_Name:       cm_d_pullup
Spice_Model_Name:      d_pullup
Description:            "digital pullup resistor"
PORT_TABLE:
Port Name:             out
Description:           "output"
Direction:             out
Default_Type:          d
Allowed_Types:         [d]
Vector:                no
Vector_Bounds:         -
Null_Allowed:          no
PARAMETER_TABLE:
Parameter_Name:        load
Description:           "load value (F)"
Data_Type:             real
Default_Value:         1.0e-12
```

```
Limits:                  -
Vector:                  no
Vector_Bounds:           -
Null_Allowed:            yes
```

Description: The digital pullup resistor is a device which emulates the behavior of an analog resistance value tied to a high voltage level. The pullup may be used in conjunction with tristate buffers to provide open-collector wired "or" constructs, or any other logical constructs which rely on a resistive pullup common to many tristated output devices. The model posts an input load value (in farads) based on the parameters "load".

```
Example SPICE Usage:
a2 9 pullup1
.model pullup1 d_pullup(load = 20.0e-12)
```

13.4.11   Pulldown

```
NAME_TABLE:
C_Function_Name:      cm_d_pulldown
Spice_Model_Name:     d_pulldown
Description:          "digital pulldown resistor"
PORT_TABLE:
Port Name:            out
Description:          "output"
Direction:            out
Default_Type:         d
Allowed_Types:        [d]
Vector:               no
Vector_Bounds:        -
Null_Allowed:         no
PARAMETER_TABLE:
Parameter_Name:       load
Description:          "load value (F)"
Data_Type:            real
Default_Value:        1.0e-12
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
```

Description: The digital pulldown resistor is a device which emulates the behavior of an analog resistance value tied to a low voltage level. The pulldown may be used in conjunction with tristate buffers to provide open-collector wired "or" constructs, or any other logical constructs which rely on a resistive pulldown common to many tristated output devices. The model posts an input load value (in farads) based on the parameters "load".

```
Example SPICE Usage:
a4 9 pulldown1
.model pulldown1 d_pulldown(load = 20.0e-12)
```

13.4.12   D Flip Flop

```
NAME_TABLE:
C_Function_Name:      cm_d_dff
Spice_Model_Name:     d_dff
Description:          "digital d-type flip flop"
PORT_TABLE:
Port Name:            data                clk
Description:          "input data"        "clock"
Direction:            in                  in
```

```
Default_Type:           d                           d
Allowed_Types:          [d]                         [d]
Vector:                 no                          no
Vector_Bounds:          -                           -
Null_Allowed:           no                          no
PORT_TABLE:
Port Name:              set                         reset
Description:            "asynch. set"               "asynch. reset"
Direction:              in                          in
Default_Type:           d                           d
Allowed_Types:          [d]                         [d]
Vector:                 no                          no
Vector_Bounds:          -                           -
Null_Allowed:           yes                         yes
PORT_TABLE:
Port Name:              out                         Nout
Description:            "data output"               "inverted data output"
Direction:              out                         out
Default_Type:           d                           d
Allowed_Types:          [d]                         [d]
Vector:                 no                          no
Vector_Bounds:          -                           -
Null_Allowed:           yes                         yes
PARAMETER_TABLE:
Parameter_Name:         clk_delay                   set_delay
Description:            "delay from clk"            "delay from set"
Data_Type:              real                        real
Default_Value:          1.0e-9                      1.0e-9
Limits:                 [1.0e-12 -]                 [1.0e-12 -]
Vector:                 no                          no
Vector_Bounds:          -                           -
Null_Allowed:           yes                         yes
PARAMETER_TABLE:
Parameter_Name:         reset_delay                 ic
Description:            "delay from reset"          "output initial state"
Data_Type:              real                        int
Default_Value:          1.0                         0
Limits:                 [1.0e-12 -]                 [0 2]
Vector:                 no                          no
Vector_Bounds:          -                           -
Null_Allowed:           yes                         yes
PARAMETER_TABLE:
Parameter_Name:         data_load                   clk_load
Description:            "data load value (F)"       "clk load value (F)"
Data_Type:              real                        real
Default_Value:          1.0e-12                     1.0e-12
Limits:                 -                           -
Vector:                 no                          no
Vector_Bounds:          -                           -
Null_Allowed:           yes                         yes
PARAMETER_TABLE:
Parameter_Name:         set_load                    reset_load
Description:            "set load value (F)"        "reset load (F)"
Data_Type:              real                        real
Default_Value:          1.0e-12                     1.0e-12
Limits:                 -                           -
Vector:                 no                          no
Vector.Bounds:          -                           -
```

```
Null_Allowed:        yes                     yes
PARAMETER_TABLE:
Parameter_Name:      rise_delay              fall_delay
Description:         "rise delay"            "fall delay"
Data_Type:           real                    real
Default_Value:       1.0e-9                  1.0e-9
Limits:              [1.0e-12 -]             [1.0e-12 -]
Vector:              no                      no
Vector_Bounds:       -                       -
Null_Allowed:        yes                     yes
```

Description: The digital d-type flip flop is a one-bit, edge-triggered storage element which will store data whenever the clk input line transitions from low to high (ZERO to ONE). In addition, asynchronous set and reset signals exist, and each of the three methods of changing the stored output of the d_dff have separate load values and delays associated with them. Additionally, you may specify separate rise and fall delay values that are added to those specified for the input lines; these allow for more faithful reproduction of the output characteristics of different IC fabrication technologies.

Note that any UNKNOWN input on the set or reset lines immediately results in an UNKNOWN output.

```
Example SPICE Usage:
a7 1 2 3 4 5 6 flop1
.model flop1 d_dff(clk_delay = 13.0e-9 set_delay = 25.0e-9
+               reset_delay = 27.0e-9 ic = 2 rise_delay = 10.0e-9
+               fall_delay = 3e-9)
```

13.4.13   JK Flip Flop

```
NAME_TABLE:
C_Function_Name:     cm_d_jkff
Spice_Model_Name:    d_jkff
Description:         "digital jk-type flip flop"
PORT_TABLE:
Port Name:           j                       k
Description:         "j input"               "k input"
Direction:           in                      in
Default_Type:        d                       d
Allowed_Types:       [d]                     [d]
Vector:              yes                     yes
Vector_Bounds:       [2 -]                   j
Null_Allowed:        no                      no
PORT_TABLE:
Port Name:           clk
Description:         "clock"
Direction:           in
Default_Type:        d
Allowed_Types:       [d]
Vector:              no
Vector_Bounds:       -
Null_Allowed:        no
PORT_TABLE:
Port Name:           set                     reset
Description:         "asynchronous set"      "asynchronous reset"
Direction:           in                      in
Default_Type:        d                       d
Allowed_Types:       [d]                     [d]
Vector:              no                      no
Vector_Bounds:       -                       -
Null_Allowed:        yes                     yes
```

```
PORT_TABLE:
Port Name:          out                 Nout
Description:        "data output"       "inverted data output"
Direction:          out                 out
Default_Type:       d                   d
Allowed_Types:      [d]                 [d]
Vector:             no                  no
Vector_Bounds       -                   -
Null_Allowed:       yes                 yes
PARAMETER_TABLE:
Parameter_Name:     clk_delay           set_delay
Description:        "delay from clk"    "delay from set"
Data_Type:          real                real
Default_Value:      1.0e-9              1.0e-9
Limits:             [1.0e-12 -]         [1.0e-12 -]
Vector:             no                  no
Vector_Bounds:      -                   -
Null_Allowed:       yes                 yes
PARAMETER_TABLE:
Parameter_Name:     reset_delay         ic
Description:        "delay from reset"  "output initial state"
Data_Type:          real                int
Default_Value:      1.0                 0
Limits:             [1.0e-12 -]         [0 2]
Vector:             no                  no
Vector_Bounds:      -                   -
Null_Allowed:       yes                 yes
PARAMETER_TABLE:
Parameter_Name:     jk_load             clk_load
Description:        "j,k load values (F)" "clk load value (F)"
Data_Type:          real                real
Default_Value:      1.0e-12             1.0e-12
Limits:             -                   -
Vector:             no                  no
Vector_Bounds:      -                   -
Null_Allowed:       yes                 yes
PARAMETER_TABLE:
Parameter_Name:     set_load            reset_load
Description:        "set load value (F)" "reset load (F)"
Data_Type:          real                real
Default_Value:      1.0e-12             1.0e-12
Limits:             -                   -
Vector:             no                  no
Vector_Bounds:      -                   -
Null_Allowed:       yes                 yes
PARAMETER_TABLE:
Parameter_Name:     rise_delay          fall_delay
Description:        "rise delay"        "fall delay"
Data_Type:          real                real
Default_Value:      1.0e-9              1.0e-9
Limits:             [1.0e-12 -]         [1.0e-12 -]
Vector:             no                  no
Vector_Bounds:      -                   -
Null_Allowed:       yes                 yes
```

Description: The digital jk-type flip flop is a one-bit, edge-triggered storage element which will store data whenever the clk input line transitions from low to high (ZERO to ONE). In addition, asynchronous set and reset signals exist, and each of the three methods of changing the stored output of the d_jkff have separate load values and delays associated with them. Additionally, you may

specify separate rise and fall delay values that are added to those specified for the input lines;
these allow for more faithful reproduction of the output characteristics of different IC fabrication
technologies.
Note that any UNKNOWN inputs other than j or k cause the output to go UNKNOWN automat-
ically.

```
Example SPICE Usage:
a8 1 2 3 4 5 6 7 flop2
.model flop2 d_jkff(clk_delay = 13.0e-9 set_delay = 25.0e-9
+                   reset_delay = 27.0e-9 ic = 2 rise_delay = 10.0e-9
+                   fall_delay = 3e-9)
```

13.4.14   Toggle Flip Flop

```
NAME_TABLE:
C_Function_Name:      cm_d_tff
Spice_Model_Name:     d_tff
Description:          "digital toggle flip flop"
PORT_TABLE:
Port Name:            t                        clk
Description:          "toggle input"           "clock"
Direction:            in                       in
Default_Type:         d                        d
Allowed_Types:        [d]                      [d]
Vector:               yes                      no
Vector_Bounds:        [2 -]                    -
Null_Allowed:         no                       no
PORT_TABLE:
Port Name:            set                      reset
Description:          "set"                    "reset"
Direction:            in                       in
Default_Type:         d                        d
Allowed_Types:        [d]                      [d]
Vector:               no                       no
Vector_Bounds:        -                        -
Null_Allowed:         yes                      yes
PORT.TABLE:
Port Name:            out                      Nout
Description:          "data output"            "inverted data output"
Direction:            out                      out
Default_Type:         d                        d
Allowed_Types:        [d]                      [d]
Vector:               no                       no
Vector_Bounds:        -                        -
Null_Allowed:         yes                      yes
PARAMETER_TABLE:
Parameter_Name:       clk_delay                set_delay
Description:          "delay from clk"         "delay from set"
Data_Type:            real                     real
Default_Value:        1.0e-9                   1.0e-9
Limits:               [1.0e-12 -]              [1.0e-12 -]
Vector:               no                       no
Vector_Bounds         -                        -
Null_Allowed:         yes                      yes
PARAMETER_TABLE:
Parameter_Name:       reset_delay              ic
Description:          "delay from reset"       "output initial state"
Data_Type:            real                     int
Default_Value:        1.0                      0
```

```
Limits:                 [1.0e-12 -]             [0 2]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         t_load                  clk_load
Description:            "toggle load value (F)" "clk load value (F)"
Data_Type:              real                    real
Default_Value:          1.0e-12                 1.0e-12
Limits:                 -                       -
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         set_load                reset_load
Description:            "set load value (F)"    "reset load (F)"
Data_Type:              real                    real
Default.Value:          1.0e-12                 1.0e-12
Limits:                 -                       -
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         rise_delay              fall_delay
Description:            "rise delay"            "fall delay"
Data_Type:              real                    real
Default_Value:          1.0e-9                  1.0e-9
Limits:                 [1.0e-12 -]             [1.0e-12 -]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
```

Description: The digital toggle-type flip flop is a one-bit, edge-triggered storage element which will toggle its current state whenever the clk input line transitions from low to high (ZERO to ONE). In addition, asynchronous set and reset signals exist, and each of the three methods of changing the stored output of the d_tff have separate load values and delays associated with them. Additionally, you may specify separate rise and fall delay values that are added to those specified for the input lines; these allow for more faithful reproduction of the output characteristics of different IC fabrication technologies.

Note that any UNKNOWN inputs other than t immediately cause the output to go UNKNOWN.

```
Example SPICE Usage:
a8 2 12 4 5 6 3 flop3
.model flop3 d_tff(clk_delay = 13.0e-9 set_delay = 25.0e-9
+               reset_delay = 27.0e-9 ic = 2 rise_delay = 10.0e-9
+               fall_delay = 3e-9 t_load = 0.2e-12)
```

13.4.15   Set-Reset Flip Flop

```
NAME_TABLE:
C_Function_Name:        cm_d_srff
Spice_Model_Name:       d_srff
Description:            "digital set-reset flip flop"
PORT_TABLE:
Port Name:              s                       r
Description:            "set input"             "reset input"
Direction:              in                      in
Default_Type:           d                       d
Allowed_Types:          [d]                     [d]
```

```
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           no                      no
PORT_TABLE:
Port Name:              clk
Description:            "clock"
Direction:              in
Default_Type:           d
Allowed_Types:          [d]
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           no
PORT_TABLE:
Port Name:              set                     reset
Description:            "asynchronous set"      "asynchronous reset"
Direction:              in                      in
Default_Type:           d                       d
Allowed_Types:          [d]                     [d]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PORT_TABLE:
Port Name:              out                     Nout
Description:            "data output"           "inverted data output"
Direction:              out                     out
Default_Type:           d                       d
Allowed_Types:          [d]                     [d]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         clk_delay               set_delay
Description:            "delay from clk"        "delay from set"
Data_Type:              real                    real
Default_Value:          1.0e-9                  1.0e-9
Limits:                 [1.0e-12 -]             [1.0e-12 -]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         reset_delay             ic
Description:            "delay from reset"      "output initial state"
Data_Type:              real                    int
Default_Value:          1.0e-9                  0
Limits:                 [1.0e-12 -]             [0 2]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
Parameter_Name:         sr_load                 clk_load
Description:            "set/reset loads (F)"   "clk load value (F)"
Data_Type:              real                    real
Default_Value:          1.0e-12                 1.0e-12
Limits:                 -                       -
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:
```

```
Parameter_Name:      set_load                reset_load
Description:         "set load value (F)"    "reset load (F)"
Data_Type:           real                    real
Default_Value:       1.0e-12                 1.0e-12
Limits:              -                       -
Vector:              no                      no
Vector_Bounds:       -                       -
Null_Allowed:        yes                     yes
PARAMETER_TABLE:
Parameter_Name:      rise_delay              fall_delay
Description:         "rise delay"            "fall delay"
Data_Type:           real                    real
Default_Value:       1.0e-9                  1.0e-9
Limits:              [1.0e-12 -]             [1.0e-12 -]
Vector:              no                      no
Vector_Bounds:       -                       -
Null_Allowed:        yes                     yes
```

Description: The digital sr-type flip flop is a one-bit, edge-triggered storage element which will store data whenever the clk input line transitions from low to high (ZERO to ONE). The value stored (i.e., the "out" value) will depend on the s and r input pin values, and will be:

```
out=ONE            if s=ONE and r=ZERO;
out=ZERO           if s=ZERO and r=ONE;
out=previous value if s=ZERO and r=ZERO;
out=UNKNOWN        if s=ONE and r=ONE;
```

In addition, asynchronous set and reset signals exist, and each of the three methods of changing the stored output of the d_srff have separate load values and delays associated with them. You may also specify separate rise and fall delay values that are added to those specified for the input lines; these allow for more faithful reproduction of the output characteristics of different IC fabrication technologies.

Note that any UNKNOWN inputs other than s and r immediately cause the output to go UNKNOWN. Example SPICE Usage:

```
a8 2 12 4 5 6 3 14 flop7
.model flop7 d_srff(clk_delay = 13.0e-9 set_delay = 25.0e-9
+                   reset_delay = 27.0e-9 ic = 2 rise_delay = 10.0e-9
+                   fall_delay = 3e-9)
```

13.4.16   D Latch

```
NAME_TABLE:
C_Function_Name:     cm_d_dlatch
Spice_Model_Name:    d_dlatch
Description:         "digital d-type latch"
PORT_TABLE:
Port Name:           data                    enable
Description:         "input data"            "enable input"
Direction:           in                      in
Default_Type:        d                       d
Allowed_Types:       [d]                     [d]
Vector:              no                      no
Vector_Bounds:       -                       -
Null_Allowed:        no                      no
PORT_TABLE:
Port Name:           set                     reset
Description:         "set"                   "reset"
Direction:           in                      in
Default_Type:        d                       d
Allowed_Types:       [d]                     [d]
```

```
Vector:                no                 no
Vector_Bounds:         -                  -
Null_Allowed:          yes                yes
PORT_TABLE:
Port Name:             out                Nout
Description:           "data output"      "inverter data output"
Direction:             out                out
Default_Type:          d                  d
Allowed_Types:         [d]                [d]
Vector:                no                 no
Vector_Bounds:         -                  -
Null_Allowed:          no                 no
PARAMETER_TABLE:
Parameter_Name:        data_delay
Description:           "delay from data"
Data_Type:             real
Default_Value:         1.0e-9
Limits:                [1.0e-12 -]
Vector:                no
Vector_Bounds:         -
Null_Allowed:          yes
PARAMETER_TABLE:
Parameter_Name:        enable_delay       set_delay
Description:           "delay from enable" "delay from SET"
Data_Type:             real               real
Default_Value:         1.0e-9             1.0e-9
Limits:                [1.0e-12 -]        [1.0e-12 -]
Vector:                no                 no
Vector_Bounds:         -                  -
Null_Allowed:          yes                yes
PARAMETER_TABLE:
Parameter_Name:        reset_delay        ic
Description:           "delay from RESET"  "output initial state"
Data_Type:             real               boolean
Default_Value:         1.0e-9             0
Limits:                [1.0e-12 -]        -
Vector:                no                 no
Vector_Bounds:         -                  -
Null_Allowed:          yes                yes
PARAMETER_TABLE:
Parameter_Name:        data_load          enable_load
Description:           "data load (F)"    "enable load value (F)"
Data_Type:             real               real
Default_Value:         1.0e-12            1.0e-12
Limits:                -                  -
Vector:                no                 no
Vector_Bounds:         -                  -
Null_Allowed:          yes                yes
PARAMETER_TABLE:
Parameter_Name:        set_load           reset_load
Description:           "set load value (F)" "reset load (F)"
Data_Type:             real               real
Default_Value:         1.0e-12            1.0e-12
Limits:                -                  -
Vector:                no                 no
Vector_Bounds:         -                  -
Null_Allowed:          yes                yes
PARAMETER_TABLE:
```

```
Parameter_Name:      rise_delay           fall_delay
Description:         "rise delay"         "fall delay"
Data_Type:           real                 real
Default_Value:       1.0e-9               1.0e-9
Limits:              [1.0e-12 -]          [1.0e-12 -]
Vector:              no                   no
Vector_Bounds:       -                    -
Null_Allowed:        yes                  yes
```

Description: The digital d-type latch is a one-bit, level-sensitive storage element which will output the value on the data line whenever the enable input line is high (ONE). The value on the data line is stored (i.e., held on the out line) whenever the enable line is low (ZERO).

In addition, asynchronous set and reset signals exist, and each of the four methods of changing the stored output of the d_dlatch (i.e., data changing with enable=ONE, enable changing to ONE from ZERO with a new value on data, raising set and raising reset) have separate delays associated with them. You may also specify separate rise and fall delay values that are added to those specified for the input lines; these allow for more faithful reproduction of the output characteristics of different IC fabrication technologies.

Note that any UNKNOWN inputs other than on the data line when enable=ZERO immediately cause the output to go UNKNOWN.

```
Example SPICE Usage:
a4 12 4 5 6 3 14 latch1
.model latch1 d_dlatch(data_delay = 13.0e-9 enable_delay = 22.0e-9
+                      set_delay = 25.0e-9
+                      reset_delay = 27.0e-9 ic = 2
+                      rise_delay = 10.0e-9 fall_delay = 3e-9)
```

13.4.17   Set-Reset Latch

```
NAME_TABLE:
C_Function_Name:     cm_d_srlatch
Spice_Model_Name:    d_srlatch
Description:         "digital sr-type latch"
PORT_TABLE:
Port Name:           s                    r
Description:         "set"                "reset"
Direction:           in                   in
Default_Type:        d                    d
Allowed_Types:       [d]                  [d]
Vector:              yes                  yes
Vector_Bounds:       [2 -]                r
Null_Allowed:        no                   no
PORT_TABLE:
Port Name:           enable
Description:         "enable"
Direction:           in
Default_Type:        d
Allowed_Types:       [d]
Vector:              no
Vector_Bounds:       -
Null_Allowed:        no
PORT_TABLE:
Port Name:           set                  reset
Description:         "set"                "reset"
Direction:           in                   in
Default_Type:        d                    d
Allowed_Types:       [d]                  [d]
Vector:              no                   no
```

```
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
PORT_TABLE:
Port Name:            out                    Nout
Description:          "data output"          "inverted data output"
Direction:            out                    out
Default_Type:         d                      d
Allowed_Types:        [d]                    [d]
Vector: no no
Vector_Bounds:        -                      -
Null_Allowed:         no                     no
PARAMETER_TABLE:
Parameter_Name:       sr_delay
Description:          "delay from s or r input change"
Data_Type:            real
Default_Value:        1.0e-9
Limits:               [1.0e-12 -]
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
PARAMETER_TABLE:
Parameter_Name:       enable_delay           set_delay
Description:          "delay from enable"    "delay from SET"
Data_Type:            real                   real
Default_Value:        1.0e-9                 1.0e-9
Limits:               [1.0e-12 -]            [1.0e-12 -]
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       reset_delay            ic
Description:          "delay from RESET"     "output initial state"
Data_Type:            real                   boolean
Default_Value:        1.0e-9                 0
Limits:               [1.0e-12 -]            -
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       sr_load                enable_load
Description:          "s & r input loads (F)" "enable load value (F)"
Data_Type:            real                   real
Default_Value:        1.0e-12                1.0e-12
Limits:               -                      -
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       set_load               reset_load
Description:          "set load value (F)"   "reset load (F)"
Data_Type:            real                   real
Default_Value:        1.0e-12                1.0e-12
Limits:               -                      -
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       rise_delay             fall_delay
```

```
Description:          "rise delay"            "fall delay"
Data_Type:            real                    real
Default_Value:        1.0e-9                  1.0e-9
Limits:               [1.0e-12 -]             [1.0e-12 -]
Vector:               no                      no
Vector_Bounds:        -                       -
Null_Allowed:         yes                     yes
```

Description: The digital sr-type latch is a one-bit, level-sensitive storage element which will output the value dictated by the state of the s and r pins whenever the enable input line is high (ONE). This value is stored (i.e., held on the out line) whenever the enable line is low (ZERO). The particular value chosen is as shown below:

```
s=ZERO, r=ZERO => out=current value (i.e., not change in output)
s=ZERO, r=ONE  => out=ZERO
s=ONE,  r=ZERO => out=ONE
s=ONE,  r=ONE  => out=UNKNOWN
```

Asynchronous set and reset signals exist, and each of the four methods of changing the stored output of the d srlatch (i.e., s/r combination changing with enable=ONE, enable changing to ONE from ZERO with an output-changing combination of s and r, raising set and raising reset) have separate delays associated with them. You may also specify separate rise and fall delay values that are added to those specified for the input lines; these allow for more faithful reproduction of the output characteristics of different IC fabrication technologies.

Note that any UNKNOWN inputs other than on the s and r lines when enable=ZERO immediately cause the output to go UNKNOWN.

```
Example SPICE Usage:
a4 12 4 5 6 3 14 16 latch2
.model latch2 d_srlatch(sr_delay = 13.0e-9 enable_delay = 22.0e-9
+                       set_delay = 25.0e-9
+                       reset_delay = 27.0e-9 ic = 2
+                       rise_delay = 10.0e-9 fall_delay = 3e-9)
```

13.4.18   State Machine

```
NAME_TABLE:
C_Function_Name:      cm_d_state
Spice_Model_Name:     d_state
Description:          "digital state machine"
PORT_TABLE:
Port Name:            in                      clk
Description:          "input"                 "clock"
Direction:            in                      in
Default_Type:         d                       d
Allowed_Types:        [d]                     [d]
Vector:               yes                     no
Vector_Bounds:        [1 -]                   -
Null_Allowed:         yes                     no
PORT_TABLE:
Port Name:            reset                   out
Description:          "reset"                 "output"
Direction:            in                      out
Default_Type:         d                       d
Allowed_Types:        [d]                     [d]
Vector:               no                      yes
Vector_Bounds:        -                       [1 -]
Null_Allowed:         yes                     no
PARAMETER_TABLE:
Parameter_Name:       clk_delay               reset_delay
```

```
Description:            "delay from CLK"        "delay from RESET"
Data_Type:              real                    real
Default_Value:          1.0e-9                  1.0e-9
Limits:                 [1.0e-12 -]             [1.0e-12 -]
Vector:                 no                      no
Vector_Bounds:          -                       -
Null_Allowed:           yes                     yes
PARAMETER_TABLE:        Parameter_Name:         state_file
Description:            "state transition specification file name"
Data_Type:              string
Default_Value:          "state.txt"
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           no
PARAMETER_TABLE:
Parameter_Name:         reset_state
Description:            "default state on RESET & at DC"
Data_Type:              int
Default_Value:          0
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           no
PARAMETER_TABLE:
Parameter_Name:         input_load
Description:            "input loading capacitance (F)"
Data_Type:              real
Default_Value:          1.0e-12
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
PARAMETER_TABLE:
Parameter_Name:         clk_load
Description:            "clock loading capacitance (F)"
Data_Type:              real
Default_Value:          1.0e-12
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
PARAMETER_TABLE:
Parameter_Name:         reset_load
Description:            "reset loading capacitance (F)"
Data_Type:              real
Default_Value:          1.0e-12
Limits:                 -
Vector:                 no
Vector_Bounds:          -
Null_Allowed:           yes
```

Description: The digital state machine provides for straightforward descriptions of clocked combina-
tional logic blocks with a variable number of inputs and outputs and with an unlimited number of
possible states. The model can be configured to behave as virtually any type of counter or clocked
combinational logic block and can be used to replace very large digital circuit schematics with an
identically functional but faster representation.

The d_state model is configured through the use of a state definition file (state.in) which resides in
a directory of your choosing. The file defines all states to be understood by the model, plus input

bit combinations which trigger changes in state. An example state.in file is shown below:

```
----------- begin file -------------
* This is an example state.in file. This file
* defines a simple 2-bit counter with one input. The
* value of this input determines whether the counter counts
* up (in = 1) or down (in = 0).
0 0s 0s 0 -> 3
         1 -> 1
1 0s 1z 0 -> 0
         1 -> 2
2 1z 0s 0 -> 1
         1 -> 3
3 1z 1z 0 -> 2
3 1z 1z 1 -> 0
----------------- end file ---------------
```

Several attributes of the above file structure should be noted. First, ALL LINES IN THE FILE MUST BE ONE OF FOUR TYPES. These are:

1. A comment, beginning with a "*" in the first column.

2. A header line, which is a complete description of the current state, the outputs corresponding to that state, an input value, and the state that the model willassume should that input be encountered. The first line of a state definition must ALWAYS be a header line.

3. A continuation line, which is a partial description of a state, consisting of an input value and the state that the model will assume should that input be encountered. Note that continuation lines may only be used after the initial header line definition for a state.

4. A line containing nothing but whitespace (space, formfeed, newline, carriage return, tab, vertical tab).

A line which is not one of the above will cause a file-loading error. Note that in the example shown, whitespace (any combination of blanks, tabs, commas) is used to separate values, and that the character "->" is used to underline the state transition implied by the input preceding it. This particular character is not critical in of itself, and can be replaced with any other character or non-broken combination of characters that you prefer (e.g. "==>", ">>", ":", "resolves_to", etc.)

The order of the output and input bits in the file is important; the first column is always interpreted to refer to the "zeroth" bit of input and output. Thus, in the file above, the output from state 1 sets out[0] to "0s", and out[1] to "1z".

The state numbers need not be in any particular order, but a state definition (which consists of the sum total of all lines which define the state, its outputs, and all methods by which a state can be exited) must be made on contiguous line numbers; a state definition cannot be broken into sub-blocks and distributed randomly throughout the file. On the other hand, the state definition can be broken up by as many comment lines as you desire.

Header files may be used throughout the state.in file, and continuation lines can be discarded completely if you so choose: continuation lines are primarily provided as a convenience.

```
Example SPICE Usage:
a4 [2 3 4 5] 1 12 [22 23 24 25 26 27 28 29] state1
.model state1 d_state(clk_delay = 13.0e-9 reset_delay = 27.0e-9
+                     state_file = newstate.txt reset_state = 2)
```

13.4.19 Frequency Divider

```
NAME_TABLE:
C_Function_Name:    cm_d_fdiv
Spice_Model_Name:   d_fdiv
Description:        "digital frequency divider"
PORT_TABLE:
```

```
Port Name:            freq_in                freq_out
Description:          "frequency input"      "frequency output"
Direction:            in                     out
Default_Type:         d                      d
Allowed_Types:        [d]                    [d]
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         no                     no
PARAMETER_TABLE:
Parameter_Name:       div_factor             high_cycles
Description:          "divide factor"        "# of cycles for high out"
Data_Type:            int                    int
Default_Value:        2                      1
Limits:               [1 -]                  [1 div_factor-1]
Vector:               no                     no
Vector_Bounds:        -                      -
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       i_count
Description:          "divider initial count value"
Data_Type:            int
Default_Value:        0
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
PARAMETER_TABLE:
Parameter_Name:       rise_delay             fall_delay
Description:          "rise delay"           "fall delay"
Data_Type:            real                   real
Default_Value:        1.0e-9                 1.0e-9
Limits:               [1.0e-12 -]            [1.0e-12 -]
Vector:               yes                    yes
Vector_Bounds:        in                     in
Null_Allowed:         yes                    yes
PARAMETER_TABLE:
Parameter_Name:       freq_in_load
Description:          "freq_in load value (F)"
Data_Type:            real
Default_Value:        1.0e-12
Limits:               -
Vector:               no
Vector_Bounds:        -
Null_Allowed:         yes
```

Description: The digital frequency divider is a programmable step-down divider which accepts an arbitrary divisor (div_factor), a duty-cycle term (high_cycles), and an initial count value (i_count). The generated output is synchronized to the rising edges of the input signal. Rise delay and fall delay on the outputs may also be specified independently.

```
Example SPICE Usage:
a4 3 7 divider
.model divider d_fdiv(div_factor = 5 high_cycles = 3
+                    i_count = 4 rise_delay = 23e-9
+                    fall_delay = 9e-9)
```

13.4.20   RAM

```
NAME_TABLE:
```

```
C_Function_Name:      cm_d_ram
Spice_Model_Name:     d_ram
Description:          "digital random-access memory"
PORT_TABLE:
Port Name:           data_in                data_out
Description:         "data input line(s)"   "data output line(s)"
Direction:           in                     out
Default_Type:        d                      d
Allowed_Types:       [d]                    [d]
Vector:              yes                    yes
Vector_Bounds:       [1 -]                  data_in
Null_Allowed:        no                     no
PORT_TABLE:
Port Name:           address                write_en
Description:         "address input line(s)" "write enable line"
Direction:           in                     in
Default_Type:        d                      d
Allowed_Types:       [d]                    [d]
Vector:              yes                    no
Vector_Bounds:       [1 -]                  -
Null_Allowed:        no                     no
PORT_TABLE:
Port Name:           select
Description:         "chip select line(s)"
Direction:           in
Default_Type:        d
Allowed_Types:       [d]
Vector:              yes
Vector_Bounds:       [1 16]
Null_Allowed:        no
PARAMETER_TABLE:
Parameter_Name:      select_value
Description:         "decimal active value for select line comparison"
Data_Type:           int
Default_Value:       1
Limits:              [0 32767]
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
PARAMETER_TABLE:
Parameter_Name:      ic
Description:         "initial bit state @ dc"
Data_Type:           int
Default_Value:       2
Limits:              [0 2]
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
PARAMETER_TABLE:
Parameter_Name:      read_delay
Description:         "read delay from address/select/write.en active"
Data_Type:           real
Default_Value:       100.0e-9
Limits:              [1.0e-12 -]
Vector:              no
Vector_Bounds:       -
Null_Allowed:        yes
PARAMETER_TABLE:
```

```
Parameter_Name:        data_load                 address_load
Description:            "data_in load value (F)" "addr. load value (F)"
Data_Type:             real                      real
Default_Value:         1.0e-12                   1.0e-12
Limits:                -                         -
Vector:                no                        no
Vector_Bounds:         -                         -
Null_Allowed:          yes                       yes
PARAMETER_TABLE:
Parameter_Name:        select_load
Description:            "select load value (F)"
Data_Type:             real
Default_Value:         1.0e-12
Limits:                -
Vector:                no
Vector_Bounds:         -
Null_Allowed:          yes
PARAMETER_TABLE:
Parameter_Name:        enable_load
Description:            "enable line load value (F)"
Data_Type:             real
Default_Value:         1.0e-12
Limits:                -
Vector:                no
Vector_Bounds:         -
Null_Allowed:          yes
```

Description: The digital RAM is an M-wide, N-deep random access memory element with programmable select lines, tristated data out lines, and a single write/~read line. The width of the RAM words (M) is set through the use of the word width parameter. The depth of the RAM (N) is set by the number of address lines input to the device. The value of N is related to the number of address input lines (P) by the following equation:

$$2^P = N$$

There is no reset line into the device. However, an initial value for all bits may be specified by setting the ic parameter to either 0 or 1. In reading a word from the ram, the read delay value is invoked, and output will not appear until that delay has been satisfied. Separate rise and fall delays are not supported for this device.

Note that UNKNOWN inputs on the address lines are not allowed during a write. In the event that an address line does indeed go unknown during a write, THE ENTIRE CONTENTS OF THE RAM WILL BE SET TO UNKNOWN. This is in contrast to the data in lines being set to unknown during a write; in that case, only the selected word will be corrupted, and this is corrected once the data lines settle back to a known value. Note that protection is added to the write en line such that extended UNKNOWN values on that line are interpreted as ZERO values. This is the equivalent of a read operation and will not corrupt the contents of the RAM. A similar mechanism exists for the select lines. If they are unknown, then it is assumed that the chip is not selected.

Detailed timing-checking routines are not provided in this model, other than for the enable delay and select delay restrictions on read operations. You are advised, therefore, to carefully check the timing into and out of the RAM for correct read and write cycle times, setup and hold times, etc. for the particular device they are attempting to model.

```
Example SPICE Usage:
a4 [3 4 5 6] [3 4 5 6] [12 13 14 15 16 17 18 19] 30 [22 23 24] ram2
.model ram2 d_ram(select_value = 2 ic = 2 read_delay = 80e-9)
```

13.4.21   Digital Source

```
NAME_TABLE:
C_Function_Name:       cm_d_source
```

```
Spice_Model_Name:    d_source
Description:         "digital signal source"
PORT_TABLE:
Port Name:           out
Description:         "output"
Direction:           out
Default_Type:        d
Allowed_Types:       [d]
Vector:              yes
Vector_Bounds:       -
Null_Allowed:        no
PARAMETER_TABLE:
Parameter_Name:      input_file
Description:         "digital input vector filename"
Data_Type:           string
Default_Value:       "source.txt"
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        no
PARAMETER_TABLE:
Parameter_Name:      input_load
Description:         "input loading capacitance (F)"
Data_Type:           real
Default_Value:       1.0e-12
Limits:              -
Vector:              no
Vector_Bounds:       -
Null_Allowed:        no
```

Description: The digital source provides for straightforward descriptions of digital signal vectors in a tabular format. The model reads input from the input file and, at the times specified in the file, generates the inputs along with the strengths listed.

The format of the input file is as shown below. Note that comment lines are delineated through the use of a single "*" character in the first column of a line. This is similar to the way the SPICE program handles comments.

```
* T        c  n  n  n . . .
* i        l  o  o  o . . .
* m        o  d  d  d . . .
* e        c  e  e  e . . .
*          k  a  b  c . . .
0.0000     Uu Uu Us Uu . . .
1.234e-9   0s 1s 1s 0z . . .
1.376e-9   0s 0s 1s 0z . . .
2.5e-7     1s 0s 1s 0z . . .
2.5006e-7  1s 1s 1s 0z . . .
5.0e-7     0s 1s 1s 0z . . .
```

Note that in the example shown, whitespace (any combination of blanks, tabs, commas) is used to separate the time and strength/state tokens. The order of the input columns is important; the first column is always interpreted to mean "time". The second through the N'th columns map to the out[0] through out[N-2] output nodes. A non-commented line which does not contain enough tokens to completely define all outputs for the digital source will cause an error. Also, time values must increase monotonically or an error will result in reading the source file.

Errors will also occur if a line exists in source.txt which is neither a comment nor vector line. The only exception to this is in the case of a line that is completely blank; this is treated as a comment (note that such lines often occur at the end of text within a file; ignoring these in particular prevents nuisance errors on the part of the simulator).

```
Example SPICE Usage:
a3 [2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17] input_vector
.model input_vector d_source(input_file = source_simple.text)
```

13.5   Predefined Node Types

The following prewritten node types are included with the XSPICE simulator. These, along with the digital node type built into the simulator, should provide you not only with valuable event-driven modeling capabilities, but also with examples to use for guidance in creating new UDN types.

13.5.1   Real Node Type

The "real" node type provides for event-driven simulation with double-precision floating point data. This type is useful for evaluating sampled-data filters and systems. The type implements all optional functions for User-Defined Nodes, including inversion and node resolution. For inversion, the sign of the value is reversed. For node resolution, the resultant value at a node is the sum of all values output to that node.

13.5.2   Int Node Type

The "int" node type provides for event-driven simulation with integer data. This type is useful for evaluating roundoff error effects in sampled-data systems. The type implements all optional functions for User-Defined Nodes, including inversion and node resolution. For inversion, the sign of the integer value is reversed. For node resolution, the resultant value at a node is the sum of all values output to that node.

Chapter 14

Verilog A Device models

14.1 Introduction

The ngspice-adms interface will implement extra HICUM level0 and level2 (HICUM model web page), MEXTRAM(MEXTRAM model web page), EKV(EKV model web page) and PSP(NXP MOS model 9 web page) models written in Verilog-A behaviour language.

14.2 adms

To compile Verilog-A compact models into ngspice-ready C models the the program admsXml is required. Details of this software are described in adms home page.

14.3 How to integrate a Verilog-A model into ngspice

14.3.1 How to setup a *.va model for ngspice

The root entry for new Verilog-A models is \src\spicelib\devices\adms. Below the modelname entry the Verilog-A code should reside in folder admsva. (e.g.: ng-spice-rework\src\spicelib\devices\adms\ekv\admsva\ekv.va) The file extension is fixed to .va.

Certain files must modified to create the interface to ngspice - see the guideline README.adms in the ngspice root.

14.3.2 Adding admsXml to your build environment

To facilitate the installation of adms, a source code package has been assembled for use with ngspice, available as a zip file for download. It is based on adms source code from the subversion repository downloaded on August 1st, 2010, and has been slightly modified (see ChangeLog).

Under OS LINUX (tested with SUSE 11.2, 64 bit) you may expand the zip file and run `./autogen_lin.sh`, followed by 'make' and 'make install'.

Under OS CYGWIN (tested with actual CYGWIN on MS Windows 7, 64 bit), please use `./autogen_cyg.sh`, followed by 'make' and 'make install'.

Under OS MINGW, a direct compilation would require the additional installation of perl module XML-LibXML which is not as straighforward as it should be. However you may start with a CYGWIN compile as decribed above. If you then go to your MSYS window, cd to the adms top directory and start `./mingw-compile.sh`, you will obtain admsXml.exe, copied to MSYS /bin, and you are ready to go. To facilitate installation under MS Windows, a `admsXml.exe` zipped binary is available. Just copy it to MSYS /bin directory and start working on your verilog models.

A short test of a successful installation is:
```
$ admsXml -v
$ [usage..]  release name="admsXml" version="2.3.0" date="Aug 4 2010" time="10:24:18"
```

Compilation of admsXml with MS Visual Studio is not possible, because the source code has variable declarations not only at the top of a block, but deliberately also in the following lines. This is o.k. by the C99 standard, but not supported by MS Visual Studio.

Chapter 15

Mixed-Level Simulation (ngspice with TCAD)

15.1 Cider

Ngspice implements mixed-level simulation through the merging of its code with CIDER (details see chapt. 29).

CIDER is a mixed-level circuit and device simulator that provides a direct link between technology parameters and circuit performance. A mixed-level circuit and device simulator can provide greater simulation accuracy than a stand-alone circuit or device simulator by numerically modeling the critical devices in a circuit. Compact models can be used for noncritical devices.

CIDER couples the latest version of SPICE3 (version 3F.2) [JOHN92] to a internal C-based device simulator, DSIM. SPICE3 provides circuit analyses, compact models for semiconductor devices, and an interactive user interface. DSIM provides accurate, one- and two-dimensional numerical device models based on the solution of Poisson's equation, and the electron and hole current-continuity equations. DSIM incorporates many of the same basic physical models found in the the Stanford two-dimensional device imulator PISCES [PINT85]. Input to CIDER consists of a SPICE-like description of the c ircuit and its compact models, and PISCES-like descriptions of the structures of numerically modeled devices. As a result, CIDER should seem familiar to designers already accustome to these two tools. For example, SPICE3F.2 input files should run without modification, producing identical results.

CIDER is based on the mixed-level circuit and device simulator CODECS [MAYA88] and is a replacement for this program. The basic algorithms of the two programs are the same. Some of the differences between CIDER and CODECS are described below. The CIDER input format has greater flexibility and allows increased access to physical model parameters. New physical models have been added to allow simulation of state-of-the-art devices. These include transverse field mobility degradation [GATE90] that is important in scaled-down MOSFETs and a polysilicon model for poly-emitter bipolar transistors. Temperature dependence has been included for most physical models over the range from -50°C to 150°C. The numerical models can be used to simulate all the basic types of semiconductor devices: resistors, MOS capacitors, diodes, BJTs, JFETs and MOSFETs. BJTs and JFETs can be modeled with or without a substrate contact. Support has been added for the management of device internal states. Post-processing of device states can be performed using the NUTMEG user interface of SPICE3. Previously computed states can be loaded into the program to provide accurate initial guesses for subssequent analyses. Finally, numerous small bugs have been discovered and fixed, and the program has been ported to a wider variety of computing platforms.

Berkeley tradition calls for the naming of new versions of programs by affixing a (number, letter, number) triplet to the end of the program name. Under this scheme, CIDER should instead be named CODECS2A.l. However, tradition has been broken in this case because major incompatibilities exist between the two programs and because it was observed that the acronym CODECS is already used in the analog design community to refer to coder-decoder circuits.

Details of the basic semiconductor equations and the physical models used by CIDER are not provided in this manual. Unfortunately, no other single source exists which describes all of the relevant background material. Comprehensive reviews of device simulation can be found in [PINT90] and the book [SELB84]. CODECS and its inversion-layer mobility model are described in [MAYA88] and LGATE90], respectively. PISCES and its models are described in [PINT85]. Temperature dependences for the PISCES models

used by CIDER are available in [SOLL90].

15.2   GSS, Genius

For LINUX users the cooperation of the TCAD software GSS with ngspice might be of interest, see http://ngspice.sourceforge.net/gss.html. This project is no longer maintained however, but has moved into the Genius simulator, still available as open source http://www.cogenda.com/downloads/category/7-genius-open-source-edition.html.

Chapter 16

Analyses and Output Control

The following command lines are for specifying analyses or plots within the circuit description file. Parallel commands exist in the interactive command interpreter (detailed in the following section). Specifying analyses and plots (or tables) in the input file is useful for batch runs. Batch mode is entered when either the `-b` option is given or when the default input source is redirected from a file. In batch mode, the analyses specified by the control lines in the input file (e.g. ".ac", ".tran", etc.) are immediately executed (unless ".control" lines exists; see the section on the interactive command interpreter). If the `-r` rawfile option is given then all data generated is written to a ngspice rawfile. The rawfile may be read by either the interactive mode of ngspice or by ngnutmeg; see the previous section for details. In this case, the `.save` line (see 16.4) may be used to record the value of internal device variables (see Appendix, chapter 30).

If a rawfile is not specified, then output plots (in "line-printer" form) and tables can be printed according to the `.print`, `.plot`, and `.four` control lines, described in chapter 16.4.

16.1 Simulator Variables (.options)

Various parameters of the simulations available in Ngspice can be altered to control the accuracy, speed, or default values for some devices. These parameters may be changed via the "option" command (described in chapt. 18.4.34) or via the ".options" line:

General form:

```
.options opt1 opt2 ... (or opt=optval ...)
```

Examples:

```
.options reltol=.005 trtol=8
```

The options line allows the user to reset program control and user options for specific simulation purposes. Options specified to Ngspice via the "option" command (see chapt.) are also passed on as if specified on a `.options` line. Any combination of the following options may be included, in any order. "x" (below) represents some positive number.

16.1.1 General Options

ACCT causes accounting and run time statistics to be printed.

NOACCT no printing of statistics, no printing of the Initial Transient Solution.

NOINIT suppresses only printing of the Initial Transient Solution, maybe combined with ACCT.

LIST causes the summary listing of the input data to be printed.

NOMOD suppresses the printout of the model parameters.

NOPAGE suppresses page ejects.

NODE causes the printing of the node table.

OPTS causes the option values to be printed.

TEMP=x Resets the operating temperature of the circuit. The default value is 27 ˚C (300K). TEMP can be overridden by a temperature specification on any temperature dependent instance

TNOM=x resets the nominal temperature at which device parameters are measured. The default value is 27 ˚C (300 deg K). TNOM can be overridden by a specification on any temperature dependent device model.

16.1.2 DC Solution Options

The following options controls properties pertaining to DC analysis and algorithms. Since transient analysis is based on DC many of the options affect the latter one.

ABSTOL=x resets the absolute current error tolerance of the program. The default value is 1 pA.

GMIN=x resets the value of GMIN, the minimum conductance allowed by the program. The default value is 1.0e-12.

ITL1=x resets the dc iteration limit. The default is 100.

ITL2=x resets the dc transfer curve iteration limit. The default is 50.

KEEPOPINFO Retain the operating point information when either an AC, Distortion, or Pole-Zero analysis is run. This is particularly useful if the circuit is large and you do not want to run a (redundant) ".OP" analysis.

PIVREL=x resets the relative ratio between the largest column entry and an acceptable pivot value. The default value is 1.0e-3. In the numerical pivoting algorithm the allowed minimum pivot value is determined by EPSREL=AMAX1(PIVREL*MAXVAL, PIVTOL) where MAXVAL is the maximum element in the column where a pivot is sought (partial pivoting).

PIVTOL=x resets the absolute minimum value for a matrix entry to be accepted as a pivot. The default value is 1.0e-13.

RELTOL=x resets the relative error tolerance of the program. The default value is 0.001 (0.1%).

RSHUNT=x introduces a resistor from each analog node to ground. The value of the resistor should be high enough to not interfere with circuit operations.

VNTOL=x resets the absolute voltage error tolerance of the program. The default value is 1 μV.

Matrix Conditioning info

In most SPICE-based simulators, problems can arise with certain circuit topologies. One of the most common problems is the absense of a DC path to ground at some node. This may happen, for example, when two capacitors are connected in series with no other connection at the common node or when certain code models are cascaded. The result is an ill-conditioned or nearly singular matrix that prevents the simulation from completing. XSPICE introduces a new "rshunt" option to help eliminate this problem. When used, this option inserts resistors to ground at all the analog nodes in the circuit. In general, the value of "rshunt" should be set to some very high resistance (e.g. 1000 Meg Ohms or greater) so that the operation of the circuit is essentially unaffected, but the matrix problems are corrected. If you should encounter a "no DC path to ground" or a "matrix is nearly singular" error message with your circuit, you should try adding the following .option card to your circuit description deck.

```
.option rshunt = 1.0e12
```

Usually a value of 1.0e12 is sufficient to correct the matrix problems. However, if you still have problems, you may wish to try lowering this value to 1.0e10 or 1.0e9.

16.1.3 Transient Analysis Options

CHGTOL=x resets the charge tolerance of the program. The default value is 1.0e-14.

CONVSTEP=x relative step limit applied to code models.

CONVABSSTEP=x absolute step limit applied to code models.

GMINSTEPS=x [*] sets number of Gmin steps to be attempted. If the value is set to zero, the gmin stepping algorithm is disabled. In such case the source stepping algorithm becomes the standard when the standard proceeudre fails to converge to a solution.

ITL3=x resets the lower transient analysis iteration limit. the default value is 4. (Note: not implemented in Spice3).

ITL4=x resets the transient analysis timepoint iteration limit. the default is 10.

ITL5=x resets the transient analysis total iteration limit. the default is 5000. Set ITL5=0 to omit this test. (Note: not implemented in Spice3).

ITL6=x [*] synonym for SRCSTEPS.

MAXEVITER=x sets the number of event iterations that are allowed at an analysis point

MAXOPALTER=x specifies the maximum number of analog/event alternations that the simulator can use in solving a hybrid circuit.

MAXORD=x [*] specifies the maximum order for the numerical integration method used by SPICE. Possible values for the Gear method are from 2 (the default) to 6. Using the value 1 with the trapezoidal method specifies backward Euler integration.

METHOD=name sets the numerical integration method used by SPICE. Possible names are "Gear" or "trapezoidal" (or just "trap"). The default is trapezoidal.

NOOPALTER=TRUE|FALSE if set to false alternations between analog/event are enabled.

RAMPTIME=x this options sets the rate of change of independent supplies and code model indcutors and capacitors with initial conditions specified.

SRCSTEPS=x [*] a non-zero value causes SPICE to use a source-stepping method to find the DC operating point. Its value specifies the number of steps.

TRTOL=x resets the transient error tolerance. The default value is 7. This parameter is an estimate of the factor by which ngspice overestimates the actual truncation error. If XSPICE is enabled and 'A' devices included, the value is internally set to 1 for higher precision. This will cost a factor of two in cpu time during transient analysis.

Supply Ramping example

A supply ramping function is provided by the simulator as an option to a transient analysis to simulate the turn-on of power supplies to a board level circuit. The supply ramping function linearly ramps the values of all independent sources and the capacitor and inductor code models (code model extension) with initial conditions toward their final value at a rate which you define. A complete ngspice deck example of usage of the `ramptime` option is shown below.

```
Example:
Supply ramping option
*
* This circuit demonstrates the use of the option
* "ramptime" which ramps independent sources and the
* capacitor and inductor initial conditions from
* zero to their final value during the time period
* specified.
*
*
.tran 0.1 5
.option ramptime=0.2
* a1 1 0 cap
.model cap capacitor (c=1000uf ic=1)
r1 1 0 1k
*
a2 2 0 ind
.model ind inductor (l=1H ic=1)
r2 2 0 1.0
*
v1 3 0 1.0
r3 3 0 1k
*
i1 4 0 1e-3
r4 4 0 1k
*
v2 5 0 0.0 sin(0 1 0.3 0 0 45.0)
r5 5 0 1k
*
.end
```

16.1.4   MOSFET Specific options

BADMOS3 Use the older version of the MOS3 model with the "kappa" discontinuity.

DEFAD=x resets the value for MOS drain diffusion area; the default is 0.0.

DEFAS=x resets the value for MOS source diffusion area; the default is 0.0.

DEFL=x resets the value for MOS channel length; the default is 100.0 μm.

DEFW=x resets the value for MOS channel width; the default is 100.0 μm.

16.1.5   Transmission Lines Specific Options

TRYTOCOMPACT Applicable only to the LTRA model (see 7.2.1). When specified, the simulator tries to condense LTRA transmission lines' past history of input voltages and currents.

16.1.6   Precedence of option and .options commands

There are various ways to set the above mentioned options in Ngspice. If no `option` or `.options` lines are set by the user, internal default values are given for each of the simulator variables.

You may set options in the init files **spinit** or **.spiceinit** via the `option` command (see chapt. 18.4.34). The values given here will supersede the default values. If you set options via the `.options` line in your input file, their values will supersede the default and init file data. Finally if you set options inside a .control ... .endc section, these values will supersede any values of the respective simulator variables given so far.

16.2   Initial Conditions

16.2.1   .NODESET: Specify Initial Node Voltage Guesses

General form:

`.NODESET V(NODNUM)=VAL V(NODNUM)=VAL ...`

Examples:

`.NODESET V(12)=4.5 V(4)=2.23`

The `.nodeset` line helps the program find the dc or initial transient solution by making a preliminary pass with the specified nodes held to the given voltages. The restriction is then released and the iteration continues to the true solution. The `.nodeset` line may be necessary for convergence on bistable or a-stable circuits. In general, this line should not be necessary.

16.2.2   .IC: Set Initial Conditions

General form:

`.ic v(nodnum)=val v(nodnum)=val ...`

Examples:

`.ic v(11)=5 v(4)=-5 v(2)=2.2`

The `.ic` line is for setting transient initial conditions. It has two different interpretations, depending on whether the `uic` parameter is specified on the `.tran` control line. Also, one should not confuse this line with the `.nodeset` line. The `.nodeset` line is only to help dc convergence, and does not affect final bias solution (except for multi-stable circuits). The two interpretations of this line are as follows:

1. When the `uic` parameter is specified on the `.tran` line, then the node voltages specified on the `.ic` control line are used to compute the capacitor, diode, BJT, JFET, and MOSFET initial conditions. This is equivalent to specifying the `ic=...` parameter on each device line, but is much more convenient. The `ic=...` parameter can still be specified and takes precedence over the `.ic` values. Since no dc bias (initial transient) solution is computed before the transient analysis, one should take care to specify all dc source voltages on the `.ic` control line if they are to be used to compute device initial conditions.

2. When the `uic` parameter is not specified on the `.tran` control line, the dc bias (initial transient) solution is computed before the transient analysis. In this case, the node voltages specified on the `.ic` control line is forced to the desired initial values during the bias solution. During transient analysis, the constraint on these node voltages is removed. This is the preferred method since it allows ngspice to compute a consistent dc solution.

16.3   Analyses

16.3.1   .AC: Small-Signal AC Analysis

General form:

```
.ac dec nd fstart fstop
.ac oct no fstart fstop
.ac lin np fstart fstop
```

Examples:

```
.ac dec 10 1 10K
.ac dec 10 1K 100MEG
.ac lin 100 1 100HZ
```

`dec` stands for decade variation, and `nd` is the number of points per decade. `oct` stands for octave variation, and `no` is the number of points per octave. `lin` stands for linear variation, and `np` is the number of points. `fstart` is the starting frequency, and `fstop` is the final frequency. If this line is included in the input file, ngspice performs an AC analysis of the circuit over the specified frequency range. Note

that in order for this analysis to be meaningful, at least one independent source must have been specified with an ac value. Typically it does not make much sense to specify more than one ac source. If you do, the result will be a superposition of all sources, thus difficult to interpret.

Example:

```
Basic RC circuit
r 1 2 1.0
c 2 0 1.0
vin 1 0 dc 0 ac 1  $ <—— the ac source
.options noacct
.ac dec 10 .01 10
.plot ac  vdb(2) xlog
.end
```

In this ac (or 'small signal') analysis all non-linear devices are linearized around their actual dc operating point. All Ls and Cs get their imaginary value, depending on the actual frequency step. Each output vector will be calculated relative to the input voltage (current) given by the ac value (Vin equals to 1 in the example above). The resulting node voltages (and branch currents) are complex vectors. Therefore you have to be careful using the plot command. Especially you may use the variants of vxx(node) described in chapter 16.4.2 like `vdb(2)` (see example above).

16.3.2   .DC: DC Transfer Function

General form:

```
.dc srcnam vstart vstop vincr [src2 start2 stop2 incr2]
```

Examples:

```
.dc VIN 0.25 5.0 0.25
.dc VDS 0 10 .5 VGS 0 5 1
.dc VCE 0 10 .25 IB 0 10U 1U
.dc RLoad 1k 2k 100
.dc TEMP −15 75 5
```

The `.dc` line defines the dc transfer curve source and sweep limits (again with capacitors open and inductors shorted). `srcnam` is the name of an independent voltage or current source, a resistor or the circuit temperature. `vstart`, `vstop`, and `vincr` are the starting, final, and incrementing values respectively. The first example causes the value of the voltage source VIN to be swept from 0.25 Volts to 5.0 Volts in increments of 0.25 Volts. A second source (`src2`) may optionally be specified with associated sweep parameters. In this case, the first source is swept over its range for each value of the second source. This option can be useful for obtaining semiconductor device output characteristics. See the example circuit description on transistor characteristics (21.3).

16.3.3   .DISTO: Distortion Analysis

General form:

```
.disto dec nd fstart fstop <f2overf1>
.disto oct no fstart fstop <f2overf1>
.disto lin np fstart fstop <f2overf1>
```

Examples:

```
.disto dec 10 1kHz 100Mhz
.disto dec 10 1kHz 100Mhz 0.9
```

The `.disto` line does a small-signal distortion analysis of the circuit. A multi-dimensional Volterra series analysis is done using multi-dimensional Taylor series to represent the nonlinearities at the operating point. Terms of up to third order are used in the series expansions.

If the optional parameter `f2overf1` is not specified, `.disto` does a harmonic analysis - i.e., it analyses distortion in the circuit using only a single input frequency F_1, which is swept as specified by arguments of the `.disto` command exactly as in the `.ac` command. Inputs at this frequency may be present at more

than one input source, and their magnitudes and phases are specified by the arguments of the `distof1` keyword in the input file lines for the input sources (see the description for independent sources). (The arguments of the `distof2` keyword 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 , vs. the input frequency F_1 as it is swept. (A value of 1 (as a complex distortion output) 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 distortion component desired ($2F_1$ or $3F_1$) can be selected using commands in ngnutmeg, and then printed or plotted. (Normally, one is interested primarily in the magnitude of the harmonic components, so the magnitude of the AC distortion value is looked at). It should be noted that these are the AC values of the actual harmonic components, and are not equal to HD2 and HD3. To obtain HD2 and HD3, one must divide by the corresponding AC values at F_1, obtained from an `.ac` line. This division can be done using ngnutmeg commands.

If the optional `f2overf1` parameter is specified, it should be a real number between (and not equal to) 0.0 and 1.0; in this case, `.disto` does a spectral analysis. It considers the circuit with sinusoidal inputs at two different frequencies F_1 and F_2. F_1 is swept according to the `.disto` control line options exactly as in the `.ac` control line. F_2 is kept fixed at a single frequency as F_1 sweeps - the value at which it is kept fixed is equal to `f2overf1` times `fstart`. Each independent source in the circuit may potentially have two (superimposed) sinusoidal inputs for distortion, at the frequencies F_1 and F_2. The magnitude and phase of the F_1 component are specified by the arguments of the `distof1` keyword in the source's input line (see the description of independent sources); the magnitude and phase of the F_2 component are specified by the arguments of the `distof2` keyword. The analysis produces plots of all node voltages/branch currents at the intermodulation product frequencies $F_1 + F_2$, $F_1 - F_2$, and $(2F_1) - F_2$, vs the swept frequency F_1. The IM product of interest may be selected using the `setplot` command, and displayed with the print and plot commands. It is to be noted as in the harmonic analysis case, the results are the actual AC voltages and currents at the intermodulation frequencies, and need to be normalized with respect to `.ac` values to obtain the IM parameters.

If the `distof1` or `distof2` keywords are missing from the description of an independent source, then 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 should be specified in degrees.

It should be carefully noted that the number `f2overf1` should ideally be an irrational number, and that since this is not possible in practice, efforts should be made to keep the denominator in its fractional representation as large as possible, certainly 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 why, consider the cases where `f2overf1` is $49/100$ and $1/2$. In a spectral analysis, the outputs produced are at $F_1 + F_2$, $F_1 - F_2$ and $2F_1 - F_2$. In the latter case, $F_1 - F_2 = F_2$, so the result at the $F_1 - F_2$ component is erroneous because there is 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 problem is not there in the case where `f2overf1` $= 49/100$, because $F_1 - F_2 = 51/100$ $F_1 <> 49/100$ $F_1 = F_2$. In this case, there are two very closely spaced frequency components at F_2 and $F_1 - F_2$. One of the advantages of the Volterra series technique is that it computes distortions at mix frequencies expressed symbolically (i.e. $nF_1 + mF_2$), therefore one is able to obtain the strengths of distortion components accurately even if the separation between them is very small, as opposed to transient analysis for example. The disadvantage is of course that if two of the mix frequencies coincide, the results are not merged together and presented (though this could presumably be done as a postprocessing step). Currently, the interested user should keep track of the mix frequencies himself or herself and add the distortions at coinciding mix frequencies together should it be necessary.

16.3.4   .NOISE: Noise Analysis

General form:

```
. noise  v( output  <,ref >)  src  ( dec  |  lin  |  oct )  pts  fstart  fstop
+ <pts_per_summary>
```

Examples:

```
. noise  v(5)  VIN  dec  10  1kHZ  100Mhz
. noise  v(5 ,3)  V1  oct  8  1.0  1.0 e6  1
```

The .noise line does a noise analysis of the circuit. output is the node at which the total output noise is desired; if ref is specified, then the noise voltage v(output) - v(ref) is calculated. By default, ref is assumed to be ground. src is the name of an independent source to which input noise is referred. pts, fstart and fstop are .ac type parameters that specify the frequency range over which plots are desired. pts_per_summary is an optional integer; if specified, the noise contributions of each noise generator is produced every pts_per_summary frequency points. The .noise control line produces two plots:

1. one for the Noise Spectral Density curves and

2. one for the total Integrated Noise over the specified frequency range.

All noise voltages/currents are in squared units (V^2/Hz and A^2/Hz for spectral density, V^2 and A^2 for integrated noise).

16.3.5   .OP: Operating Point Analysis

General form:

.op

The inclusion of this line in an input file directs ngspice to determine the dc operating point of the circuit with inductors shorted and capacitors opened.

Note: a DC analysis is automatically performed prior to a transient analysis to determine the transient initial conditions, and prior to an AC small-signal, Noise, and Pole-Zero analysis to determine the linearized, small-signal models for nonlinear devices (see the KEEPOPINFO variable 16.1.2).

16.3.6   .PZ: Pole-Zero Analysis

General form:

```
.pz  node1  node2  node3  node4  cur  pol
.pz  node1  node2  node3  node4  cur  zer
.pz  node1  node2  node3  node4  cur  pz
.pz  node1  node2  node3  node4  vol  pol
.pz  node1  node2  NODE3  node4  vol  zer
.pz  node1  node2  node3  node4  vol  pz
```

Examples:

```
.pz  1  0  3  0  cur  pol
.pz  2  3  5  0  vol  zer
.pz  4  1  4  1  cur  pz
```

cur stands for a transfer function of the type (output voltage)/(input current) while vol stands for a transfer function of the type (output voltage)/(input voltage). pol stands for pole analysis only, zer for zero analysis only and pz for both. This feature is provided mainly because if there is a nonconvergence in finding poles or zeros, then, at least the other can be found. Finally, node1 and node2 are the two input nodes and node3 and node4 are the two output nodes. Thus, there is complete freedom regarding the output and input ports and the type of transfer function.

In interactive mode, the command syntax is the same except that the first field is pz instead of .pz. To print the results, one should use the command "print all".

16.3.7 .SENS: DC or Small-Signal AC Sensitivity Analysis

General form:
@example

```
.SENS OUTVAR
.SENS OUTVAR AC DEC ND FSTART FSTOP
.SENS OUTVAR AC OCT NO FSTART FSTOP
.SENS OUTVAR AC LIN NP FSTART FSTOP
```

@end example
Examples:
@example

```
.SENS V(1,OUT)
.SENS V(OUT) AC DEC 10 100 100k
.SENS I(VTEST)
```

@end example

The sensitivity of OUTVAR to all non-zero device parameters is calculated when the SENS analysis is specified. OUTVAR is a circuit variable (node voltage or voltage-source branch current). The first form calculates sensitivity of the DC operating-point value of OUTVAR. The second form calculates sensitivity of the AC values of OUTVAR. The parameters listed for AC sensitivity are the same as in an AC analysis (see ".AC" above). The output values are in dimensions of change in output per unit change of input (as opposed to percent change in output or per percent change of input).

16.3.8 .TF: Transfer Function Analysis

General form:

```
.tf outvar insrc
```

Examples:

```
.tf v(5, 3) VIN
.tf i(VLOAD) VIN
```

The `.tf` line defines the small-signal output and input for the dc small-signal analysis. `outvar` is the smallsignal output variable and `insrc` is the small-signal input source. If this line is included, ngspice computes the dc small-signal value of the transfer function (output/input), input resistance, and output resistance. For the first example, ngspice would compute the ratio of V(5, 3) to VIN, the small-signal input resistance at VIN, and the smallsignal output resistance measured across nodes 5 and 3.

16.3.9 .TRAN: Transient Analysis

General form:

```
.tran tstep tstop <tstart <tmax>> <uic>
```

Examples:

```
.tran 1ns 100ns
.tran 1ns 1000ns 500ns
.tran 10ns 1us
```

`tstep` is the printing or plotting increment for lineprinter output. For use with the post-processor, `tstep` is the suggested computing increment. `tstop` is the final time, and `tstart` is the initial time. If `tstart` is omitted, it is assumed to be zero. The transient analysis always begins at time zero. In the interval <zero, `tstart`>, the circuit is analyzed (to reach a steady state), but no outputs are stored. In the interval <`tstart`, `tstop`>, the circuit is analyzed and outputs are stored. `tmax` is the maximum stepsize that ngspice uses; for default, the program chooses either `tstep` or (`tstop`-`tstart`)/50.0, whichever is smaller. `tmax` is useful when one wishes to guarantee a computing interval which is smaller than the printer increment, `tstep`.

`uic` (use initial conditions) is an optional keyword which indicates that the user does not want ngspice to solve for the quiescent operating point before beginning the transient analysis. If this keyword is

specified, ngspice uses the values specified using IC=... on the various elements as the initial transient condition and proceeds with the analysis. If the `.ic` control line has been specified, then the node voltages on the `.ic` line are used to compute the initial conditions for the devices. Look at the description on the `.ic` control line for its interpretation when `uic` is not specified.

16.3.10   .MEAS: Measurements after Op, Ac and Transient Analysis

The .meas or .measure statement is used to analyse the output data of a tran, ac, or dc simulation. The command is executed immediately after the simulation has finished.

.meas analysis may not be used in batch mode (`-b` command line option), if an output file (rawfile) is given at the same time (`-r rawfile` command line option). In this batch mode ngspice will write its simulation output data directly to the output file. The data is not kept in memory, thus is no longer available for further analysis. This is made to allow a very large output stream with only a relatively small memory usage. For .meas to be active you need to run the batch mode with a `.plot` or `.print` command. A better alternative may be to start ngspice in interactive mode.

If you need batch like operation, you may add a `.control ...   .endc` section to the input file:
Example:

```
∗input file
...
.tran 1ns 1000ns
...
∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗
.control
run
write outputfile data
.endc
∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗∗
.end
```

and start ngspice in interactive mode, e.g. by running the command
```
ngspice inputfile .
```
.meas<ure> then prints its user-defined data analysis to the standard output. The analysis includes propagation, delay, rise time, fall time, peak-to-peak voltage, minimum or maximum voltage, the integral or derivative over a specified period and several other user defined values.

The measure type **{DC|AC|TRAN|SP}** depends on the data which are to be evaluated, either originating from a dc analysis, an ac analysis, a transient simulation or a spectrum from the `spec` or `fft` commands.

result will be a vector containing the result of the measurement. **trig_variable**, **targ_variable**, and **out_variable** are vectors stemming from the simulation, e.g. a voltage vector v(out).

VAL=val expects a real number val. It may be as well a parameter in " or {} expanding to a real number.

TD=td and **AT=time** expect a time value.

CROSS=# requires an integer number #. **CROSS=LAST** is possible as well. The same is expected by **RISE** and **FALL**.
```
*
***********
```
Be careful because not all of the .measure commands have been implemented so far!
'out_variable=out_variable2' is missing for WHEN and FIND
'deriv' is missing
```
***********
*
```
In the following lines you will get some explanation on the .measure commands. A simple simulation file with two sines of different frequencies may serve as an example. The transient simulation delivers time as the independent variable and two voltages as output (dependent variables).

Input file:

```
File: simple-meas-tran.sp
* Simple .measurement examples
* transient simulation of two sine signals with different frequencies
vac1 1 0 DC 0 sin(0 1 1k 0 0)
vac2 2 0 DC 0 sin(0 1.2 0.9k 0 0)
.tran 10u 5m
*
.measure tran ... $ for the different inputs see below!
*
.control
run
plot v(1) v(2)
.endc
.end
```

After displaying the general syntax of the .measurement statement, some examples are posted, referring to the input file given above.

.measure according to general form 1 measures the difference in dc voltage, frequency or time between two points selected from one or two output vectors. The current examples all are using transient simulation. Measurements start after a delay time td. If you run other examples with ac simulation or spectrum analysis, time may be replaced by frequency, after a dc simulation the independent variable may become a voltage or current.

General form 1:

```
.MEASURE {DC|AC|TRAN|SP} result TRIG trig_variable VAL=val <TD=td> <CROSS=#
    | CROSS=LAST> <RISE=# | RISE=LAST> <FALL=# | FALL=LAST> <TRIG AT=time>
    TARG targ_variable VAL=val <TD=td> <CROSS=# | CROSS=LAST> <RISE=# | RISE
    =LAST> <FALL=# | FALL=LAST> <TARG AT=time>
```

Measure statement example (for use in the input file given above):

`.measure tran tdiff TRIG v(1) VAL=0.5 RISE=1 TARG v(1) VAL=0.5 RISE=2`

measures the time difference between v(1) reaching 0.5 V for the first time on its first rising slope (TRIG) versus reaching 0.5 V again on its second rising slope (TARG). I.e. it measures the signal period.

Output:

`tdiff = 1.000000e-003 targ= 1.083343e-003 trig= 8.334295e-005`

Measure statement example:

`.measure tran tdiff TRIG v(1) VAL=0.5 RISE=1 TARG v(1) VAL=0.5 RISE=3`

measures the time difference between v(1) reaching 0.5 V for the first time on its rising slope versus reaching 0.5 V on its rising slope for the third time (i.e. two periods).

Measure statement:

`.measure tran tdiff TRIG v(1) VAL=0.5 RISE=1 TARG v(1) VAL=0.5 FALL=1`

measures the time difference between v(1) reaching 0.5V for the first time on its rising slope versus reaching 0.5 V on its first falling slope.

Measure statement:

`.measure tran tdiff TRIG v(1) VAL=0 FALL=3 TARG v(2) VAL=0 FALL=3`

measures the time difference between v(1) reaching 0V its third falling slope versus v(2) reaching 0 V on its third falling slope.

Measure statement:

`.measure tran tdiff TRIG v(1) VAL=-0.6 CROSS=1 TARG v(2) VAL=-0.8 CROSS=1`

measures the time difference between v(1) crossing -0.6 V for the first time (any slope) versus v(2) crossing -0.8 V for the first time (any slope).

Measure statement:

`.measure tran tdiff TRIG AT=1m TARG v(2) VAL=-0.8 CROSS=3`

measures the time difference between the time point 1ms versus the time when v(2) crosses -0.8 V for the third time (any slope).

The **FIND** and **WHEN** functions allow to measure any dependent or independent time, frequency, or dc parameter, when two signals cross each other or a signal crosses a given value. Measurements start after a delay TD and may be restricted to a range between FROM and TO.

General form 2:

.MEASURE {DC|AC|TRAN|SP} result WHEN out_variable=val <TD=td> <FROM=val> <
    TO=val> <CROSS=# | CROSS=LAST> <RISE=# | RISE=LAST> <FALL=# | FALL=LAST>

Measure statement:
`.measure tran teval WHEN v(2)=0.7 CROSS=LAST`
measures the time point when v(2) crosses 0.7 V for the last time (any slope).
General form 3:

.MEASURE {DC|AC|TRAN|SP} result WHEN out_variable=out_variable2 <TD=td> <
    FROM=val> <TO=val> <CROSS=# | CROSS=LAST> <RISE=# | RISE=LAST> <FALL=# |
    FALL=LAST>

Measure statement:
`.measure tran teval WHEN v(2)=v(1) RISE=LAST`
measures the time point when v(2) and v(1) are equal, v(2) rising for the last time. **<− FIXME:
Not yet implemented!**
General form 4:

.MEASURE {DC|AC|TRAN|SP} result FIND out_variable WHEN out_variable2=val <
    TD=td> <FROM=val> <TO=val> <CROSS=# | CROSS=LAST> <RISE=# | RISE=LAST> <
    FALL=# | FALL=LAST>

Measure statement:
`.measure tran yeval FIND v(2) WHEN v(1)=-0.4 FALL=LAST`
returns the dependent (y) variable drawn from v(2) at the time point when v(1) equals a value of
-0.4, v(2) falling for the last time. **<− FIXME: Not yet implemented!**
General form 5:

.MEASURE {DC|AC|TRAN|SP} result FIND out_variable WHEN out_variable2=
    out_variable3 <TD=td> <CROSS=# | CROSS=LAST> <RISE=#|RISE=LAST> <FALL
    =#|FALL=LAST>

<− FIXME: Not yet implemented!
General form 6:

.MEASURE {DC|AC|TRAN|SP} result FIND out_variable AT=val

Measure statement:
`.measure tran yeval FIND v(2) AT=2m`
returns the dependent (y) variable drawn from v(2) at the time point 2 ms (given by AT=time).
General form 7:

.MEASURE {DC|AC|TRAN|SP} result {AVG|MIN|MAX|PP|RMS|MIN_AT|MAX_AT}
    out_variable <TD=td> <FROM=val> <TO=val>

Measure statements:
`.measure tran ymax MAX v(2) from=2m to=3m`
returns the maximum value of v(2) inside the time interval between 2 ms and 3 ms.
`.measure tran tymax MAX_AT v(2) from=2m to=3m`
returns the time point of the maximum value of v(2) inside the time interval between 2 ms and 3 ms.
`.measure tran ypp PP v(1) from=2m to=4m`
returns the peak to peak value of v(1) inside the time interval between 2 ms and 4 ms.
`.measure tran yrms RMS v(1) from=2m to=4m`
returns the root mean square value of v(1) inside the time interval between 2 ms and 4 ms.
`.measure tran yavg AVG v(1) from=2m to=4m`
returns the average value of v(1) inside the time interval between 2 ms and 4 ms.
General form 8:

.MEASURE {DC|AC|TRAN|SP} result INTEG<RAL> out_variable <TD=td> <FROM=val>
    <TO=val>

Measure statement:
`.measure tran yint INTEG v(2) from=2m to=3m`
returns the area under v(2) inside the time interval between 2 ms and 3 ms.

General form 9:
.MEASURE {DC|AC|TRAN|SP} result   param='expression'

Measure statement:

.param fval=5

.measure tran yadd param='fval + 7'

will evaluate the given expression fval + 7 and return the value 12.

.param vout_diff=50k

.meas tran bw_chk param='(vout_diff < 100k) ? 1 : 0'

will evaluate the given ternary function and return the value 1.

'Expression' is evaluated according to the rules given in chapt. 2.8.5 during start up of ngspice. Thus it may not contain vectors like v(10), e.g. anything resulting from a simulation.

The par('expression') option (16.4.6) allows to use algebraic expressions in the .measure lines. Every out_variable may be replaced by par('expression') within the general forms 1-9 described above. Internally par('expression') will be substituted by a vector according to the rules of the B source (chapt. 6.1). A typical example of the general form is shown below:

General form 10:
.MEASURE {DC|AC|TRAN|SP} result   FIND par('expression') AT=val

Measure statement:

.measure tran vtest find par('(v(2)*v(1))') AT=2.3m

will return the product of the two voltages at timne point 2.3 ms.

General form:
.MEASURE {DC|AC|TRAN|SP} result DERIV<ATIVE> out_variable AT=val

.MEASURE {DC|AC|TRAN|SP} result DERIV<ATIVE> out_variable WHEN
     out_variable2=val   + <TD=td>
+ <CROSS=# | CROSS=LAST> <RISE=#|RISE=LAST> <FALL=#|FALL=LAST>

.MEASURE {DC|AC|TRAN|SP} result DERIV<ATIVE> out_variable
+ WHEN out_variable2=out_variable3
+ <TD=td>
+ <CROSS=# | CROSS=LAST> <RISE=#|RISE=LAST> <FALL=#|FALL=LAST>

.MEASURE {DC|AC|TRAN|SP} result DERIV<ATIVE> ... is not yet available.

Some other examples, also showing the use of parameters, are given below. Corresponding demonstration input files are distributed with ngspice in folder /examples/measure.

Other examples:

```
.meas tran inv_delay2 trig v(in) val='vp/2' td=1n fall=1 targ v(out)
+val='vp/2' rise=1
.meas tran test_data1 trig AT = 1n targ v(out) val='vp/2' rise=3
.meas tran out_slew trig v(out) val='0.2*vp' rise=2 targ v(out)
+val='0.8*vp' rise=2
.meas tran delay_chk param='(inv_delay < 100ps) ? 1 : 0'
.meas tran skew when v(out)=0.6
.meas tran skew2 when v(out)=skew_meas
.meas tran skew3 when v(out)=skew_meas fall=2
.meas tran skew4 when v(out)=skew_meas fall=LAST
.meas tran skew5 FIND v(out) AT=2n
.meas tran v0_min  min i(v0) from='dfall' to='dfall+period'
.meas tran v0_avg avg i(v0) from='dfall' to='dfall+period'
.meas tran v0_integ integ i(v0) from='dfall' to='dfall+period'
.meas tran v0_rms rms i(v0) from='dfall' to='dfall+period'
.meas dc is_at FIND i(vs) AT=1
.meas dc is_max max i(vs) from=0 to=3.5
.meas dc vds_at when i(vs)=0.01
.meas ac vout_at FIND v(out) AT=1MEG
.meas ac vout_atd FIND vdb(out) AT=1MEG
.meas ac vout_max max v(out) from=1k to=10MEG
.meas ac freq_at when v(out)=0.1
.meas ac vout_diff trig v(out) val=0.1 rise=1 targ v(out) val=0.1 fall=1
.meas ac fixed_diff trig AT = 10k targ v(out) val=0.1 rise=1
.meas ac vout_avg  avg  v(out)  from=10k to=1MEG
.meas ac vout_integ integ v(out) from=20k to=500k
.meas ac freq_at2 when v(out)=0.1 fall=LAST
.meas ac bw_chk param='(vout_diff < 100k) ? 1 : 0'
.meas ac vout_rms rms v(out) from=10 to=1G
```

16.4   Batch Output

16.4.1   .SAVE: Name vector(s) to be saved in raw file

General form:

```
.save vector vector vector ...
```

Examples:

```
.save i(vin) input output
.save @m1[id]
```

The vectors listed on the .SAVE line are recorded in the rawfile for use later with ngspice or ngnutmeg (ngnutmeg is just the data-analysis half of ngspice, without the ability to simulate). The standard vector names are accepted. If no .SAVE line is given, then the default set of vectors are saved (node voltages and voltage source branch currents). If .SAVE lines are given, only those vectors specified are saved. For more discussion on internal device data, see Appendix, chapt. 30. See also the section on the interactive command interpreter for information on how to use the rawfile.

16.4.2 .PRINT Lines

General form:

```
. print  prtype  ov1  <ov2  ...  ov8>
```

Examples:

```
. print  tran  v(4)  i(vin)
. print  dc  v(2)  i(vsrc)  v(23,  17)
. print  ac  vm(4,  2)  vr(7)  vp(8,  3)
```

The `.print` line defines the contents of a tabular listing of one to eight output variables. `prtype` is the type of the analysis (DC, AC, TRAN, NOISE, or DISTO) for which the specified outputs are desired. The form for voltage or current output variables is the same as given in the previous section for the print command; Spice2 restricts the output variable to the following forms (though this restriction is not enforced by ngspice):

V(N1<,N2>)	specifies the voltage difference between nodes N1 and N2. If N2 (and the preceding comma) is omitted, ground (0) is assumed. See the print command in the previous section for more details. For compatibility with spice2, the following five additional values can be accessed for the ac analysis by replacing the "V" in V(N1,N2) with:
	<table><tr><td>VR</td><td>Real part</td></tr><tr><td>VI</td><td>Imaginary part</td></tr><tr><td>VM</td><td>Magnitude</td></tr><tr><td>VP</td><td>Phase</td></tr><tr><td>VDB</td><td>20log10(magnitude)</td></tr></table>
I(VXXXXXXX)	specifies the current flowing in the independent voltage source named VXXXXXXX. Positive current flows from the positive node, through the source, to the negative node. (Not yet implemented: For the ac analysis, the corresponding replacements for the letter I may be made in the same way as described for voltage outputs.)

Output variables for the noise and distortion analyses have a different general form from that of the other analyses. There is no limit on the number of `.print` lines for each type of analysis. The `par('expression')` option (16.4.6) allows to use algebraic expressions in the `.print` lines.

16.4.3 .PLOT Lines

General form:

```
. plot  pltype  ov1  <(plo1,  phi1)>  <ov2  <(plo2,  phi2)>  ...  ov8>
```

Examples:

```
. plot  dc  v(4)  v(5)  v(1)
. plot  tran  v(17,  5)  (2,  5)  i(vin)  v(17)  (1,  9)
. plot  ac  vm(5)  vm(31,  24)  vdb(5)  vp(5)
. plot  disto  hd2  hd3(R)  sim2
. plot  tran  v(5,  3)  v(4)  (0,  5)  v(7)  (0,  10)
```

The `.plot` line defines the contents of one plot of from one to eight output variables. `pltype` is the type of analysis (DC, AC, TRAN, NOISE, or DISTO) for which the specified outputs are desired. The syntax for the *ovi* is identical to that for the `.print` line and for the plot command in the interactive mode.

The overlap of two or more traces on any plot is indicated by the letter "X". When more than one output variable appears on the same plot, the first variable specified is printed as well as plotted. If a printout of all variables is desired, then a companion `.print` line should be included. There is no limit

on the number of `.plot` lines specified for each type of analysis. The `par('expression')` option (16.4.6) allows to use algebraic expressions in the `.plot` lines.

16.4.4   .FOUR: Fourier Analysis of Transient Analysis Output

General form:

```
. four  freq  ov1 <ov2 ov3 ...>
```

Examples:

```
. four  100K v(5)
```

The `.four` (or Fourier) line controls whether ngspice performs a Fourier analysis as a part of the transient analysis. `freq` is the fundamental frequency, and `ov1` is the desired vector to be analysed. The Fourier analysis is performed over the interval <TSTOP-period, TSTOP>, where TSTOP is the final time specified for the transient analysis, and period is one period of the fundamental frequency. The dc component and the first nine harmonics are determined. For maximum accuracy, TMAX (see the `.tran` line) should be set to period/100.0 (or less for very high-Q circuits). The `par('expression')` option (16.4.6) allows to use algebraic expressions in the `.four` lines.

16.4.5   .PROBE: Name vector(s) to be saved in raw file

General form:

```
. save  vector <vector  vector  ...>
```

Examples:

```
. probe  i(vin)  input  output
. probe  @m1[id]
```

Same as .SAVE (see chapt. 16.4.1).

16.4.6   par('expression'): Algebraic expressions for output

General form:

```
par('expression')
output=par('expression')   $ not in .measure
```

Examples:

```
. four  1001  sq1=par('v(1)*v(1)')
. measure  tran  vtest  find  par('(v(2)*v(1))') AT=2.3m
. print  tran  output=par('v(1)/v(2)')  v(1)  v(2)
. plot  dc  v(1)  diff=par('(v(4)−v(2))/0.01')  out222
```

In the output lines `.four`, `.plot`, `.print`, `.save` and in the `.measure` evaluation it is possible to add algebraic expression for output, in addition to vectors. All of these output lines accept `par('expression')`, where expression is any expression as has already been defined for the B source (see chapter 6.1). Thus `expression` may contain predefined functions, numerical values, constants, simulator output like v(n1) or i(vdb), parameters predefined by a .param statement, and the variables hertz, temper, and time.

Internally expression is replaced by an internally generated voltage node, which is the output of a B source, one node and B source per par('...'). Several par('...') are allowed in each line, up to 99 per input file. The internal nodes are named pa_00 to pa_99. If your input file already contains such node names, an error will occur, unless you rename these nodes.

In `.four`, `.plot`, `.print`, `.save`, but not in `.measure`, an alternative syntax `output=par('expression')` is possible. `par('expression')` may be used as described above. `output` is the name of the new node to replace the expression. So `output` has to be unique and a valid node name.

The syntax of `output=par('expression')` is strict, no spaces between par and (', or between (and ' are allowed, (' and ') both are required. Also there is not much error checking on your input, if there is a typo, for example, an error may pop up at an unexpected place.

Chapter 17

Starting ngspice

17.1 Introduction

Ngspice consists of a simulator and a front-end for data analysis and plotting. The front-end may be run as a separate "stand-alone" program under the name ngnutmeg. Ngnutmeg will read in the "raw" data output file created by ngspice -r or with the write command in an interactive ngspice session. Ngnutmeg or interactive ngspice can plot data from a simulation on a graphics terminal or a workstation display. Most of the commands available in the interactive Ngspice front end are available in nutmeg; where this is not the case, ngspice-only commands have been marked with an asterisk ("*").

Ngspice and ngnutmeg use the X Window System for plotting (see chapter 19.2) if they find the environment variable DISPLAY (OS LINUX, Cygwin, BCD ...). Otherwise, a terminal independent (non-graphical) interface is used. If you are using X on a workstation, the DISPLAY variable should already be set; if you want to display graphics on a system different from the one you are running ngspice or ngutmeg on, DISPLAY should be of the form "machine:0.0". See the appropriate documentation on the X Window System for more details.

The MS Windows vesrsions of ngspice and ngnutmeg will have a native gaphics interface (see chapter 19.1).

ngnutmeg is a subset of ngspice dedicated to data evaluation, still made available for historical reasons. Typically ngspice will give you access to all commands required for simulation and analysis.

17.2 Where to obtain ngspice

The actual distribution of ngspice may be downloaded from the ngspice download web page. The installation for LINUX or MS Windows is described in the file **INSTALL** to be found in the top level directory.

If you want to check out the source code which is actually under development, you may have a look at the ngspice source code repository, which is stored using the concurrent version system (CVS). The CVS repository may be browsed on the CVS web page, also useful for downloading individual files. You may however download (or check out) the complete source code tree from the console window (LINUX, CYGWIN or MSYS/MINGW) by issuing the command (in a single line)

```
cvs -z3 -d:pserver:anonymous@ngspice.cvs.sourceforge.net:/cvsroot/ngspice -lf
co -P ngspice/ng-spice-rework
```

You need to have CVS installed, which is available for all three OSs. The whole source tree is then available in <current directory>/ngspice/ng-spice-rework. Compilation and local installation is again described in **INSTALL**. If you later want to update your files and download the recent changes from the repository, you just type

```
cvs -z3 -d:pserver:anonymous@ngspice.cvs.sourceforge.net:/cvsroot/ngspice -lf update
-d -P
```

17.3   Command line options for starting ngspice and ngnutmeg

Command Synopsis:

```
ngspice [ −o logfile ] [ −r rawfile] [ −b ] [ −i ] [ input file ... ]
ngnutmeg [ − ] [ datafile ... ]
```

Options are:

Option	Long option	Meaning
-		Don't try to load the default data file ("rawspice.raw") if no other files are given (ngnutmeg only).
-n	−no-spiceinit	Don't try to source the file ".spiceinit" upon startup. Normally ngspice and ngnutmeg try to find the file in the current directory, and if it is not found then in the user's home directory (obsolete).
-t TERM	−terminal=TERM	The program is being run on a terminal with mfb name term (obsolete).
-b	−batch	Run in batch mode. Ngspice reads the default input source (e.g. keyboard) or reads the given input file and performs the analyses specified; output is either Spice2-like line-printer plots ("ascii plots") or a ngspice rawfile. See the following section for details. Note that if the input source is not a terminal (e.g. using the IO redirection notation of "<") ngspice defaults to batch mode (-i overrides). This option is valid for ngspice only.
-s	−server	Run in server mode. This is like batch mode, except that a temporary rawfile is used and then written to the standard output, preceded by a line with a single "@", after the sicmulation is done. This mode is used by the ngspice daemon. This option is valid for ngspice only. Example from the console window: `cat adder.cir|ngspice -s|more`
-i	−interactive	Run in interactive mode. This is useful if the standard input is not a terminal but interactive mode is desired. Command completion is not available unless the standard input is a terminal, however. This option is valid for ngspice only.
-r FILE	−rawfile=FILE	Use rawfile as the default file into which the results of the simulation are saved. This option is valid for ngspice only.
-p	−pipe	Allow a program (e.g., xcircuit) to act as a GUI frontend for ngspice through a pipe. Thus ngspice will assume that the pipe is a tty and allows to run in interactive mode.
-o FILE	−output=FILE	All logs generated during a batch run (-b) will be saved in outfile.
-h	−help	A short help statement of the command line syntax.
-v	−version	Prints a version information.

-a	−autorun	Start simulation immediately, as if a control section
		`.control`
		`run`
		`.endc`
		had been added to the input file.

Further arguments to ngspice are taken to be ngspice input files, which are read and saved (if running in batch mode then they are run immediately). Ngspice accepts Spice3 (and also most Spice2) input files, and outputs ascii plots, fourier analyses, and node printouts as specified in `.plot`, `.four`, and `.print` cards. If an out parameter is given on a `.width` card, the effect is the same as `set width = ...`. Since ngspice ascii plots do not use multiple ranges, however, if vectors together on a `.plot` card have different ranges they do not provide as much information as they do in a scalable graphics plot.

For ngnutmeg, further arguments are taken to be data files in binary or ascii format (see ngsconvert(1)) which are loaded into ngnutmeg. If the file is in binary format, it may be only partially completed (useful for examining output before the simulation is finished). One file may contain any number of data sets from different analyses.

17.4 Starting options

17.4.1 Batch mode

Let's take as an example the Four-Bit binary adder MOS circuit shown in chapter 21.6, stored in a file **adder-mos.cir**. You may start the simulation immediately by calling

    ngspice -b -r adder.raw -o adder.log adder-mos.cir

ngspice will start, simulate according to the .tran command and store the output data in a rawfile adder.raw. Comments, warnings and infos go to log file adder.log.

17.4.2 Interactive mode

If you call

    ngspice adder-mos.cir

ngspice will start, load the circuit file, parse the circuit and then wait for your input. You may then start the simulation by issuing the **run** command, and following completion you may analyse the data by any of the commands given in chapter 18.4.

17.4.3 Interactive mode with control file or control section

If you add the following control section to your input file adder-mos.cir, you may call

    ngspice adder-mos.cir

and see ngspice starting, simulating and then plotting immediately.

Control section:

```
*  ADDER − 4 BIT ALL−NAND−GATE BINARY ADDER
.control
set noaskquit
save vcc#branch
run
plot vcc#branch
rusage all
.endc
```

Any suitable command listed in chapter 18.4 may be added to the control section, as well as control structures described in chapter 18.5. Batch-like behaviour may be obtained by changing the control section to

Control section with batch-like behaviour:

```
*  ADDER − 4 BIT ALL−NAND−GATE BINARY ADDER
.control
set noaskquit
save vcc#branch
run
write adder.raw vcc#branch
quit
.endc
```

If you put this control section into a file, say adder-start.sp, you may just add the line

    .include adder-start.sp

to your input file adder-mos.cir to obtain the batch-like behaviour. In the following example the line **.tran ...** from the input file is overridden by the **tran** command given in the control section.

Control section overriding the .tran command:

```
*   ADDER − 4 BIT ALL–NAND–GATE BINARY ADDER
.control
set noaskquit
save vcc#branch
tran 1n 500n
plot vcc#branch
rusage all
.endc
```

17.5   Standard configuration file spinit

Upon start up ngspice reads its configuration file **spinit**. **spinit** may be found in C:\Spice\share\ngspice\scripts (Windows) or /usr/local/share/ngspice/scripts (LINUX). The path may be overriden by setting the environmental variable SPICE_LIB_DIR to a path where /scripts will be added. ngspice for Windows will also search for **spinit** in the directory where ngspice.exe resides. If **spinit** is not found, a warning message is issued, but ngspice will continue (but of course without code models etc.).

Standard spinit contents:

```
* Standard ngspice init file
alias exit quit
alias acct rusage all
set x11lineararcs
*set rndseed=12
*set filetype=ascii
*set ngdebug

*unset brief

strcmp __flag $program "ngspice"
if $__flag = 0

* For SPICE2 POLYs, edit the below line to point to the location
* of your codemodel.
 codemodel C:/Spice/lib/spice/spice2poly.cm

* The other codemodels
 codemodel C:/Spice/lib/spice/analog.cm
 codemodel C:/Spice/lib/spice/digital.cm
 codemodel C:/Spice/lib/spice/xtradev.cm
 codemodel C:/Spice/lib/spice/xtraevt.cm

end
unset __flag
```

spinit contains a script which is run upon startup of ngspice. You may find details of scripting in the next chapter. Aliases (name equivalences) are set. **set filetype=ascii** will yield ascii output in the output data file (rawfile), a more compact binary format is used otherwise. The asterisk '*' will comment out this line. If used by ngspice, spinit will then load the XSPICE code models from their absolute paths. You may also define relative paths here.   **set ngdebug** will yield a lot of additional debug output. Any other contents of the script. e.g. plotting preferences, may be included here and started automatically by ngspice. The compatibility mode of ngspice has to be set in spinit by **set ngbehavior=all**.

If the standard path for the libraries (see standard spinit above or **/usr/local/lib/spice** under CYGWIN and LINUX) is not adeaquate, you may add for eaxample the ./configure options **--prefix=/usr --libdir=/usr/lib64** to set the codemodel search path to **/usr/lib64/spice**. Besides the standard **lib** only **lib64** is aknowledged.

17.6 User defined configuration file .spiceinit

In addition to `spinit` you may define a file `.spiceinit` and put it into the current directory or in your home directory. This file will be read in and executed after `spinit`, but before any other input file is read. It may contain any script and override the commands given in `spinit`. If the command line option `-n` is used upon ngspice startup, this file will be ignored.

17.7 Environmental variables

17.7.1 Ngspice specific variables

SPICE_LIB_DIR default: /usr/local/share/ngspice (LINUX, CYGWIN), C:\Spice\share\ngspice
(Windows)

SPICE_EXEC_DIR default: /usr/local/bin (LINUX, CYGWIN), C:\Spice\bin (Windows)

SPICE_BUGADDR default: http://ngspice.sourceforge.net/bugrep.html
Where to send bug reports on ngspice.

SPICE_EDITOR default: vi (LINUX, CYGWIN), notepad.exe (MINGW, Visual Studio)
Set the editor called in the **edit** command. Always overrides the EDITOR env. variable.

SPICE_ASCIIRAWFILE default: 0
Format of the rawfile. 0 for binary, and 1 for ascii.

SPICE_NEWS default: $SPICE_LIB_DIR/news
A file which is copied verbatim to stdout when ngspice starts in interactive mode.

SPICE_HELP_DIR default: $SPICE_LIB_DIR/helpdir
Help directory, not used in Windows mode

SPICE_HOST default: empty string
Used in the **rspice** command (probably obsolete, to be documented)

SPICE_SCRIPTS default: $SPICE_LIB_DIR/scripts
In this directory the spinit file will be searched.

SPICE_PATH default: $SPICE_EXEC_DIR/ngspice
Used in the **aspice** command (probably obsolete, to be documented)

NGSPICE_MEAS_PRECISION default: 5
Sets the number of digits if output values are printed by the **meas(ure)** command.

SPICE_NO_DATASEG_CHECK default: undefined
If defined, will suppress memory resource info (probably obsolete, not used on Windows or where the /proc information system is available.)

17.7.2 Common environment variables

TERM LINES COLS DISPLAY HOME PATH EDITOR SHELL POSIXLY_CORRECT

17.8 Memory usage

Ngspice started with batch option (-b) and rawfile output (-r rawfile) will store all simulation data immediately into the rawfile without keeping them in memory. Thus very large circuits may be simulated, the memory requested upon ngspice startup will depend on the citcuit size, but will not increase during simulation.

If you start ngspice in interactive mode or 'interactive' with control section, all data will be kept in memory, to be available for later evaluation. A large circuit may outgrow even GBytes of memory. The same may happen after a very long simulation run with many vectors and many time steps to be stored. Issuing the `save <nodes>` command will help to reduce memory requirements by saving only the data defined by the command.

17.9   Simulation time

Simulating large circuits may take an considerable amount of cpu time. If this is of importance, you should compile ngspice with the flags for optimum speed, set during configuring ngspice compilation. Under LINUX, MINGW, and CYGWIN you should select the following option to disable the debug mode, which slows down ngspice:

    ./configure --disable-debug

Adding `--disable-debug` will set the -O2 optimisation flag for compiling and linking.

Under MS Visual Studio, you will have to select the **release** version which includes optimization for speed.

If you have selected XSPICE (see chapters 13 and II) as part of your compilation configuration (by adding the option `--enable-xspice` to your `./configure` command), the value of **trtol** (see 16.1.3) is set internally to 1 (insted of default 7) for higher precision if 'A' devices included in the circuit. This may double or even triple the cpu time needed for any transient simulation, because the amount of time steps and thus iteration steps is more than doubled. For MS Visual Studio compilation there is currently no simple way to exclude XSPICE during compilation.

Another way is to set the **option** `trtol=7` in your **.spiceinit** initialization file (via the option command 18.4.34) or in your circuit input file (via an .options line 16.1) to obtain standard spice3 tolerances and a speed gain of two. Beware however of convergence or precision issues if you use XSPICE 'A' devices.

If your circuit comprises mostly of MOS transistors, and you have a multi-core processor at hand, you may benefit from OpenMP parallel processing, as decribed next (17.10).

17.10   Ngspice on multi-core processors using OpenMP

17.10.1   Introduction

Todays computers typically come with CPUs having more than one core. It will thus be useful to enhance ngspice to make use of such multi-core processors.

Using circuits comprising mostly of transistors and e.g. the BSIM3 model, around $2/3$ of the cpu time is spent in evaluating the model equations (e.g. in the BSIM3Load() function). The same happens with other advanced transistor models. Thus this function should be parallized, if possible. Resulting from that the parrallel processing has to be within a dedicated device model. Interstingly solving the matrix takes only about 10% of the cpu time, so parallelizing the matrix solver is of secondary interest here!

A recent publication [1] has described a way to exactly do that using OpenMP, which is available on many platforms and is easy to use, especially if you want to parallelize processing of a for-loop.

I have chosen the BSIM3 version 3.3.0 model, located in the BSIM3 directory, as the first example. The BSIM3load() function in b3ld.c contains two nested for-loops using linked lists (models and instances, e.g. individual transistors). Unfortunately OpenMP requires a loop with an integer index. So in file B3set.c an array is defined, filled with pointers to all instances of BSIM3 and stored in model->BSIM3InstanceArray.

BSIM3load() is now a wrapper function, calling the for-loop, which runs through functions BSIM3LoadOMP(), once per instance. Inside BSIM3LoadOMP() the model equations are caculated.

Typically you now need to synchronize the activities, in that storing the results into the matrix has to be guarded. The trick offered by the authors now is that the storage is moved out of the BSIM3LoadOMP() function. Inside BSIM3LoadOMP() the updated data are stored in extra locations locally per instance, defined in bsim3def.h. Only after the complete for-loop is exercised, the update to the matrix is done in an extra function BSIM3LoadRhsMat() in the main thread after the parallelized loop. No extra synchronisation is required.

Then the thread programming needed is only a single line!!

    #pragma omp parallel for num_threads(nthreads) private(here)

introducing the for-loop.

This of course is made possible only thanks to the OpenMP guys and the clever trick on no synchronisation introduced by the above cited authors.

The time-measuring function `getrusage()` used with LINUX or Cygwin to determine the cpu time usage (with the `rusage` option enabled) counts tics from every core, adds them up, and thus reports a CPU time value enlarged by a factor of 8 if 8 threads have been chosen. So now ngspice is forced to use `ftime` for time measuring if OpenMP is selected.

Table 17.1: OpenMP performance

Threads	CPU time [s]	CPU time [s]
	Windows	LINUX
1 (standard)	167	165
1 (OpenMP)	174	167
2	110	110
3	95	94-120
4	83	107
6	94	90
8	93	91

17.10.2 Some results

Some results on an inverter chain with 627 CMOS inverters, running for 200ns, compiled with Visual Studio professional 2008 on Windows 7 (full optimization) or gcc 4.4, SUSE LINUX 11.2, -O2, on a i7 860 machine with four real cores (and 4 virtuals using hyperthreading):

So we see a ngspice speed up of nearly a factor of two! Even on an older notebook with dual core processor, I have got more than 1.5x improvement using two threads. Similar results are to be expected from BSIM4.

17.10.3 Usage

To say it clearly: OpenMP is installed inside the model equations of a particular model. So for the moment it is available only in **BSIM3 version 3.3.0**, not in version 3.2.4 nor in any other BSIM3 model, in **BSIM4 version 4.6.5**, not an any other BSIM4 model, and in B4SOI, version 4.3.1, not in any other SOI model. Older parameter files of version 4.6.x (x any number up to 5) are accepted, you have to check for compatibility.

Under **LINUX** you may run

```
./autogen.sh
./configure ...  --enable-openmp
make install
```

The same has been tested under MS Windows with **CYGWIN** and **MINGW** as well and delivers similar results.

Under **MS Windows** with **Visual Studio Professional** you have to place an additional preprocessor flag **USE_OMP**, and then enable /**openmp**. Visual Studio Express is not sufficient due to lack of OpenMP support. Even Visual Studio Professional lacks debugging support for OpenMP. There are local preprocessor flags (USE_OMP3 in bsim3def.h, USE_OMP4 in bsim4def.h, and USE_OMP4SOI in b4soidef.h) which you may modify individually if you want to switch off OpenMP in only one of the models BSIM3, BSIM4, or B4SOI.

The number of threads has to be set manually by placing

```
set num_threads=4
```

into spinit or .spiceinit. If OpnenMP is enabled, but num_threads not set, a default value `num_threads=2` is set internally.

If you run a circuit, please keep in mind to select BSIM3 (levels 8, 49) version 3.3.0 (12.2.9), by placing this version number into your parameter files), BSIM4 (levels 14, 54) version 4.6.5 (12.2.10), or B4SOI (levels 10, 58) version 4.3.1 (12.2.12).

If you run ./configure without `--enable-openmp` (or without USE_OMP preprocessor flag under MS Windows), you will get the standard, not parallelized BSIM3 and BSIM4 model, as has been available from Berkeley. If OpenMP is selected and the number of threads set to 1, there will be only a very slight cpu time disadvantage (typ. 3%) compared to the standard, non OpenMP build.

17.10.4 Literature

[1] R.K. Perng, T.-H. Weng, and K.-C. Li: "On Performance Enhancement of Circuit Simulation Using Multithreaded Techniques", IEEE International Conference on Computational Science and Engineering, 2009, pp. 158-165

17.11   Server mode option -s

A program may write the spice input to the console. This output is redirected to ngspice via '|'. ngspice called with the -p option writes its output to the console, which again is redirected to a receiving program by '|'. In the following simple example **cat** reads the input file and prints it content to the console, which is by a first pipe redirected to ngspice, which transfers its output to **more** via another pipe.

Example:

```
cat input.cir | ngspice −s | more
```

17.12   Ngspice control via input, output fifos

The following bash script (under LINUX)
  - launches ngspice in another thread.
  - writes some commands in ngspice input
  - reads the output and prints them on the console.
Example:

```
#!/usr/bin/env bash

NGSPICE_COMMAND="ngspice"

rm input.fifo
rm output.fifo

mkfifo input.fifo
mkfifo output.fifo

$NGSPICE_COMMAND −p −i <input.fifo >output.fifo &

exec 3>input.fifo
echo "I can write to input.fifo"

echo "Start processing..."
echo ""

echo "source circuit.cir" >&3
echo "set noaskquit" >&3
echo "set nobreak" >&3
echo "tran 0.01ms 0.1ms">&3
echo "print n0" >&3
echo "quit" >&3

echo "Try to open output.fifo ..."
exec 4<output.fifo
echo "I can read from output.fifo"

echo "Ready to read..."
while read output
do
        echo $output
done <&4

exec 3>&−
exec 4>&−

echo "End processing"
```

The input file for **spice** is:

Circuit.cir:

```
* Circuit.cir
V1 n0 0 SIN(0 10 1kHz)
C1 n1 n0 3.3nF
R1 0 n1 1k
.end
```

17.13 Reporting bugs and errors

Ngspice is a complex piece of software. The source code contains over 1500 files. Various models and simulation procedures are provided, some of them not used and tested intensively. Therefore errors may be found, some still eveloving from the original spice3f5 code, others introduced during the ongoing code enhancements.

If you happen to experience an error during the usage of ngspice, please send a report to the development team. Ngspice is hosted on sourceforge, the preferred place to post a bug report is the ngspice bug tracker. We would prefer to have your bug tested against the actual source code available at CVS, but of course a report using the most recent ngspice release is welcome! Please provide the following information with your report:

Ngspice version
Operating system
Small input file to reproduce the bug
Actual output versus the expected output

Chapter 18

Interactive Interpreter

18.1 Expressions, Functions, and Constants

Ngspice and ngnutmeg data is in the form of vectors: time, voltage, etc. Each vector has a type, and vectors can be operated on and combined algebraically in ways consistent with their types. Vectors are normally created when a data file is read in (see the load command below), and when the initial datafile is loaded. They can also be created with the let command.

An expression is an algebraic formula involving vectors and scalars (a scalar is a vector of length 1) and the following operations:

+ − * / ^ % ,

% is the modulo operator, and the comma operator has two meanings: if it is present in the argument list of a user definable function, it serves to separate the arguments. Otherwise, the term x , y is synonymous with x + j(y). Also available are the logical operations & (and), | (or), ! (not), and the relational operations <, >, >=, <=, =, and <> (not equal). If used in an algebraic expression they work like they would in C, producing values of 0 or 1. The relational operators have the following synonyms:

Operator	Synonym
gt	>
lt	<
ge	>=
le	<=
ne	<>
and	&
or	|
not	!
eq	=

These are useful when < and > might be confused with IO redirection (which is almost always).

The following functions are available:

Name	Function
mag(vector)	Magnitude of vector (same as abs(vector)).
ph(vector)	Phase of vector.
j(vector)	i(sqrt(-1)) times vector.
real(vector	The real component of vector.
imag(vector)	The imaginary part of vector.
db(vector)	20 log10(mag(vector)).
log(vector)	The logarithm (base 10) of vector.
ln(vector)	The natural logarithm (base e) of vector.
exp(vector)	e to the vector power.
abs(vector)	The absolute value of vector (same as mag).
sqrt(vector)	The square root of vector.
sin(vector)	The sine of vector.
cos(vector)	The cosine of vector.
tan(vector)	The tangent of vector.
atan(vector)	The inverse tangent of vector.
norm(vector)	The vector normalized to 1 (i.e, the largest magnitude of any component is 1).
rnd(vector)	A vector with each component a random integer between 0 and the absolute value of the vectors's corresponding component.
mean(vector)	The result is a scalar (a length 1 vector) that is the mean of the elements of vector.
avg(vector)	The average of a vector.
group_delay(vector)	Calculates the group delay -dphase[rad]/dω[rad/s]. Input is the complex vector of a system transfer function versus frequency, resembling damping and phase per frequency value. Output is a vector of group delay values (real values of delay times) versus frequency.
vector(number)	The result is a vector of length number, with elements 0, 1, ... number - 1. If number is a vector then just the first element is taken, and if it isn't an integer then the floor of the magnitude is used.
unitvec(number)	The result is a vector of length number, all elements having a value 1.
length(vector)	The length of vector.
interpolate(plot.vector)	The result of interpolating the named vector onto the scale of the current plot. This function uses the variable polydegree to determine the degree of interpolation.
deriv(vector)	Calculates the derivative of the given vector. This uses numeric differentiation by interpolating a polynomial and may not produce satisfactory results (particularly with iterated differentiation). The implementation only calculates the derivative with respect to the real component of that vector's scale.
vecd(vector)	Compute the differential of a vector.
vecmin(vector)	Returns the value of the vector element with maximum value.
vecmax(vector)	Returns the value of the vector element with minimum value.
sgauss(vector)	Returns a vector of random numbers drawn from a Gaussian distribution (real value, mean = 0 , standard deviation = 1). The length of the vector returned is determined by the input vector. The contents of the input vector will not be used. A call to sgauss(0) will return a single value of a random number as a vector of length 1..
sunif(vector)	Returns a vector of random real numbers uniform distributed in the interval [-1 .. 1[. The length of the vector returned is determined by the input vector. The contents of the input vector will not be used. A call to sunif(0) will return a single value of a random number as a vector of length 1.

A vector may be either the name of a vector already defined or a floating-point number (a scalar). A number may be written in any format acceptable to ngspice, such as 14.6Meg or -1.231e-4. Note that you can either use scientific notation or one of the abbreviations like MEG or G, but not both. As with ngspice, a number may have trailing alphabetic characters after it.

The notation expr [num] denotes the num'th element of expr. For multi-dimensional vectors, a vector of one less dimension is returned. Also for multi-dimensional vectors, the notation expr[m][n] will return the nth element of the mth subvector. To get a subrange of a vector, use the form expr[lower, upper]. To reference vectors in a plot that is not the current plot (see the setplot command, below), the notation plotname.vecname can be used. Either a plotname or a vector name may be the wildcard all. If the plotname is all, matching vectors from all plots are specified, and if the vector name is all, all vectors in the specified plots are referenced. Note that you may not use binary operations on expressions involving wildcards - it is not obvious what all + all should denote, for instance. Thus some (contrived) examples of expressions are:

Expressions example:

```
cos (TIME) + db(v(3))
sin (cos (log ([1  2  3  4  5  6  7  8  9  10])))
TIME * rnd(v(9)) − 15 * cos (vin#branch) ^ [7.9 e5  8]
not ((ac3 .FREQ[32] & tran1 .TIME[10]) gt 3)
```

Vector names in ngspice may have a name such as @name[param], where name is either the name of a device instance or model. This denotes the value of the param parameter of the device or model. See Appendix, chapt. 30 for details of what parameters are available. The value is a vector of length 1. This function is also available with the show command, and is available with variables for convenience for command scripts. There are a number of pre-defined constants in nutmeg. They are:

Constant	Name	Value
π	J	3.14159...
e	e (the base of natural logarithms)	2.71828...
c	c (the speed of light)	299,792,500 $^m/_{sec}$
i	i (the square root of -1)	
kelvin	kelvin (Absolute zero in Centigrade)	-273.15$°C$
q	echarge (The charge of an electron)	1.6021918e-19 C
K	boltz (The Boltzman's constant)	1.3806226e-23$^J/_K$
h	planck (The Planck's constant)	6.626200e-34

These are all in MKS units. If you have another variable with a name that conflicts with one of these then it takes precedence.

18.2 Plots

The output vectors of any analysis are stored in plots, a traditional SPICE notion. A plot is a group of vectors. A first `tran` command will generate several vectors within a plot tran1. A subsequent `tran` command will store their vectors in tran2. Then a `linearize` command will linearize all vectors and store them in tran3, which then becomes the current plot. A `fft` will generate a plot spec1, again now the current plot. The `display` command always will show all vectors in the current plot. `Setplot` followed by Return will show all plots. `Setplot name` will change the current plot to 'name' (e.g. `setplot tran2` will make tran2 the current plot). A sequence `name.vector` may be used to access the vector from a foreign plot.

You may generate plots by yourself: `setplot new` will generate a new plot named unknown1, `set curplottitle="a new plot"` will set a title, set `curplotname=myplot` will set its name as a short description, set `curplotdate="Sat Aug 28 10:49:42 2010"` will set its date. Note that strings with spaces have to be given with double quotes.

Of course the notion 'plot' will be used by this manual also in its more common meaning, denoting a graphics plot or being a `plot` command. Be careful to get the correct meaning.

18.3 Command Interpretation

If a word is typed as a command, and there is no built-in command with that name, the directories in the `sourcepath` list are searched in order for the file. If it is found, it is read in as a command file (as

if it were sourced). Before it is read, however, the variables `argc` and `argv` are set to the number of words following the filename on the command line, and a list of those words respectively. After the file is finished, these variables are unset. Note that if a command file calls another, it must save its `argv` and `argc` since they are altered. Also, command files may not be re-entrant since there are no local variables. (Of course, the procedures may explicitly manipulate a stack...) This way one can write scripts analogous to shell scripts for ngnutmeg and ngspice.

Note that for the script to work with ngspice, it must begin with a blank line (or whatever else, since it is thrown away) and then a line with `.control` on it. This is an unfortunate result of the source command being used for both circuit input and command file execution. Note also that this allows the user to merely type the name of a circuit file as a command and it is automatically run. The commands are executed immediately, without running any analyses that may be specified in the circuit (to execute the analyses before the script executes, include a "`run`" command in the script).

There are various command scripts installed in `/usr/local/lib/spice/scripts` (or whatever the path is on your machine), and the default `sourcepath` includes this directory, so you can use these command files (almost) like builtin commands.

18.4   Commands

Commands marked with a * are only available in ngspice, not in ngnutmeg.

18.4.1   Ac*: Perform an AC, small-signal frequency response analysis

General Form:

ac (DEC | OCT | LIN) N Fstart Fstop

Do an small signal ac analysis (see also chapter 16.3.1) over the specified frequency range.
`DEC` decade variation, and `N` is the number of points per decade.
`OCT` stands for octave variation, and `N` is the number of points per octave.
`LIN` stands for linear variation, and `N` is the number of points.
`fstart` is the starting frequency, and `fstop` is the final frequency.
Note that in order for this analysis to be meaningful, at least one independent source must have been specified with an ac value.

In this ac analysis all non-linear devices are linearized around their actual dc operating point. All Ls and Cs get their imaginary value, depending on the actual frequency step. Each output vector will be calculated relative to the input voltage (current) given by the ac value (Iin equals to 1 in the example below). The resulting node voltages (and branch currents) are complex vectors. Therefore you have to be careful using the plot command.
Example:

```
* AC test
Iin 1 0 AC 1
R1 1 2 100
L1 2 0 1

.control
AC LIN 101 10 10K
plot v(2)          $ real part !
plot mag(v(2))     $ magnitude
plot db(v(2))      $ same as vdb(2)
plot imag(v(2))
plot real(v(2))
plot phase(v(2))   $ phase in rad
plot 180/PI*phase(v(2)) $ phase in deg
.endc
.end
```

In addition to the plot examples given above you may use the variants of vxx(node) described in chapter 16.4.2 like `vdb(2)`.

18.4.2   Alias: Create an alias for a command

General Form:

```
alias [word] [text ...]
```

Causes word to be aliased to text. History substitutions may be used, as in C-shell aliases.

18.4.3   Alter*: Change a device or model parameter

Alter changes the value for a device or a specified parameter of a device or model.

General Form:

```
alter dev = <expression>
alter dev param = <expression>
alter @dev[param] = <expression>
```

<expression> must be real (complex isn't handled right now, integer is fine though, but no strings. For booleans, use 0/1.

Old style (pre 3f4):

```
alter device value
alter device parameter value [ parameter value ]
```

Using the old style, its first form is used by simple devices which have one principal value (resistors, capacitors, etc.) where the second form is for more complex devices (bjt's, etc.). Model parameters can be changed with the second form if the name contains a "#". For specifying vectors as values, start the vector with "[", followed by the values in the vector, and end with "]". Be sure to place a space between each of the values and before and after the "[" and "]".

Some examples are given below:

Examples (Spice3f4 style):

```
alter vd = 0.1
alter vg dc = 0.6
alter @m1[w]= 15e−06
alter  @vg[sin] [ −1 1.5 2MEG ]
alter @Vi[pwl] = [ 0 1.2 100p 0 ]
```


18.4.4   Altermod*: Change a model parameter

General form:

```
altermod mod = <expression>
altermod mod param = <expression>
altermod @mod[param] = <expression>
```

Example:

```
altermod m1 vth0 = 0.7
```

Altermod is a version of the alter (see 18.4.3) command which operates on models and is used in the same manner.

18.4.5   Asciiplot: Plot values using old-style character plots

General Form:

```
asciiplot plotargs
```

Produce a line printer plot of the vectors. The plot is sent to the standard output, so you can put it into a file with asciiplot args ... > file. The set options width, height, and nobreak determine the width and height of the plot, and whether there are page breaks, respectively. Note that you will have problems if you try to asciiplot something with an X-scale that isn't monotonic (i.e, something like sin(TIME)), because asciiplot uses a simple-minded linear interpolation. The asciiplot command doesn't deal with log scales or the delta keywords.

18.4.6   Aspice*: Asynchronous ngspice run

General Form:

```
aspice input−file [output−file]
```

Start an ngspice run, and when it is finished load the resulting data. The raw data is kept in a temporary file. If output-file is specified then the diagnostic output is directed into that file, otherwise it is thrown away.

18.4.7   Bug: Mail a bug report

General Form:

```
bug
```

Send a bug report. Please include a short summary of the problem, the version number and name of the operating system that you are running, the version of ngspice that you are running, and the relevant ngspice input file. (If you have defined BUGADDR, the mail is delivered to there.)

18.4.8   Cd: Change directory

General Form:

```
cd [directory]
```

Change the current working directory to directory, or to the user's home directory if none is given.

18.4.9   Compose: Compose a vector

General Form:

```
compose name values value1 [ value2 ... ]
compose name parm = val [ parm = val ... ]
```

The first form takes the values and creates a new vector, the values may be arbitrary expressions.
The second form has the following possible parameters:

start The value at which the vector should start.

stop The value at which the vector should end.

step The difference between sucessive elements.

lin The number of points, linearly spaced..

18.4.10   Destroy: Delete a data set

General Form:

```
destroy [plotnames | all]
```

Release the memory holding the data for the specified runs.

18.4.11   Dc*: Perform a DC-sweep analysis

General Form:

```
dc Source−Name Vstart Vstop Vincr [ Source2 Vstart2 Vstop2 Vincr2 ]
```

Do a dc transfer curve analysis. See the previous sections (chapter 16.3.2) for more details.

18.4.12 Define: Define a function

General Form:

```
define function(arg1, arg2, ...) expression
```

Define the user-definable function with the name function and arguments arg1, arg2, ... to be expression, which may involve the arguments. When the function is later used, the arguments it is given are substituted for the formal arguments when it is parsed. If expression is not present, any definition for function is printed, and if there are no arguments to define then all currently active definitions are printed. Note that you may have different functions defined with the same name but different arities. Some useful definitions are:

Example:

```
define max(x,y)  (x > y) * x + (x <= y) * y
define min(x,y)  (x < y) * x + (x >= y) * y
```

18.4.13 Deftype: Define a new type for a vector or plot

General Form: defines

```
deftype [v | p] typename abbrev
```

types for vectors and plots. abbrev will be used to parse things like abbrev(name) and to label axes with M<abbrev>, instead of numbers. It may be ommitted. Also, the command "deftype p plottype pattern ..." will assign plottype as the name to any plot with one of the patterns in its Name: field.

Example:

```
deftype v capacitance F
settype capacitance moscap
plot moscap vs v(cc)
```

18.4.14 Delete*: Remove a trace or breakpoint

General Form:

```
delete [ debug-number ... ]
```

Delete the specified breakpoints and traces. The debug numbers are those shown by the status command (unless you do status > file, in which case the debug numbers are not printed).

18.4.15 Diff: Compare vectors

General Form:

```
diff plot1 plot2 [vec ...]
```

Compare all the vectors in the specified plots, or only the named vectors if any are given. There are different vectors in the two plots, or any values in the vectors differ significantly the difference is reported. The variable diff_abstol, diff_reltol, and diff_vntol are used to determine a significant difference.

18.4.16 Display: List known vectors and types

General Form:

```
display [varname ...]
```

Prints a summary of currently defined vectors, or of the names specified. The vectors are sorted by name unless the variable nosort is set. The information given is the name of the vector, the length, the type of the vector, and whether it is real or complex data. Additionally, one vector is labelled [scale]. When a command such as plot is given without a vs 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. If you undefine the scale (i.e, let TIME = []), one of the remaining vectors becomes the new scale (which is undetermined). You may set the scale to another vector of the plot with the command setscale (18.4.50).

18.4.17   Echo: Print text

General Form:

```
echo [ text ...]
```

Echos the given text to the screen.

18.4.18   Edit*: Edit the current circuit

General Form:

```
edit [ file ]
```

Print the current ngspice input file into a file, call up the editor on that file and allow the user to modify it, and then read it back in, replacing the original file. If a filename is given, then edit that file and load it, making the circuit the current one. The editor may be defined in **spinit** by a command line like

```
set editor=C:/programs/notepad++.exe
```

18.4.19   FFT: fast Fourier transform of vectors

General Form:

```
fft vector1 [vector2] ...
```

This analysis provides a fast Fourier transform of the input vector(s). fft is much faster than spec (18.4.57) (about a factor of 50 to 100 for larger vectors) !

The fft command will create a new plot consisting of the Fourier transforms of the vectors given on the command line. Each vector given should be a transient analysis result, i.e. it should have 'time' as a scale. You will have got these vectors by the `tran Tstep Tstop Tstart` command.

The vector should have a linear equidistant time scale. Therefore linearization using the `linearize` command is recommended before running fft. Be careful selecting a Tstep value small enough for good interpolation, e.g. much smaller than any signal period to be resolved by `fft` (see `linearize` command). The Fast Fourier Transform will be computed using a window function as given with the specwindow variable. Its code is based on the FFT function provided at http://local.wasp.uwa.edu.au/~pbourke/other/dft/, downloaded April 5th, 2008. A new plot named specx will be generated with a new vector (having the same name as the input vector, see command above) containing the transformed data.

How to compute the fft from a transient simulation output:

```
ngspice 8 -> setplot tran1
ngspice 9 -> linearize V(2)
ngspice 9 -> set specwindow=blackman
ngspice 10 -> fft V(2)
ngspice 11 -> plot mag(V(2))
```

Linearize will create a new vector V(2) in a new plot **tran2**. The command `fft V(2)` will create a new plot **spec1** with vector V(2) holding the resulting data.

The variables listed in the following table control operation of the `fft` command. Each can be set with the set command before calling `fft`.

specwindow: This variable is set to one of the following strings, which will determine the type of windowing used for the Fourier transform in the spec command. If not set, the default is "hanning".

none No windowing

rectangular Rectangular window

bartlet Bartlett (also triangle) window

blackman Blackman window

hanning Hanning (also hann or cosine) window

hamming Hamming window

gaussian Gaussian window

flattop Flat top window
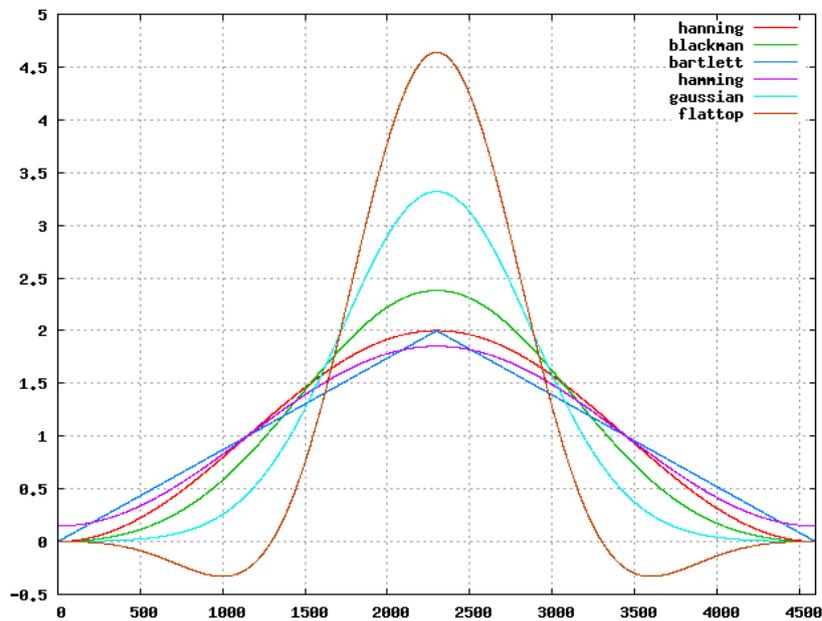
Figure 18.1: Spec and FFT window functions (Gaussian order = 4)

specwindoworder: This can be set to an integer in the range 2-8. This sets the order when the Gaussian window is used in the spec command. If not set, order 2 is used.

18.4.20 Fourier: Perform a fourier transform

General Form:

```
fourier fundamental_frequency [value ...]
```

Does a fourier analysis of each of the given values, using the first 10 multiples of the fundamental frequency (or the first nfreqs, if that variable is set - see below). The output is like that of the `.four` ngspice line (chapter 16.4.4). The values may be any valid expression. The values are interpolated onto a fixed-space grid with the number of points given by the fourgridsize variable, or 200 if it is not set. The interpolation is of degree polydegree if that variable is set, or 1. If polydegree is 0, then no interpolation is done. This is likely to give erroneous results if the time scale is not monotonic, though.

18.4.21 Gnuplot: Graphics output via Gnuplot

General Form:

```
gnuplot file plotargs
```

Like plot, but using gnuplot for graphics output and further data manipulation. ngspice creates a file called `file.plt` containing the gnuplot command sequence, a file called `file.data` containing the data to be plotted, and a file called `file.eps` containg a postscript hardcopy of the plot. On LINUX gnuplot is called via xterm, which offers a gnuplot console to manipulate the data. On Windows a gnuplot command console window is opened as well as the plot window. Of course you have to have gnuplot installed properly on your system. This option will work with Gnuplot version 4.2.6, but unfortunately not with version 4.4 (as of March 2010).

18.4.22 Hardcopy: Save a plot to a file for printing

General Form:

```
hardcopy file plotargs
```

Just like plot, except that it creates a file called `file` containing the plot. The file is a postscript image. As an alternative the plot(5) format is available by setting the hcopydevtype variable to `plot5`, and can be printed by either the plot(1) program or lpr with the -g flag.

18.4.23 Help: Print summaries of Ngspice commands

Prints help. This help information, however, is spice3f5-like, stemming from 1991 and thus is outdated. If the argument `all` is given, a short description of everything you could possibly type is printed. If commands are given, descriptions of those commands are printed. Otherwise help for only a few major commands is printed. On Windows this **help** command is no longer available. Spice3f5 compatible help may be found at http://newton.ex.ac.uk/teaching/CDHW/Electronics2/userguide/. For ngspice please use this manual.

18.4.24 History: Review previous commands

General Form:

```
history [number]
```

Print out the history, or the last `number` commands typed at the keyboard.

18.4.25 Iplot*: Incremental plot

General Form:

```
iplot [ node ...]
```

Incrementally plot the values of the nodes while ngspice runs. The iplot command can be used with the where command to find trouble spots in a transient simulation.

18.4.26 Jobs*: List active asynchronous ngspice runs

General Form:

```
jobs
```

Report on the asynchronous ngspice jobs currently running. Ngnutmeg checks to see if the jobs are finished every time you execute a command. If it is done then the data is loaded and becomes available.

18.4.27 Let: Assign a value to a vector

General Form:

```
let name = expr
```

Creates a new vector called name with the value specified by expr, an expression as described above. If expr is [] (a zero-length vector) then the vector becomes undefined. Individual elements of a vector may be modified by appending a subscript to name (ex. name[0]). If there are no arguments, let is the same as display.
    The command **let** creates a vector in the current plot, use **setplot** (18.4.49) to create a new plot.
    See also **unlet** (18.4.69), **compose** (18.4.9).

18.4.28 Linearize*: Interpolate to a linear scale

General Form:

```
linearize vec ...
```

Create a new plot with all of the vectors in the current plot, or only those mentioned as arguments to the command, all data linearized onto an equidistant time scale.

How compute the fft from a transient simulation output:

```
ngspice  8  –>  setplot  tran1
ngspice  9  –>  linearize  V(2)
ngspice  9  –>  set  specwindow=blackman
ngspice  10  –>  fft  V(2)
ngspice  11  –>  plot  mag(V(2)) tstep
```

Linearize will create new vectors **vec** or renew all vectors of the current plot if no arguments are given. The new vectors are interpolated onto a linear time scale, which is determined by the values of `tstep`, `tstart`, and `tstop` in the currently active transient analysis. The currently loaded input file must include a transient analysis (a tran command may be run interactively before the last reset, alternately), and the current plot must be from this transient analysis. The length of the new vector is `(tstop - tstart) / tstep + 1.5`. This command is needed for example if you want to do a fft analysis (18.4.19). Please note that the parameter `tstep` of your transient analysis (see chapter 16.3.9) has to be small enough to get adequate resolution, otherwise the command `linearize` will do sub-sampling of your signal.

18.4.29   Listing*: Print a listing of the current circuit

General Form:

```
listing  [logical]  [physical]  [deck]  [expand]  [param]
```

If the **logical** argument is given, the listing is with all continuation lines collapsed into one line, and if the **physical** argument is given the lines are printed out as they were found in the file. The default is logical. A **deck** listing is just like the physical listing, except without the line numbers it recreates the input file verbatim (except that it does not preserve case). If the word **expand** is present, the circuit is printed with all subcircuits expanded. The option **param** allows to print all parameters and their actual values.

18.4.30   Load: Load rawfile data

General Form:

```
load  [filename]  ...
```

Loads either binary or ascii format rawfile data from the files named. The default filename is rawspice.raw, or the argument to the -r flag if there was one.

18.4.31   Meas*: Mesurements on simulation data

General Form (example):

```
MEAS {DC|AC|TRAN} result  TRIG trig_variable VAL=val <TD=td>
<CROSS=# | CROSS=LAST> <RISE=#|RISE=LAST> <FALL=#|FALL=LAST>
<TRIG AT=time> TARG targ_variable VAL=val <TD=td> <CROSS=# | CROSS=LAST>
<RISE=#|RISE=LAST> <FALL=#|FALL=LAST> <TRIG AT=time>
```

All of the input forms found in 16.3.10 may be used here with the command **meas** instead of `.meas(ure)`. Using meas inside the .control ... .endc section offers additional features compared to the .meas use. meas will print the results as usual, but in addition will store its measurement result (typically the token `result` given in the command line) in a vector. This vector may be used in following command lines of the script as an input value of another command. For details of the command see chapt. 16.3.10.

18.4.32   Noise*: Noise analysis

See the .NOISE analysis (16.3.4) for details.

The `noise` command will generate two plots (typically named noise1 and noise2) with Noise Spectral Density Curves and Integrated Noise data. To write these data into output file(s), you may use the following command sequence:

Command sequence for writing noise data to file(s):

```
.control
tran 1e-6 1e-3
write test_tran.raw
noise V(out) vinp dec 333 1 1e8 16
print inoise_total onoise_total
*first option to get all of the output (two files)
setplot noise1
write test_noise1.raw all
setplot noise2
write test_noise2.raw all
* second option (all in one raw-file)
write testall.raw noise1.all noise2.all
.endc
```

18.4.33   Op*: Perform an operating point analysis

General Form:

```
op
```

Do an operating point analysis. See chapter 16.3.5 for more details.

18.4.34   Option*: Set a ngspice option

General Form:

```
option [option=val] [option=val] ...
```

Set any of the simulator variables as listed in chapt. 16.1. See this chapter also for more information on the available options. The `option` command without any argument lists the actual options set in the simulator (to be verified). Multiple options may be set in a single line.

The following example demonstates a control section, which may be added to your circuit file to test the influence of variable trtol on the number of iterations and on the simulation time.

Command sequence for testing option trtol:

```
.control
set noinit

option trtol=1
echo
echo trtol=1
run
rusage traniter trantime
reset
option trtol=3
echo
echo trtol=3
run
rusage traniter trantime
reset
option trtol=5
echo
echo trtol=5
run
rusage traniter trantime
reset
option trtol=7
echo
echo trtol=7
run
rusage traniter trantime
plot tran1.v(out25) tran1.v(out50) v(out25)  v(out50)
.endc
```

18.4.35   Plot: Plot values on the display

General Form:

```
plot exprs [ylimit ylo yhi] [xlimit xlo xhi] [xindices xilo xihi]
[xcompress comp] [xdelta xdel] [ydelta ydel] [xlog] [ylog] [loglog]
[vs xname] [xlabel word] [ylabel word] [title word] [samep]
[linear]
```

Plot the given vectors or exprs on the screen (if you are on a graphics terminal). The xlimit and ylimit arguments determine the high and low x- and y-limits of the axes, respectively. The xindices arguments determine what range of points are to be plotted - everything between the xilo'th point and the xihi'th point is plotted. The xcompress argument specifies that only one out of every comp points should be plotted. If an xdelta or a ydelta parameter is present, it specifies the spacing between grid lines on the X- and Y-axis. These parameter names may be abbreviated to xl, yl, xind, xcomp, xdel, and ydel respectively.

The xname argument is an expression to use as the scale on the x-axis. If xlog or ylog are present then the X or Y scale, respectively, is logarithmic (loglog is the same as specifying both). The xlabel and ylabel arguments cause the specified labels to be used for the X and Y axes, respectively.

If samep is given, the values of the other parameters (other than xname) from the previous plot, hardcopy, or asciiplot command is used unless re-defined on the command line.

The title argument is used in the place of the plot name at the bottom of the graph.

The linear keyword is used to override a default logscale plot (as in the output for an AC analysis).

Finally, the keyword `polar` generates a polar plot. To produce a smith plot, use the keyword `smith`. Note that the data is transformed, so for smith plots you will see the data transformed by the function $(x-1)/(x+1)$. To produce a polar plot with a smith grid but without performing the smith transform, use the keyword `smithgrid`.

If you specify `plot all`, all vectors (including the scale vector) are plotted versus the scale vector (see commands display (18.4.16) or setscale (18.4.50) on viweing the vectors of the current plot). The

command `plot ally` will not plot the scale vector, but all other 'real' y values. The command `plot alli` will yield all current vectors, the command `plot allv` all voltage vectors.

18.4.36   Print: Print values

General Form:

```
print [ col ] [ line ] expr  ...
```

Prints the vector(s) described by the expression expr. If the col argument is present, print the vectors named side by side. If line is given, the vectors are printed horizontally. col is the default, unless all the vectors named have a length of one, in which case line is the default. The options width, length, and nobreak are effective for this command (see asciiplot). If the expression is all, all of the vectors available are printed. Thus print col all > file prints everything in the file in SPICE2 format. The scale vector (time, frequency) is always in the first column unless the variable noprintscale is true. You may use the vectors alli, allv, ally with the print command, but then the scale vector will not be printed.
   Examples:

```
set  width=300
print  all

set  length=500
```

18.4.37   Quit: Leave Ngspice or Nutmeg

General Form:

```
quit
```

Quit ngnutmeg or ngspice. Ngspice will ask for an aknowledgement if parameters have not been saved. If **'set noaskquit'** is specified, ngspice will terminate immediately.

18.4.38   Rehash: Reset internal hash tables

General Form:

```
rehash
```

Recalculate the internal hash tables used when looking up UNIX commands, and make all UNIX commands in the user's PATH available for command completion. This is useless unless you have set unixcom first (see above).

18.4.39   Reset*: Reset an analysis

General Form:

```
reset
```

Throw out any intermediate data in the circuit (e.g, after a breakpoint or after one or more analyses have been done already), and re-parse the input file. The circuit can then be re-run from it's initial state, overriding the affect of any set or alter commands. In Spice-3e and earlier versions this was done automatically by the run command.

18.4.40   Reshape: Alter the dimensionality or dimensions of a vector

General Form:

```
reshape vector vector ...
or
reshape vector vector ... [ dimension , dimension , ... ]
or
reshape vector vector ... [ dimension ][ dimension ] ...
```

This command changes the dimensions of a vector or a set of vectors. The final dimension may be left off and it will be filled in automatically. If no dimensions are specified, then the dimensions of the first vector are copied to the other vectors. An error message of the form 'dimensions of x were inconsistent' can be ignored.

18.4.41 Resume*: Continue a simulation after a stop

General Form:

```
resume
```

Resume a simulation after a stop or interruption (control-C).

18.4.42 Rspice*: Remote ngspice submission

General Form:

```
rspice input file
```

Runs a ngspice remotely taking the input file as a ngspice input file, or the current circuit if no argument is given. Ngnutmeg or ngspice waits for the job to complete, and passes output from the remote job to the user's standard output. When the job is finished the data is loaded in as with aspice. If the variable rhost is set, ngnutmeg connects to this host instead of the default remote ngspice server machine. This command uses the "rsh" command and thereby requires authentication via a ".rhosts" file or other equivalent method. Note that "rsh" refers to the "remote shell" program, which may be "remsh" on your system; to override the default name of "rsh", set the variable remote_shell. If the variable rprogram is set, then rspice uses this as the pathname to the program to run on the remote system.

Note: rspice will not acknowledge elements that have been changed via the "alter" or "altermod" commands.

18.4.43 Run*: Run analysis from the input file

General Form:

```
run [ rawfile ]
```

Run the simulation as specified in the input file. If there were any of the control lines .ac, .op, .tran, or .dc, they are executed. The output is put in **rawfile** if it was given, in addition to being available interactively.

18.4.44 Rusage: Resource usage

General Form:

```
rusage [ resource ... ]
```

Print resource usage statistics. If any resources are given, just print the usage of that resource. Most resources require that a circuit be loaded. Currently valid resources are:

elapsed The amount of time elapsed since the last rusage elapsed call.

faults Number of page faults and context switches (BSD only).

space Data space used.

time CPU time used so far.

temp Operating temperature.

tnom Temperature at which device parameters were measured.

equations Circuit Equations

time Total Analysis Time

totiter Total iterations

accept Accepted timepoints

rejected Rejected timepoints

loadtime Time spent loading the circuit matrix and RHS.

reordertime Matrix reordering time

lutime L-U decomposition time

solvetime Matrix solve time

trantime Transient analysis time

tranpoints Transient timepoints

traniter Transient iterations

trancuriters Transient iterations for the last time point*

tranlutime Transient L-U decomposition time

transolvetime Transient matrix solve time

everything All of the above.

* listed incorrectly as "Transient iterations per point".

18.4.45   Save*: Save a set of outputs

General Form:

```
save [ all | allv | alli | output ...]
```

Save a set of outputs, discarding the rest. Maybe used to dramatically reduce memory (RAM) requirements if only a few useful nodes or branches are saved. If a node has been mentioned in a save command, it appears in the working plot after a run has completed, or in the rawfile if ngspice is run in batch mode. If a node is traced or plotted (see below) it is also saved. For backward compatibility, if there are no save commands given, all outputs are saved.

When the keyword "all" or the keyword "allv", appears in the save command, all node voltages, voltage source currents and inductor currents are saved in addition to any other values listed. If the keyword "alli" appears in the save command, all device currents are saved.

Note: the current implementation saves only the currents of devices which have internal nodes, i.e. MOSFETs with non zero RD and RS; BJTs with non-zero RC, RB and RE; DIODEs with non-zero RS; etc. Resistor and capacitor currents are not saved with this option. These deficiencies will be addressed in a later revision.

Save voltage and current:

```
save vd_node vs#branch
```

Note: **save** will not accept vectors (in contrast to .**save**). Nodes or branches have to be specified for <output> ! In the .**control** .... .**endc** section **save** should occur before the **run** or **tran** command to become effective. **Save** allows to store and later access internal device parameters. e.g. in a command like

Save internal parameters:

```
save all @mn1[gm]
```

which saves all analysis output data plus **gm** of transistor mn1 to the internal memory (see also 18.1).

18.4.46   Sens*: Run a sensitivity analysis

General Form:

```
sens  output_variable
sens  output_variable  ac  (  DEC  |  OCT  |  LIN  )  N  Fstart  Fstop
```

Perform a Sensitivity analysis. `output_variable` is either a node voltage (ex. "v(1)" or "v(A,out)") or a current through a voltage source (ex. "i(vtest)"). The first form calculates DC sensitivities, the second form calculates AC sensitivies. The output values are in dimensions of change in output per unit change of input (as opposed to percent change in output or per percent change of input).

18.4.47   Set: Set the value of a variable

General Form:

```
set  [word]
set  [word = value]  ...
```

Set the value of word to be value, if it is present. You can set any word to be any value, numeric or string. If no value is given then the value is the boolean 'true'. If you enter a string containing spaces, you have to enclose it with double quotes.

The value of word may be inserted into a command by writing $word. If a variable is set to a list of values that are enclosed in parentheses (which must be separated from their values by white space), the value of the variable is the list.

The variables used by ngspice are listed in section 18.6.

`Set` entered without any parameter will list all variables set, and their values, if applicable.

18.4.48   Setcirc*: Change the current circuit

General Form:

```
setcirc  [circuit name]
```

The current circuit is the one that is used for the simulation commands below. When a circuit is loaded with the source command (see below, **??**-Read-a:) it becomes the current circuit.

18.4.49   Setplot: Switch the current set of vectors

General Form:

```
setplot  [plotname]
```

Set the current plot to the plot with the given name, or if no name is given, prompt the user with a menu. (Note that the plots are named as they are loaded, with names like tran1 or op2. These names are shown by the setplot and display commands and are used by diff, below.) If the "New plot" item is selected, the current plot becomes one with no vectors defined.

Note that here the word "plot" refers to a group of vectors that are the result of one ngspice run. When more than one file is loaded in, or more than one plot is present in one file, ngspice keeps them separate and only shows you the vectors in the current plot.

18.4.50   Setscale: Set the scale vector for the current plot

General Form:

```
setscale  [vector]
```

Defines the scale vector for the current plot. If no argument is given, the current scale vector is printed. The scale vector delivers the values for the x-axis in a 2D plot.

18.4.51   Settype: Set the type of a vector

General Form:

```
settype  type  vector  ...
```

Change the type of the named vectors to `type`. Type names can be found in the following table.

Type	Unit		Type	Unit
notype			pole	
time	s		zero	
frequency	Hz		s-param	
voltage	V		temp-sweep	Celsius
current	A		res-sweep	Ohms
onoise-spectrum	(V or A)^2/Hz		impedance	Ohms
onoise-integrated	V or A		admittance	Mhos
inoise-spectrum	(V or A)^2/Hz		power	W
inoise-integrated	V or A		phase	Degree
			decibel	dB

18.4.52   Shell: Call the command interpreter

General Form:

```
shell [ command ]
```

Call the operating system's command interpreter; execute the specified command or call for interactive use.

18.4.53   Shift: Alter a list variable

General Form:

```
shift [ varname ] [ number ]
```

If `varname` is the name of a list variable, it is shifted to the left by number elements (i.e, the number leftmost elements are removed). The default `varname` is `argv`, and the default number is 1.

18.4.54   Show*: List device state

General Form:

```
show devices [ : parameters ] , ...
```

The show command prints out tables summarizing the operating condition of selected devices (much like the spice2 operation point summary). If device is missing, a default set of devices are listed, if device is a single letter, devices of that type are listed; if device is a subcircuit name (beginning and ending in ":") only devices in that subcircuit are shown (end the name in a double-":" to get devices within sub-subcircuits recursively). The second and third forms may be combined ("letter:subcircuit:") or ("letter:subcircuit::") to select a specific type of device from a subcircuit. A device's full name may be specified to list only that device. Finally, devices may be selected by model by using the form "#modelname" or ":subcircuit#modelname" or "letter:subcircuit#modelname".

If no parameters are specified, the values for a standard set of parameters are listed. If the list of parameters contains a "+", the default set of parameters is listed along with any other specified parameters.

For both devices and parameters, the word "all" has the obvious meaning.

Note: there must be spaces separating the ":" that divides the device list from the parameter list.

18.4.55   Showmod*: List model parameter values

General Form:

```
showmod models [ : parameters ] , ...
```

The showmod command operates like the show command (above) but prints out model parameter values. The applicable forms for models are a single letter specifying the device type letter, "letter:subckt:", "modelname", ":subckt:modelname", or "letter:subcircuit:modelname".

18.4.56   Source: Read a ngspice input file

General Form:

```
source file
```

For ngspice: read the ngspice input file file. Ngnutmeg and ngspice commands may be included in the file, and must be enclosed between the lines `.control` and `.endc`. These commands are executed immediately after the circuit is loaded, so a control line of `ac ...` works the same as the corresponding .ac card. The first line in any input file is considered a title line and not parsed but kept as the name of the circuit. The exception to this rule is the file .spiceinit. Thus, a ngspice command script must begin with a blank line and then with a .control line. Also, any line starting with the characters *# is considered a control line.

For ngutmeg: reads commands from the file filename. Lines beginning with the character * are considered comments and are ignored.

18.4.57   Spec: Create a frequency domain plot

General Form:

```
spec start_freq stop_freq step_freq vector [vector ...]
```

Calculates a new complex vector containing the Fourier transform of the input vector (typically the linearized result of a transient analysis). The default behaviour is to use a Hanning window, but this can be changed by setting the variables specwindow and specwindoworder appropriately.

Typical usage:

```
ngspice 13 -> linearize
ngspice 14 -> set specwindow = "blackman"
ngspice 15 -> spec 10 1000000 1000 v(out)
ngspice 16 -> plot mag(v(out))
```

Possible values for specwindow are: none, hanning, cosine, rectangular, hamming, triangle, bartlet, blackman, gaussian and flattop. In the case of a gaussian window specwindoworder is a number specifying its order. For a list of window functions see 18.4.19.

18.4.58   Status*: Display breakpoint information

General Form:

```
status
```

Display all of the traces and breakpoints currently in effect.

18.4.59   Step*: Run a fixed number of timepoints

General Form:

```
step [number]
```

Iterate number times, or once, and then stop.

18.4.60   Stop*: Set a breakpoint

General Form:

```
stop [ after n] [ when value cond value ] ...
```

Set a breakpoint. The argument after `n` means stop after iteration number "n", and the argument `when value cond value` means stop when the first value is in the given relation with the second value, the possible relations being

Symbol	Alias	Meaning
=	eq	equal to
<>	ne	not equal
>	gt	greater than
<	lt	less than
>=	ge	greater than or equal to
<=	le	less than or equal to

IO redirection is disabled for the `stop` command, since the relational operations conflict with it (it doesn't produce any output anyway). The values above may be node names in the running circuit, or real values. If more than one condition is given, e.g. `stop after 4 when v(1) > 4 when v(2) < 2`, the conjunction of the conditions is implied.

18.4.61   Strcmp: Compare two strings

General Form:

```
strcmp _flag $string1 "string2"
```

The command compares two strings, either given by a variable (string1) or as a string in quotes ("string2"). _flag is set as an output variable to '0', if both strings are equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2; and a value less than zero indicates the opposite (like the C strcmp function).

18.4.62   Sysinfo*: Print system information

General Form:

```
sysinfo
```

The command prints system information useful for sending bug report to developers. Information consists of:

- Name of the operating system,

- CPU type,

- Number of physical processors (not available under Windows OS), number of logical processors,

- Total amount of DRAM available,

- DRAM currently available.

The example below shows the use of this command.

```
ngspice 1 -> sysinfo
OS: CYGWIN_NT-5.1  1.5.25(0.156/4/2)  2008-06-12  19:34
CPU: Intel(R) Pentium(R) 4 CPU 3.40GHz
Logical processors: 2
Total DRAM available = 1535.480469 MB.
DRAM currently available = 984.683594 MB.
ngspice 2 ->
```

> This command is tested under Windows OS and LINUX. It may not be available on your environment. If it is not available, please send analogous information when submitting bug reports.

18.4.63   Tf*: Run a Transfer Function analysis

General Form:

```
tf output_node input_source
```

The `tf` command performs a transfer function analysis, returning:

- the transfer function (output/input),

- output resistance,

- and input resistance

between the given output node and the given input source. The analysis assumes a small-signal DC (slowly varying) input. The following example file
Example input file:

```
* Tf test circuit
vs     1     0     dc 5
r1     1     2     100
r2     2     3     50
r3     3     0     150
r4     2     0     200

.control
tf v(3,5) vs
print all
.endc

.end
```

will yield the following output:
```
transfer_function = 3.750000e-001
output_impedance_at_v(3,5) = 6.662500e+001
vs#input_impedance = 2.000000e+002
```

18.4.64   Trace*: Trace nodes

General Form:

```
trace [ node ...]
```

For every step of an analysis, the value of the node is printed. Several traces may be active at once. Tracing is not applicable for all analyses. To remove a trace, use the delete command.

18.4.65   Tran*: Perform a transient analysis

General Form:

```
tran Tstep Tstop [ Tstart [ Tmax ] ] [ UIC ]
```

Perform a transient analysis. See chapter 16.3.9 of this manual for more details.

18.4.66   Transpose: Swap the elements in a multi-dimensional data set

General Form:

```
transpose vector vector ...
```

This command transposes a multidimensional vector. No analysis in ngspice produces multidimensional vectors, although the DC transfer curve may be run with two varying sources. You must use the "reshape" command to reform the one-dimensional vectors into two dimensional vectors. In addition, the default scale is incorrect for plotting. You must plot versus the vector corresponding to the second source, but you must also refer only to the first segment of this second source vector. For example (circuit to produce the transfer characteristic of a MOS transistor):
How to produce the transfer characteristic of a MOS transistor:

```
ngspice > dc vgg 0 5 1 vdd 0 5 1
ngspice > plot i(vdd)
ngspice > reshape all [6,6]
ngspice > transpose i(vdd) v(drain)
ngspice > plot i(vdd) vs v(drain)[0]
```

18.4.67   Unalias: Retract an alias

General Form:

unalias [word ...]

Removes any aliases present for the words.

18.4.68   Undefine: Retract a definition

General Form:

undefine function

Definitions for the named user-defined functions are deleted.

18.4.69   Unlet: Delete the specified vector(s)

General Form:

unlet vector [vector ...]

Delete the specified vector(s). See also **let** (18.4.27).

18.4.70   Unset: Clear a variable

General Form:

unset [word ...]

Clear the value of the specified variable(s) (word).

18.4.71   Version: Print the version of ngspice

General Form:

version [−s | −f | <version id>]

Print out the version of ngnutmeg that is running, if invoked without argument or with **-s** or **-f**. If the argument is a **<version id>** (any string different from **-s** or **-f** is considered a **<version id>**), the command checks to make sure that the arguments match the current version of ngspice. (This is mainly used as a `Command:` line in rawfiles.)

Options description:

- No option: The output of the command is the message you can see when running ngspice from the command line, no more no less.

- **-s**(hort): A shorter version of the message you see when calling ngspice from the command line.

- **-f**(ull): You may want to use this option if you want to know what extensions are included into the simulator and what compilation switches are active. A list of compilation options and included extensions is appended to the normal (not short) message. May be useful when sending bug reports.

The following example shows what the command returns is some situations:

Use of the version commad:

```
ngspice 10 -> version
******
** ngspice -22 : Circuit level simulation program
** The U. C. Berkeley CAD Group
** Copyright 1985-1994, Regents of the University of California.
** Please submit bug-reports to: ngspice-devel@@lists.sourceforge.net
** Creation Date: Sun Aug 24 00:35:57 CEST 2003
******
ngspice 11 -> version 14
Note: rawfile is version 14 (current version is 15)
ngspice 12 -> version 15
ngspice 13 ->
```

Note for developers: The option listing returned when version is called with the -f option is built at compile time using #ifdef blocks. When new compile switch are added, if you want them to appear on the list, you have to modify the code in misccoms.c.

18.4.72 Where*: Identify troublesome node or device

General Form:

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 you can examine the circuit and either correct the problem or make a bug report. You may do this either in the middle of a run or after the simulator has given up on the analysis. For transient simulation, the iplot command can be used to monitor the progress of the analysis. When the analysis slows down severly or hangs, interrupt the simulator (with control-C) and issue the where command. Note that only one node or device is printed; there may be problems with more than one node.

18.4.73 Wrdata: Write data to a file (simple table)

General Form:

wrdata [file] [vecs]

Writes out the vectors to file.

This is a very simple printout of data in array form. Column one is the default scale data, column two the simulated data. If more than one vector is given, the third column again is the default scale, the fourth the data of the second vector. The default format is ascii. All vectors have to stem from the same plot, otherwise a seg fault may occur. No further information is written to the file, so you have to keep track of your multiple outputs. The format may be changed in the near future.

output example from two vectors:

```
0.000000e+000    -1.845890e-006    0.000000e+000    0.000000e+000
7.629471e+006     4.243518e-006    7.629471e+006   -4.930171e-006
1.525894e+007    -5.794628e-006    1.525894e+007    4.769020e-006
2.288841e+007     5.086875e-006    2.288841e+007   -3.670687e-006
3.051788e+007    -3.683623e-006    3.051788e+007    1.754215e-006
3.814735e+007     1.330798e-006    3.814735e+007   -1.091843e-006
4.577682e+007    -3.804620e-007    4.577682e+007    2.274678e-006
5.340630e+007     9.047444e-007    5.340630e+007   -3.815083e-006
6.103577e+007    -2.792511e-006    6.103577e+007    4.766727e-006
6.866524e+007     5.657498e-006    6.866524e+007   -2.397679e-006

....
```

18.4.74   Write: Write data to a file (Spice3f5 format)

General Form:

write [file] [exprs]

Writes out the expressions to `file`.

First vectors are grouped together by plots, and written out as such (i.e, if the expression list contained three vectors from one plot and two from another, then two plots are written, one with three vectors and one with two). Additionally, if the scale for a vector isn't present, it is automatically written out as well.

The default format is binary, but this can be changed to ascii with the `set filetype` command. The default filename is `rawspice.raw`, or the argument to the `-r` flag on the command line, if there was one, and the default expression list is all.

18.4.75   Wrs2p: Write scattering parameters to file (Touchstone® format)

General Form:

wrs2p [file]

Writes out the s-parameters of a two-port to `file`.

In the active plot the following is required: vectors **frequency, S11 S12 S21 S22**, all having the same length and having complex values (as a result of an ac analysis), and vector **Rbase**. For details how to generate these data see chapt. 18.9.

The file format is Touchstone® Version 1, ascii, frequency in Hz, real and imaginary parts of **Snn** versus frequency.

The default filename is `s-param.s2p`.

output example:

```
!2-port S-parameter file
!Title: test for scattering parameters
!Generated by ngspice at Sat Oct 16 13:51:18  2010
# Hz S RI R 50
!freq            ReS11           ImS11           ReS21             ...
 2.500000e+006  -1.358762e-003  -1.726349e-002   9.966563e-001
 5.000000e+006  -5.439573e-003  -3.397117e-002   9.867253e-001
....
```

18.4.76   Xgraph: use the xgraph(1) program for plotting.

General Form:

xgraph file [exprs] [plot options]

The ngspice/ngnutmeg xgraph command plots data like the plot command but via `xgraph`, a popular X11 plotting program. If file is either "`temp`" or "`tmp`" a temporary file is used to hold the data while being plotted. For available plot options, see the `plot` command. All options except for polar or smith plots are supported.

18.5   Control Structures

18.5.1   While - End

General Form:

while condition
statement
...
end

While condition, an arbitrary algebraic expression, is true, execute the statements.

18.5.2 Repeat - End

General Form:

```
repeat [number]
statement
...
end
```

Execute the statements number times, or forever if no argument is given.

18.5.3 Dowhile - End

General Form:

```
dowhile condition
statement
...
end
```

The same as while, except that the condition is tested after the statements are executed.

18.5.4 Foreach - End

General Form:

```
foreach var value ...
statement
...
end
```

The statements are executed once for each of the values, each time with the variable `var` set to the current one. (`var` can be accessed by the `$var` notation - see below).

18.5.5 If - Then - Else

General Form:

```
if condition
statement
...
else
statement
...
end
```

If the condition is non-zero then the first set of statements are executed, otherwise the second set. The else and the second set of statements may be omitted.

18.5.6 Label

General Form:

```
label word
```

If a statement of the form goto word is encountered, control is transferred to this point, otherwise this is a no-op.

18.5.7 Goto

General Form:

```
goto word
```

If a statement of the form label word is present in the block or an enclosing block, control is transferred there. Note that if the label is at the top level, it must be before the goto statement (i.e, a forward goto may occur only within a block). A block to just include goto on the top level may look like

Example noop block to include forward goto on top level:

```
if (1)
...
goto gohere
...
label gohere
end
```

18.5.8   Continue

General Form:

```
continue
```

If there is a while, dowhile, or foreach block enclosing this statement, control passes to the test, or in the case of foreach, the next value is taken. Otherwise an error results.

18.5.9   Break

General Form:

```
break
```

If there is a while, dowhile, or foreach block enclosing this statement, control passes out of the block. Otherwise an error results.

Of course, control structures may be nested. When a block is entered and the input is the terminal, the prompt becomes a number of >'s corresponding to the number of blocks the user has entered. The current control structures may be examined with the debugging command cdump.

18.6   Variables

The operation of both ngutmeg and ngspice may be affected by setting variables with the "set" command. In addition to the variables mentioned below, the set command in ngspice also affect the behaviour of the simulator via the options previously described under the section on ".OPTIONS". You also may define new variables inside .control ... .endc for later use in your user-defined script (see chapter 18.7).

The following list is in alphabetical order. The predefined variables meaningful to ngspice (ngnutmeg) which may be altered by the set command are:

appendwrite Append to the file when a write command is issued, if one already exists.

brief If set to FALSE, the netlist will be printed.

colorN These variables determine the colors used, if X is being run on a color display. N may be between 0 and 15. Color 0 is the background, color 1 is the grid and text color, and colors 2 through 15 are used in order for vectors plotted. The value of the color variables should be names of colors, which may be found in the file /usr/lib/rgb.txt. ngspice for Windows does support only white background (color0=white with black grid and text) or or color0=black with white grid and text.

cpdebug Print control debugging information.

curplotdate Sets the date of the current plot.

curplotname Sets the name of the current plot.

curplottitle Sets the title (a short description) of the current plot.

debug If set then a lot of debugging information is printed.

device The name (`/dev/tty??`) of the graphics device. If this variable isn't set then the user's terminal is used. To do plotting on another monitor you probably have to set both the device and term variables. (If device is set to the name of a file, nutmeg dumps the graphics control codes into this file – this is useful for saving plots.)

diff_abstol The relative tolerance used by the **diff** command (default is 1e-12).

diff_reltol The relative tolerance used by the **diff** command (default is 0.001).

diff_vntol The absolute tolerance for voltage type vectors used by the **diff** command (default is 1e-6).

echo Print out each command before it is executed.

filetype This can be either `ascii` or `binary`, and determines what format are. The default is `ascii`.

fourgridsize How many points to use for interpolating into when doing fourier analysis.

gridsize If this variable is set to an integer, this number is used as the number of equally spaced points to use for the Y axis when plotting. Otherwise the current scale is used (which may not have equally spaced points). If the current scale isn't strictly monotonic, then this option has no effect.

gridstyle Sets the grid during plotting with the plot command. Will be overriden by direct entry of gridstyle in the plot command. A linear grid is standard for both x and y axis. Allowed values are `lingrid loglog xlog ylog smith smithgrid polar nogrid`.

hcopydev If this is set, when the hardcopy command is run the resulting file is automatically printed on the printer named hcopydev with the command `lpr -Phcopydev -g file`.

hcopyfont This variable specifies the font name for hardcopy output plots. The value is device dependent.

hcopyfontsize This is a scaling factor for the font used in hardcopy plots.

hcopydevtype This variable specifies the type of the printer output to use in the hardcopy command. If **hcopydevtype** is not set, Postscript format is assumed. `plot (5)` is recognized as an alternative output format. When used in conjunction with **hcopydev**, **hcopydevtype** should specify a format supported by the printer.

hcopyscale This is a scaling factor for the font used in hardcopy plots (between 0 and 10).

hcopywidth Sets width of the hardcopy plot.

hcopyheight Sets height of the hardcopy plot.

hcopypscolor Sets the color of the hardcopy output. If not set, black & white plotting is assumed with different linestyles for each output vector plotted. Setting to any valid color integer value yields a colored plot background (0: black 1: white, others see below) and colored solid lines. This is valid for postscript only.

hcopypstxcolor This variable sets the color of the text in the postscript hardcopy output. If not set, black is assumed on white background, white on black background. Valid colors are 0: black 1: white 2: red 3: blue 4: orange 5: green 6: pink 7: brown 8: khaki 9: plum 10: orchid 11: violet 12: maroon 13: turqoise 14: sienna 15: coral 16: cyan 17: magenta 18: gray for smith grid 19: gray for smith grid 20: gray for normal grid

height The length of the page for asciiplot and print col.

history The number of events to save in the history list.

lprplot5 This is a `printf(3s)` style format string used to specify the command to use for sending plot(5)-style plots to a printer or plotter. The first parameter supplied is the printer name, the second parameter supplied is a file name containing the plot. Both parameters are strings. It is trivial to cause ngspice to abort by supplying a unreasonable format string.

lprps This is a `printf(3s)` style format string used to specify the command to use for sending PostScript plots to a printer or plotter. The first parameter supplied is the printer name, the second parameter supplied is a file name containing the plot. Both parameters are strings. It is trivial to cause ngspice to abort by supplying a unreasonable format string.

nfreqs The number of frequencies to compute in the fourier command. (Defaults to 10.)

ngbehavior Sets the compatibility mode of ngspice. Its value 'all' has to be defined in spinit and will allow hspice and pspice compatibility. This value may be set as a standard in the future.

nobjthack BJTs can have either 3 or 4 nodes, which makes it difficult for the subcircuit expansion routines to decide what to rename. If the fourth parameter has been declared as a model name, then it is assumed that there are 3 nodes, otherwise it is considered a node. To disable this, you can set the variable "nobjthack" which forces BJTs to have 4 nodes (for the purposes of subcircuit expansion, at least).

nobreak Don't have asciiplot and print col break between pages.

noasciiplotvalue Don't print the first vector plotted to the left when doing an asciiplot.

noclobber Don't overwrite existing files when doing IO redirection.

noglob Don't expand the global characters '*', '?', '[', and ']'. This is the default.

nomoremode If nomoremode is not set, whenever a large amount of data is being printed to the screen (e.g, the print or asciiplot commands), the output is stopped every screenful and continues when a carriage return is typed. If nomoremode is set then data scrolls off the screen without check.

nonomatch If noglob is unset and a global expression cannot be matched, use the global characters literally instead of complaining.

nosort Don't have display sort the variable names.

noprintscale Don't print the scale in the leftmost column when a print col command is given.

numdgt The number of digits to print when printing tables of data (fourier, print col). The default precision is 6 digits. On the VAX, approximately 16 decimal digits are available using double precision, so `numdgt` should not be more than 16. If the number is negative, one fewer digit is printed to ensure constant widths in tables.

num_threads The number of of threads to be used if OpenMP (see chapt. 17.10) is selected. The default value is 2.

plotstyle This should be one of `linplot`, `combplot`, or `pointplot`. `linplot`, the default, causes points to be plotted as parts of connected lines. `combplot` causes a comb plot to be done. It plots vectors by drawing a vertical line from each point to the X-axis, as opposed to joining the points. `pointplot` causes each point to be plotted separately.

pointchars Set a string as a list of characters to be used as points in a point plot. Standard is ox*+#abcdefhgijklmnpqrstuvwyz. Characters §€ are not allowed.

polydegree The degree of the polynomial that the plot command should fit to the data. If polydegree is N, then nutmeg fits a degree N polynomial to every set of N points and draw 10 intermediate points in between each end point. If the points aren't monotonic, then it tries rotating the curve and reducing the degree until a fit is achieved.

polysteps The number of points to interpolate between every pair of points available when doing curve fitting. The default is 10.

program The name of the current program (argv[0]).

prompt The prompt, with the character '!' replaced by the current event number. Single quotes ' ' are required around the string entered!

rawfile The default name for rawfiles created.

remote_shell Overrides the name used for generating rspice runs (default is "rsh").

rndseed Seed value for random number generator (used by sgauss, sunif, and rnd functions). If not set, the process Id is used as seed value.

rhost The machine to use for remote ngspice runs, instead of the default one (see the description of the rspice command, below).

rprogram The name of the remote program to use in the rspice command.

sourcepath A list of the directories to search when a source command is given. The default is the current directory and the standard ngspice library (`/usr/local/lib/ngspice`, or whatever LIBPATH is #defined to in the ngspice source.

specwindow Windowing for commands `spec` (18.4.57) or `fft` (18.4.19). May be on of the following: `bartlet blackman cosine gaussian hamming hanning none rectangular triangle`.

specwindoworder Integer value 2 - 8 (default 2), used by commands `spec` or `fft`.

spicepath The program to use for the aspice command. The default is /cad/bin/spice.

term The mfb name of the current terminal.

ticmarks An integer value n, n tics (a small 'x') will be set on your graph. (Arrangement of the tics ?)

ticlist A list of integers, e.g. (4 14 24) to set tics (small 'x') on your graph.(Arrangement of the tics ?)

units If this is **degrees**, then all the trig functions will use degrees instead of radians.

unixcom If a command isn't defined, try to execute it as a UNIX command. Setting this option has the effect of giving a rehash command, below. This is useful for people who want to use ngnutmeg as a login shell.

wfont Set the font for the graphics plot in MS Windows. Typical fonts are `courier, times, arial` and all others found on your machine. Default is `courier`.

wfont_size The size of the windows font. Default is depending on systems settings, something like

width The width of the page for asciiplot and print col.

x11lineararcs Some X11 implementations have poor arc drawing. If you set this option, Ngspice will plot using an approximation to the curve using straight lines.

xbrushwidth Linewidth for grid, border and graph.

xfont Set the font for the graphics plot in X11 (LINUX, Cygwin, etc.). Input format has still to be checked.

There are several set variables that ngspice uses but ngutmeg does not. They are:

editor The editor to use for the edit command.

modelcard The name of the model card (normally `.MODEL`)

noaskquit Do not check to make sure that there are no circuits suspended and no plots unsaved. Normally ngspice warns the user when he tries to quit if this is the case.

nobjthack Assume that BJTs have 4 nodes.

noparse Don't attempt to parse input files when they are read in (useful for debugging). Of course, they cannot be run if they are not parsed.

nosubckt Don't expand subcircuits.

renumber Renumber input lines when an input file has .include's.

subend The card to end subcircuits (normally `.ends`).

subinvoke The prefix to invoke subcircuits (normally `X`).

substart The card to begin subcircuits (normally `.subckt`).

18.7   Scripts

Expressions, functions, constants, commands, variables, vectors, and control structures may be assembled into scripts within a .control ... .endc section of the input file. The script allows to automate a more complex ngspice behaviour: simulations are performed, output data are the analysed, simulations repeated with modified parameters, output vectors for plotting are assembled. The ngspice scripting language is not very powerful, but easily integrated into the simulation flow.

The ngspice input file for scripting contains the usual circuit netlist, modelcards, and a script, enclosed in the .control .. .endc section. ngspice is started in interactive mode with the input file in the command line (or sourced later with the **source** command). After reading the input file, the command sequence is immediately processed. Variables or vectors set by previous commands may be used in commands following their definition. data may be stored, plotted or grouped into new vectors for additional charts supporting data evaluation.

18.7.1   Variables

Variables are defined and initialized with the **set** command (18.4). **set output=10** will defined the variable **output** and set it to a (real) number 10. Predined variables, which are used inside ngspice for specific purposes, are listed in chapt. 18.6. Variables are accessible globally. The values of variables may be used in commands by writing **$varname** where the value of the variable is to appear, e.g. **$output**. The special variables **$$** and **$<** refer to the process ID of the program and a line of input which is read from the terminal when the variable is evaluated, respectively. If a variable has a name of the form **$&word**, then word is considered a vector (see below), and its value is taken to be the value of the variable. If **$foo** is a valid variable, and is of type list, then the expression **$foo[low-high]** represents a range of elements. Either the upper index or the lower may be left out, and the reverse of a list may be obtained with **$foo[len-0]**. Also, the notation **$?foo** evaluates to 1 if the variable foo is defined, 0 otherwise, and **$#foo** evaluates to the number of elements in foo if it is a list, 1 if it is a number or string, and 0 if it is a boolean variable.

18.7.2   Vectors

Ngspice and ngnutmeg data is in the form of vectors: time, voltage, etc. Each vector has a type, and vectors can be operated on and combined algebraically in ways consistent with their types. Vectors are normally created as a result of a transient or dc simulation. They are also established when a data file is read in (see the load command 18.4.30). They can also be created with the let command 18.4.27inside a script. If a variable has a name of the form $&word, then 'word' is considered a vector, and its value is taken to be the value of the variable.

18.7.3   Commands

Commands have been described in chapter 18.4.

18.7.4   control structures

Control structures have been described in chapter 18.5. Some simple examples will be given below.

Control structure examples:

```
Test sequences for ngspice control structures
*vectors are used (except foreach)
*start in interactive mode

.control

* test sequence for while, dowhile
  let loop = 0
  echo
  echo enter loop with  "$&loop"
  dowhile loop < 3
    echo within dowhile loop "$&loop"
    let loop = loop + 1
  end
  echo after dowhile loop "$&loop"
  echo
  let loop = 0
  while loop < 3
    echo within while loop "$&loop"
    let loop = loop + 1
  end
  echo after while loop "$&loop"
  let loop = 3
  echo
  echo enter loop with  "$&loop"
  dowhile loop < 3
    echo within dowhile loop "$&loop"       $ output expected
    let loop = loop + 1
  end
  echo after dowhile loop "$&loop"
  echo
  let loop = 3
  while loop < 3
    echo within while loop "$&loop"         $ no output expected
    let loop = loop + 1
  end
  echo after while loop "$&loop"
```

Control structure examples (continued):

```
* test for while, repeat, if, break
  let loop = 0
  while loop < 4
    let index = 0
    repeat
      let index = index + 1
      if index > 4
        break
      end
    end
    echo index "$&index"   loop "$&loop"
    let loop = loop + 1
  end

* test sequence for foreach
  echo
  foreach outvar 0 0.5 1 1.5
    echo parameters: $outvar   $ foreach parameters are variables, not vectors!
  end

* test for if ... else ... end
  echo
  let loop = 0
  let index = 1
  dowhile loop < 10
    let index = index * 2
    if index < 128
      echo "$&index" lt 128
    else
      echo "$&index"  ge 128
    end
    let loop = loop + 1
  end

* simple test for label, goto
  echo
  let loop = 0
  label starthere
  echo start "$&loop"
  let loop = loop + 1
  if loop < 3
    goto starthere
  end
  echo end "$&loop"

* test for label, nested goto
  echo
  let loop = 0
  label starthere1
  echo start nested "$&loop"
  let loop = loop + 1
  if loop < 3
    if loop < 3
      goto starthere1
    end
  end
  echo end "$&loop"
```

Control structure examples (continued):

```
* test for label, goto
  echo
  let index = 0
  label starthere2
  let loop = 0
  echo We are at start with index "$&index" and loop "$&loop"
  if index < 6
    label inhere
    let index = index + 1
    if loop < 3
      let loop = loop + 1
      if index > 1
        echo jump2
        goto starthere2
      end
    end
    echo jump
    goto inhere
  end
  echo We are at end with index "$&index" and loop "$&loop"

* test goto in while loop
  echo
  let loop = 0
  if 1     $ outer loop to allow nested forward label 'endlabel'
    while loop < 10
      if loop > 5
        echo jump
        goto endlabel
      end
      let loop = loop + 1
    end
    echo before  $ never reached
    label endlabel
    echo after "$&loop"
  end

*test for using variables
* simple test for label, goto
  echo
  set loop = 0
  label starthe
  echo start $loop
  let loop = $loop + 1  $ expression needs vector at lhs
  set loop = "$&loop"   $ convert vector contents to variable
  if $loop < 3
    goto starthe
  end
  echo end $loop
.endc
```

18.7.5   Example script 'spectrum'

A typical example script named **spectrum** is delivered with the ngspice distribution. Even if it is made obsolete by the internal **spec** command (see 18.4.57) and especially by the much faster **fft** command (see 18.4.19), it may act as a good example for getting aquainted with the ngspice (or nutmeg) post-processor language.

As a suitable input for spectrum you may run a ring-oscillator, delivered with ngspice in e.g. test/b-sim3soi/ring51_41.cir. For an adequate resolution you will need a simulation time of 1 µs. Then a small control script may start ngspice by loading the R.O. simulation data and start **spectrum**.

Small script to start ngspice, read the simulation data and start spectrum:

```
* test for script 'spectrum'
.control
load ring51_41.out
spectrum 10MEG 2500MEG 1MEG v(out25) v(out50)
.endc
```

18.7.6 Example script for random numbers

Generation and test of random numbers with Gaussian distribution

```
* agauss test in ngspice
* generate a sequence of gaussian distributed random numbers.
* test the distribution by sorting the numbers into a histogram (buckets)
.control
  define agauss(nom, avar, sig) (nom + avar/sig * sgauss(0))

  let mc_runs = 200
  let run = 0
  let no_buck = 8                          $ number of buckets
  let bucket = unitvec(no_buck)            $ each element contains 1
  let delta = 3e-11                        $ width of each bucket, depends on avar and sig
  let lolimit = 1e-09 - 3*delta
  let hilimit = 1e-09 + 3*delta

  dowhile run < mc_runs
    let val = agauss(1e-09, 1e-10, 3) $ get the random number
    if (val < lolimit)
        let bucket[0] = bucket[0] + 1 $ 'lowest' bucket
    end
    let part = 1
    dowhile part < (no_buck - 1)
      if ((val < (lolimit + part*delta)) & (val > (lolimit + (part-1)*delta)))
        let bucket[part] = bucket[part] + 1
                break
      end
      let part = part + 1
    end
    if (val > hilimit)
        let bucket[no_buck - 1] = bucket[no_buck - 1] + 1
$ 'highest' bucket
    end
    let run = run + 1
  end

  let part = 0
  dowhile part < no_buck
    let value = bucket[part] - 1
    set value = "$&value"
    echo $value                                $ print the buckets' contents
    let part = part + 1
  end

.endc

.end
```

18.8 Monte-Carlo Simulation

The ngspice scripting language may be used to run Monte-Carlo simulations with statistically varying device or model parameters. Ngspice contains a powerful random number generator running in the background. Its seed value is derived from the process id upon startup of ngspice. If you need reprodicible random numbers, you may start your script with a call to `set rndseed=<int value>`. Calls to the functions sgauss(0) or sunif(0) (see 18.1) will return Gaussian or uniform distributed random numbers (real numbers), stored in a vector. You may define (see 18.4.12) your own function using sgauss or sunif,

e.g. to change the mean or range. In a loop (see 18.5) then you may call the alter (18.4.3) or altermod (18.4.4) statements with random parameters followed by an analysis like op, dc, ac, tran or other.

18.8.1   Example 1

The first examples is a LC band pass filter, where L and C device parameters will be changed 100 times. Each change is followed by an ac analysis. All graphs of output voltage versus frequency are plotted. The file is available in the distribuition as /examples/Monte_Carlo/MonteCarlo.sp as well as from the CVS repository.

Monte-Carlo example 1

```
Perform  Monte  Carlo  simulation  in  ngspice
V1  N001  0  AC  1  DC  0
R1  N002  N001  141
*
C1  OUT  0  1e−09
L1  OUT  0  10e−06
C2  N002  0  1e−09
L2  N002  0  10e−06
L3  N003  N002  40e−06
C3  OUT  N003  250e−12
*
R2  0  OUT  141
*
.control
   let  mc_runs  =  100
   let  run  =  1
   set  curplot  =  new         $  create  a  new  plot
   set  scratch  =  $curplot   $  store  its  name  to  'scratch'
*
   define  unif(nom,  var)  (nom  +  nom∗var  ∗  sunif(0))
   define  aunif(nom,  avar)  (nom  +  avar  ∗  sunif(0))
   define  gauss(nom,  var,  sig)  (nom  +  nom∗var/sig  ∗  sgauss(0))
   define  agauss(nom,  avar,  sig)  (nom  +  avar/sig  ∗  sgauss(0))
*
   dowhile  run  <=  mc_runs
*      alter  c1  =  unif(1e−09,  0.1)
*      alter  l1  =  aunif(10e−06,  2e−06)
*      alter  c2  =  aunif(1e−09,  100e−12)
*      alter  l2  =  unif(10e−06,  0.2)
*      alter  l3  =  aunif(40e−06,  8e−06)
*      alter  c3  =  unif(250e−12,  0.15)
     alter  c1  =  gauss(1e−09,  0.1,  3)
     alter  l1  =  agauss(10e−06,  2e−06,  3)
     alter  c2  =  agauss(1e−09,  100e−12,  3)
     alter  l2  =  gauss(10e−06,  0.2,  3)
     alter  l3  =  agauss(40e−06,  8e−06,  3)
     alter  c3  =  gauss(250e−12,  0.15,  3)
     ac  oct  100  250K  10Meg
     set  run  ="$&run"       $  create  a  variable  from  the  vector
     set  dt  =  $curplot      $  store  the  current  plot  to  dt
     setplot  $scratch       $  make  'scratch'  the  active  plot
     let  vout{$run}={$dt}.v(out)  $  store  the  output  vector  to  plot  'scratch'
     setplot  $dt            $  go  back  to  the  previous  plot
     let  run  =  run  +  1
   end
   plot  db({$scratch}.all)
.endc

.end
```

18.8.2   Example 2

A more sophisticated input file for Monte Carlo simulation is distributed with the file /examples/-Monte_Carlo/MCring.sp (or CVS repository). Due to its length it is not reproduced here, but some comments on its enhancements over example 1 (18.8.1) are presented in the following.

A 25-stage ring oscillator is the circuit used with a transient simulation. It comprises of CMOS inverters, modelled with BSIM3. Several model parameters (vth, u0, tox, L, and W) shall be varied

statistically between each simulation run. The frequency of oscillation will be measured by a fft and stored. Finally a histogram of all measured frequencies will be plotted.

The function calls to `sunif(0)` and `sgauss(0)` return uniformly or Gaussian distributed random numbers. A function `unif`, defined by the line

    define unif(nom, var) (nom + (nom*var) * sunif(0))

will return a value with mean `nom` and deviation `var` relative to `nom`.

The line

    set n1vth0=@n1[vth0]

will store the threshold voltage vth0, given by the model parameter set, into a variable `n1vth0`, ready to be used by `unif`, `aunif`, `gauss`, or `agauss` function calls.

In the simulation loop the altermod command changes the model parameters before a call to tran. After the transient simulation the resulting vector is linearized, a fft is calculated, and the maximum of the fft signal is measured by the `meas` command and stored in a vector maxffts. Finally the contents of the vector maxffts is plotted in a histogram.

For more details, please have a look at the strongly commented input file MCring.sp.

18.9   Scattering parameters (s-parameters)

18.9.1   Intro

A command line script, available from the ngspice distribution at examples/control_structs/s-param.cir, together with the command wrs2p (see chapt. 18.4.75) allows to calculate, print and plot the scattering parameters S11, S21, S12, and S22 of any two port circuit at varying frequencies.

The printed output using wrs2p is a **Touchstone® version 1** format file. The file follows the format according to The Toustone File Format Specification, Version 2.0, available from here. An example is given as number 13 on page 15 of that specification.

18.9.2   S-parameter measurement basics

S-parameters allow a two-port description not just by permutating I_1, U_1, I_2, U_2, but using a superposition, leading to a power view of the port (We only look at two-ports here, because multi-ports are not (yet?) implemented.).

You may start with the effective power, being negative or positive

$$P = u \cdot i \tag{18.1}$$

The value of P may be the difference of two real numbers, with K being another real number.

$$ui = P = a^2 - b^2 = (a+b)(a-b) = (a+b)(KK^{-1})(a-b) = \{K(a+b)\}\{K^{-1}(a-b)\} \tag{18.2}$$

Thus you get

$$K^{-1}u = a + b \tag{18.3}$$

$$Ki = a - b \tag{18.4}$$

and finally

$$a = \frac{u + K^2 i}{2K} \tag{18.5}$$

$$b = \frac{u - K^2 i}{2K} \tag{18.6}$$

Bei introducing the reference resistance $Z_0 := K^2 > 0$ we get finally the Heaviside transformation

$$a = \frac{u + Z_0 i}{2\sqrt{Z_0}}, \quad b = \frac{u - Z_0 i}{2\sqrt{Z_0}} \tag{18.7}$$

In case of our two-port we subject our variables to a Heaviside transformation

$$a_1 = \frac{U_1 + Z_0 I_1}{2\sqrt{Z_0}} \quad b_1 = \frac{U_1 - Z_0 I_1}{2\sqrt{Z_0}} \tag{18.8}$$

$$a_2 = \frac{U_2 + Z_0 I_2}{2\sqrt{Z_0}} \quad b_2 = \frac{U_2 - Z_0 I_2}{2\sqrt{Z_0}} \tag{18.9}$$

The s-matrix for a two-port then is

$$\left(\begin{array}{c} b_1 \\ b_2 \end{array} \right) = \left(\begin{array}{cc} s_{11} & s_{12} \\ s_{21} & s_{22} \end{array} \right) \left(\begin{array}{c} a_1 \\ a_2 \end{array} \right) \tag{18.10}$$

Two obtain s_{11} we have to set $a_2 = 0$. This is accomplished by loading the output port exactly with the reference resistance Z_0, which sinks a current $I_2 = -U_2/Z_0$ from the port.

$$s_{11} = \left(\frac{b_1}{a_1} \right)_{a_2=0} \tag{18.11}$$

$$s_{11} = \frac{U_1 - Z_0 I_1}{U_1 + Z_0 I_1} \tag{18.12}$$

Loading the input port from an ac source U_0 via a resistor with resistance value Z_0, we obtain the relation

$$U_0 = Z_0 I_1 + U_1 \tag{18.13}$$

Entering this into 18.12, we get

$$s_{11} = \frac{2U_1 - U_0}{U_0} \tag{18.14}$$

For s_{21} we obtain similarily

$$s_{21} = \left(\frac{b_2}{a_1} \right)_{a_2=0} \tag{18.15}$$

$$s_{21} = \frac{U_2 - Z_0 I_2}{U_1 + Z_0 I_1} = \frac{2U_2}{U_0} \tag{18.16}$$

Equations 18.14 and 18.16 now tell us how to measure s_{11} and s_{21}: Measure U_1 at the input port, multiply by 2 using an E source, subtracting U_0 which for simplicity is set to 1, and divide by U_0. At the same time measure U_2 at the output port, multiply by 2 and divide by U_0. Biasing and measuring is done by subcircuit S_PARAM. To obtain s_{22} and s_{12}, you have to exchange the input and output ports of your two-port and do the same measurement again. This is achieved by switching resistors from low ($1m\Omega$) to high ($1T\Omega$) and thus switching the input and output ports.

18.9.3 Usage

Copy and then edit s-param.cir. You will find ths file in directory /examples/control_structs of the ngspice distribution.

The reference resistance (often called characteristic impedance) for the measurements is added as a parameter

`.param Rbase=50`

The bias voltages at the input and output ports of the circuit are set as parameters as well:

`.param Vbias_in=1 Vbias_out=2`

Place your circuit at the appropriate place in the input file, e.g. replacing the existing example circuits. The input port of your circuit has two nodes **in, 0**. The output port has the two nodes **out, 0**. The bias voltages are connected to your circuit via the resistances of value **Rbase** at the input and output respectively. This may be of importance for the operating point calculations if your circuit draws a large dc current.

Now edit the ac commands (see 18.4.1) according to the circuit provided, e.g.

`ac lin 100 2.5MEG 250MEG $ use for Tschebyschef`

Be careful to keep both ac lines in the .control ... .endc section the same and only change both in equal measure!

Select the plot commands (lin/log, or smith grid) or the 'write to file' commands (write, wrdata, or wrs2p) according to your needs.

Run ngspice in interactive mode

```
ngspice s-param.cir
```

18.10   MISCELLANEOUS (old stuff, has to be checked for relevance)

Ngnutmeg occasionally checks to see if it is getting close to running out of space, and warns the user if this is the case. (This is more likely to be useful with the ngspice front end.)

C-shell type quoting with " and ", and backquote substitution may be used. Within single quotes, no further substitution (like history substitution) is done, and within double quotes, the words are kept together but further substitution is done. Any text between backquotes is replaced by the result of executing the text as a command to the shell.

Tenex-style ('set filec' in the 4.3 C-shell) command, filename, and keyword completion is possible: If EOF (control-D) is typed after the first character on the line, a list of the commands or possible arguments is printed (If it is alone on the line it exits nutmeg). If escape is typed, then nutmeg tries to complete what the user has already typed. To get a list of all commands, the user should type <space> ^D.

History substitutions, similar to C-shell history substitutions, are also available - see the C-shell manual page for all of the details. The characters ~, @{, and @} have the same effects as they do in the C-Shell, i.e., home directory and alternative expansion. It is possible to use the wildcard characters *, ?, [, and] also, but only if you unset noglob first. This makes them rather useless for typing algebraic expressions, so you should set noglob again after you are done with wildcard expansion. Note that the pattern [^abc] matchs all characters except a, b, and c. IO redirection is available - the symbols >, >>, >&, >>&, and < have the same effects as in the C-shell.

You may type multiple commands on one line, separated by semicolons.

If X is being used, the cursor may be positioned at any point on the screen when the window is up and characters typed at the keyboard are added to the window at that point. The window may then be sent to a printer using the xpr(1) program.

18.11   Bugs (old stuff, has to be checked for relevance)

When defining aliases like alias pdb plot db('!:1' - '!:2') you must be careful to quote the argument list substitutions in this manner. If you quote the whole argument it might not work properly.

In a user-defined function, the arguments cannot be part of a name that uses the plot.vec syntax. For example: define check(v(1)) cos(tran1.v(1)) does not work.

If you type plot all all, or otherwise use a wildcard reference for one plot twice in a command, the effect is unpredictable.

The @@name[param] notation might not work with trace, iplot, etc. yet.

Chapter 19

Graphical User Interfaces

19.1 MS Windows

If compiled properly (e.g. using the –with-windows flag for ./configure under MINGW), ngspice for Windows offers a simple graphical user interface. In fact this interface does not offer much more for data input than a console would offer, e.g. command line inputs, command history and program text output. First of all it applies the Windows api for data plotting. If you run the sample input file given below, you will get an output as shown in fig. 16.1.

Input file:

```
*****Single NMOS Transistor For BSIM3V3.1  general purpose check (Id-Vd) ***
*
*** circuit description ***
m1 2 1 3 0 n1 L=0.6u W=10.0u
vgs 1 0 3.5
vds 2 0 3.5
vss 3 0 0
*
.dc vds 0 3.5 0.05 vgs 0 3.5 0.5
*
.control
run
plot vss#branch
.endc
*
* UCB parameters BSIM3v3.2
.include ../Exam_BSIM3/Modelcards/modelcard.nmos
.include ../Exam_BSIM3/Modelcards/modelcard.pmos
*
.end
```

The GUI consists of an I/O port (lower window) and a graphics window, created by the `plot` command.
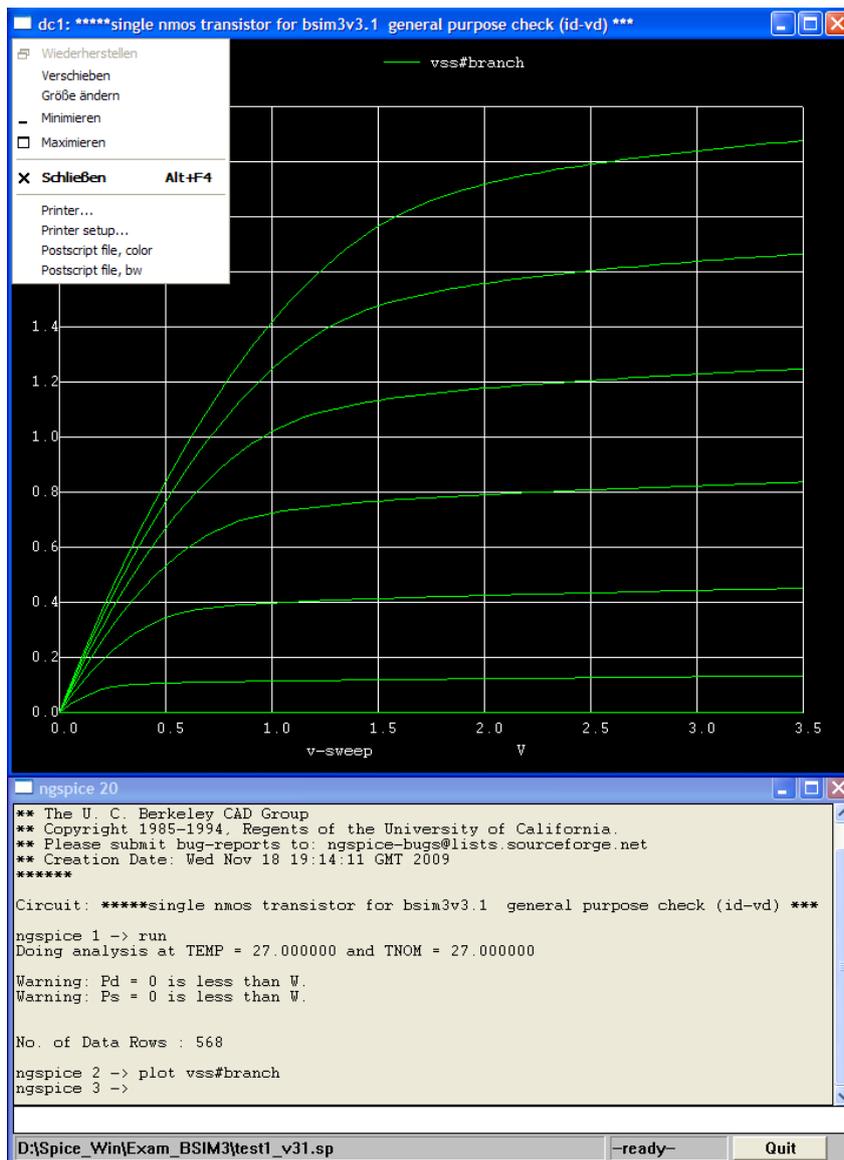
Figure 19.1: MS Windows GUI

The ouput window displays messages issued by ngspice. You may scroll the window to get more of the text. The input box (white box) may be activated by a mouse click to accept any of the valid ngspice commends. The lower left output bar displays the actual input file. ngspice progress during setup and simulation is shown in the progress window ("–ready–"). The Quit button allow to interrupt ngspice. If ngspice is actively simulating, due to using only a single thread, this interrupt has to wait until the winow is accessible from within ngspice, e.g. during an update of the progress window.

In the plot window there is the upper left button, which activated a drop down menue. You may select to print the plot window shown (a very simple printer interface, to be improved), set up any of the printers available on your computer, or issue a postscript file of the actual plot window, either black&white or colored.

Insted of plotting with black background, you may set the background to any other colour, preferably to "white" using the command shown below.

Input file modification for white background:

```
.control
run
* white background
set color0=white
* black grid and text (only needed with X11, automatic with MS Win)
set color1=black
* wider grid and plot lines
set xbrushwidth=2
plot vss#branch
.endc
```
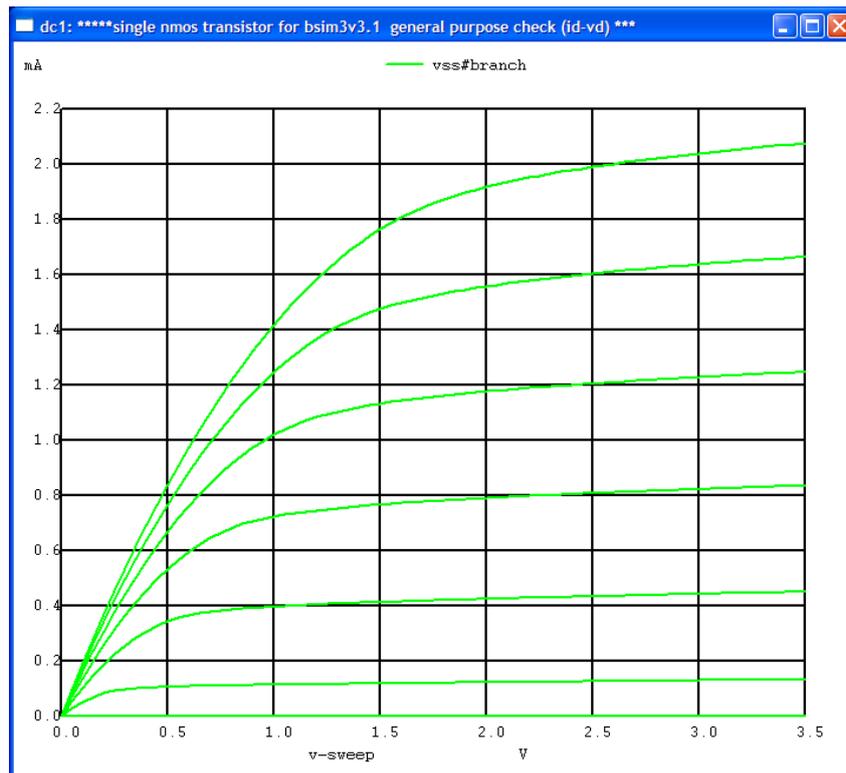
Figure 19.2: Plotting with white background

19.2   LINUX

to be written

19.3   Integration with CAD software and "third party" GUIs

In this chapter you will find some links and comments on GIUs for ngspice offered from other projects and on the integration of ngspice into a circuit development flow. The data given rely mostly on information available from the web and thus is out of our control. It also may be far from complete. The GUIs KJWaves and GNUSpiceGUI help you to navigate the commands to need to perform your simulation. XCircuit and the GEDA tools gschem and gnetlist offer integrating schematic capture and simulation.

19.3.1   KJWaves

KJWaves was written to be a cross-platform SPICE tool in pure Java. It aids in viewing, modifying, and simulating SPICE CIRCUIT files. Output from SPICE3 (ngspice) can be read and displayed. Resulting graphs may be printed and saved. The Java executable will run under LINUX and Windows (and maybe

other OSs). The development site is available at http://sourceforge.net/projects/kjwaves/. You may find the project home page at http://www.comefly.us/.

19.3.2   GNU Spice GUI

Another GUI, to be found at http://sourceforge.net/projects/gspiceui/.

19.3.3   XCircuit

CYGWIN and especially LINUX users may find XCircuit valuable to establish a development flow including schematic capture and circuit simulation.

19.3.4   GEDA

The gEDA project is developing a full GPL'd suite and toolkit of Electronic Design Automation tools for use with a LINUX. Ngspice may be integrated into the development flow. Two web sites offer tutorials using gschem and gnetlist with ngspice:

http://geda.seul.org/wiki/geda:csygas
http://geda.seul.org/wiki/geda:ngspice_and_gschem

Chapter 20

TCLspice

Spice historically comes as a simulation engine with a Command Line Interface. Spice engine now can be used with friendly Graphical User Interfaces. Tclspice represent a third approch to interfacing ngspice simulation functionalities. Tclspice is nothing more than a new way of compiling and using spice source code Spice is no longer considered as a standalone program but as a library invoqued by a TCL interpreter. It either permits direct simulation in a friendly TCL shell (this is quite analogous to the cli interface), or it permits the elaboration of more complex, more specific, or more user friendly simulation programs, by writing TCL scripts.

20.1 tclspice framework

The technical difference between the CLI interface and tclspice is that the CLI interface is compiled as a standalone program, whereas tclspice is a shared object. Tclspice is designed to work with tools that expands the capabilities of spice: TCL for the scripting and programming language interface and BLT for data processing and display. This two tools give tclspice all of its relevance, with the insurance the functionality is maintained by competent people.

Making tclspice produces two files: libspice.so and pkgIndex.tcl. libspice.so is the executable binary that the TCL interpreter calls to handle spice commands. pkgIndex.tcl take place in the TCL directory tree, providing the spice package[1] to the TCL user.

BLT is a TCL package. It is quite well documented. It permits to handle mathematical vectorial data structure for calculus and display, in a Tk interpreter like wish.

20.2 spicetoblt

Tclspice opens its doors to TCL and BLT with a single specific command spicetoblt.

TCLspice gets its identity in the command spice::vectoblt This command copies data computed by the simulation engine into a tcl variable. vectoblt is composed of three words: vec, to and blt. Vec means spice vector data. To is the English preposition, and blt is a useful tcl package providing a vector data structure. Example:

```
blt::vector create Iex
spice::vectoblt Vex#branch Iex
```

Here an empty blt vector is created. It is then filled with the vector representation of the current flowing out of source Vex. Vex#branch is native spices syntax. Iex is the name of the BLT vector.

The reverse operation is handled by native spice commands, such as alter, let and set.

20.3 Running TCLspice

TCLspice consists of a library or a package to include in your tcl console or script:

---

[1] package has to be understood as the TCL package

```
load /somepath/libspice.so
package require spice
```

Then you can execute any native spice command by preceding it with spice:: For example if you want to source the testCapa.cir netlist, type the following:

```
spice::source testCapa.cir
spice::spicetoblt example...
```

Plotting data is not a matter of spice, but of tcl. Once the data is stored in a blt vector, it can be plotted. Example:

```
blt::graph .cimvd −title "Cim = f(Vd)"
pack .cimvd
.cimvd element create line1 −xdata Vcmd −ydata Cim
```

With blt::graph a plotting structure is allocated in memory. With pack it is placed into the output window, and becomes visible. The last command, and not the least, plots the function Cim = f(Vcmd), where Cim and Vcmd are two BLT vectors.

20.4   examples

20.4.1   Active capacitor measurement

In this crude implementation of a circuit described by Marc KODRNJA, in his PhD thesis that I found on the Internet. This simulation outputs a graph representing the virtual capacitance versus the command voltage. The function $C = f(V)$ is calculated point by point. For each control voltage value, the virtual capacitance is calculated with the voltage and intensity accross the output port in a frequency simulation. A control value that should be as close to zero as possible is calculated to assess simulation success.

Invokation:

This script can be invoked by typing wish testbench1.tcl

testbench1.tcl

This line loads the simulator capabilities

```
package require spice
```

This is a comment (Quite useful if you intend to live with other Human beings)

```
# Test of virtual capacitore circuit
# Vary the control voltage and log the resulting capacitance
```

A good example of the calling of a spice command: preceed it with spice::

```
spice::source "testCapa.cir"
```

This reminds that any regular TCL command is of course possible

```
set n 30 set dv 0.2
set vmax [expr $dv/2]
set vmin [expr −1 ∗ $dv/2]
set pas [expr $dv/ $n]
```

BLT vector is the structure used to manipulate data. Instantiate the vectors

```
blt::vector  create  Ctmp
blt::vector  create  Cim
blt::vector  create  check
blt::vector  create  Vcmd
```

Data is, in my coding style, plotted into graph objects. Instantiate the graph

```
blt::graph  .cimvd  −title  "Cim = f(Vd)"
blt::graph  .checkvd  −title  "Rim = f(Vd)"
blt::vector  create  Iex
blt::vector  create  freq
blt::graph  .freqanal  −title  "Analyse frequentielle"
#
# First  simulation:  A  simple  AC  plot
#
set  v  [expr  {$vmin + $n ∗ $pas / 4}]
spice::alter  vd = $v
spice::op
spice::ac  dec  10  100  100k
```

Retrieve a the intensity of the current accross Vex source

```
spice::vectoblt  {Vex#branch}  Iex
```

Retrieve the frequency at wich the current have been assessed

```
spice::vectoblt  {frequency}  freq
```

Room the graph in the display window

```
pack  .freqanal
```

Plot the function Iex =f(V)

```
.freqanal  element  create  line1 −xdata  freq −ydata  Iex
#
# Second  simulation:  Capacitance  versus  voltage  control
# for  {set i 0}  {[expr $n − $i]}  {incr i }  { set  v  [expr  {$vmin + $i ∗ $pas}]
spice::alter  vd = $v
spice::op  spice::ac  dec  10  100  100k
```

Image capacitance is calculated by spice, instead of TCL there is no objective reason

```
spice::let  Cim = real(mean(Vex#branch/(2∗Pi∗i∗frequency ∗(V(5)−V(6)))))
spice::vectoblt  Cim  Ctmp
```

Build function vector point by point

```
Cim  append  $Ctmp(0:end)
```

Build a control vector to check simulation success

```
spice::let  err = real(mean(sqrt((Vex#branch −(2∗Pi∗i∗frequency ∗Cim∗V(5)−V(6)))^2)))
spice::vectoblt  err  Ctmp  check
append  $Ctmp(0:end)
```

Build abscissa vector

```
FALTA ALGO...  Vcmd  append  $v }
```

Plot

```
pack . cimvd
.cimvd element create line1 −xdata Vcmd −ydata Cim
pack . checkvd
.checkvd element create line1 −xdata Vcmd −ydata check
```

20.4.2   Optimisation of a linearization circuit for a Thermistor

This example is both the first and the last optimization program I wrote for an electronic circuit. It is far from perfect.

The temperature response of a CTN is exponential. It is thus nonlinear. In a battery charger application floating voltage varies linearly with temperature. A TL431 voltage reference sees its output voltage controlled by two resistors (r10, r12) and a thermistor (r11). The simulation is run at a given temperature. The thermistor is modeled in spice by a regular resistor. Its resistivity is assessed by the TCL script. It is set with a spice::alter command before running the simulation. This script uses an iterative optimization approach to try to converge to a set of two resistor values which minimizes the error between the expected floating voltage and the TL431 output.

Invokation:

This script can be executed by the user by simply executing the file in a terminal.

```
./testbench3.tcl
```

Two issues are important to point out[2]:

- During optimisation loop, graphical display of the current temperature response is not yet possible and I don't know why. Each time a simulation is performed, some memory is allocated for it.

- The simulation result remains in memory until the libspice library is unloaded (typically: when the tcl script ends) or when a spice::clean command is performed. In this kind of simulation, not cleaning the memory space will freeze your computer and you'll have to restart it. Be aware of that.

20.4.3   testbench3.tcl

This calls the shell sh who then runs wish with the file itself.

```
#!/bin/sh
# WishFix \
exec wish "$0" ${1+"$@"}
#
#
#
```

Regular package for simulation

```
package require spice
```

Here the important line is source differentiate.tcl which contains optimisation library

```
source differentiate.tcl
```

Generates a temperature vector

---

[2]For those who are really interested in optimizing circuits: Some parameters are very important for quick and correct convergence. The optimizer walks step by step to a local minimum of the cost function you define. Starting from an initial vector YOU provide, it converges step by step. Consider trying another start vector if the result is not the one you expected.

The optimizer will carry on walking until it reaches a vector which resulting cost is smaller than the target cost YOU provide it. You will also provide a maximum iteration count in case the target can not be achieved. Balance your time, specifications, and every other parameters. For a balance between quick and accurate convergence adjust the "factor" variable, at the beginning of minimumSteepestDescent in the file differentiate.tcl.

```
proc temperatures_calc {temp_inf temp_sup points} {
set tstep [ expr " ( $temp_sup − $temp_inf ) / $points " ]
set t $temp_inf
set temperatures ""
for { set i 0 } { $i < $points } { incr i } {
set t [ expr { $t + $tstep } ]
set temperatures "$temperatures $t"
}
return $temperatures }
```

generates thermistor resistivity as a vector, typically run: thermistance_calc res B [temperatures_calc temp_inf temp_sup points]

```
proc thermistance_calc { res B points } {
set tzero 273.15
set tref 25
set thermistance ""
foreach t $points {
set res_temp [expr " $res * exp ( $B * ( 1 / ($tzero + $t) − 1 / ( $tzero +
    $tref ) ) ) " ]
set thermistance "$thermistance $res_temp"
}
return $thermistance }
```

generates the expected floating value as a vector, typically run: tref_calc res B [temperatures_calc temp_inf temp_sup points]

```
proc tref_calc { points } {
set tref ""
foreach t $points {
set tref " $tref [ expr " 6 * (2.275 −0.005*($t − 20) ) − 9" ] "
}
return $tref }
```

In the optimisation algorithm, this function computes the effective floating voltage at the given temperature.

```
### NOTE:
### As component values are modified by a spice::alter Component values can
    be considered as global variable.
### R10 and R12 are not passed to iteration function because it is expected
    to be correct , ie to have been modified soon before proc iteration { t
    } { set tzero 273.15 spice::alter r11 = [ thermistance_calc 10000 3900
    $t ]
# Temperature simulation often crashes. Comment it out...
#spice::set temp = [ expr " $tzero + $t " ]
spice::op
spice::vectoblt vref_temp tref_tmp
###NOTE:
###As the library is executed once for the whole script execution , it is
    important to manage the memory
###and regularly destroy unused data set. The data computed here will not
    be reused. Clean it
spice::destroy all return [ tref_tmp range 0 0 ] }
```

This is the cost function optimization algorithm will try to minimize. It is a square norm of the error accross the temperature range [-25:75]°C (square norm: norme 2 in french I'm not sure of the english translation)

```
proc cost { r10 r12 } {
tref_blt length 0
spice::alter r10 = $r10
spice::alter r12 = $r12
foreach point [ temperatures_blt range 0 [ expr " [temperatures_blt length
    ] − 1" ] ] {
tref_blt append [ iteration $point ]
}
set result [ blt::vector expr " 1000 * sum(( tref_blt − expected_blt )^2 )"
    ]
disp_curve $r10 $r12
return $result }
```

This function displays the expected and effective value of the voltage, as well as the r10 and r12 resistor values

```
proc disp_curve { r10 r12 } { .g configure −title "Valeurs optimales: R10 =
    $r10 R12 = $r12" }
```

Main loop starts here

```
#
# Optimization
# blt::vector create tref_tmp
blt::vector create tref_blt
blt::vector create expected_blt
blt::vector create temperatures_blt temperatures_blt
append [ temperatures_calc −25 75 30 ] expected_blt
append [ tref_calc [temperatures_blt range 0 [ expr " [ temperatures_blt
    length ] − 1" ] ] ]
blt::graph .g
pack .g −side top −fill both −expand true
.g element create real −pixels 4 −xdata temperatures_blt −ydata tref_blt
.g element create expected −fill red −pixels 0 −dashes dot −xdata
    temperatures_blt −ydata expected_blt
```

Source the circuit and optimize it, result is retrieved in r10r12 variable and affected to r10 and r12 with a regular expression. A bit ugly.

```
spice::source FB14.cir
set r10r12 [ ::math::optimize::minimumSteepestDescent cost { 10000 10000 }
    0.1 50 ]
regexp {([0−9.]*) ([0−9.]*)} $r10r12 r10r12 r10 r12
```

Outputs optimization result

```
#
# Results
# spice::alter r10 = $r10
spice::alter r12 = $r12
foreach point [ temperatures_blt range 0 [ expr " [temperatures_blt length
    ] − 1" ] ] {
tref_blt append [ iteration $point ]
}
disp_curve $r10 $r12
```

20.4.4 Progressive display

This example is quite simple but it is very interesting. It displays a transient simulation result on the fly. You may now be familiar with most of the lines of this script. It uses the ability of BLT objects to automatically update. When the vector data is modified, the stripchart display is modified accordingly.

testbench2.tcl

```
#!/ bin / sh
# WishFix \
   exec wish −f "$0" ${1+"$@"}
###
package require BLT package require spice
```

this avoids to type blt:: before the blt class commands

```
namespace import blt ::∗
wm title . "Vector Test script"
wm geometry . 800x600+40+40 pack propagate . false
```

A strip chart with labels but without data is created and displayed (packed)

```
stripchart .chart
pack .chart −side top −fill both −expand true
.chart axis configure x −title "Time" spice :: source example.cir
spice :: bg
run after 1000 vector
create a0 vector
create b0 vectorry
create a1 vector
create b1 vector
create stime
proc bltupdate {} {
puts [spice :: spice_data]
spice :: spicetoblt a0 a0
spice :: spicetoblt b0 b0
spice :: spicetoblt a1 a1
spice :: spicetoblt b1 b1
spice :: spicetoblt time stime
after 100 bltupdate }
bltupdate .chart element create a0 −color red −xdata stime −ydata a0
.chart element create b0 −color blue −xdata stime −ydata b0
.chart element create a1 −color yellow −xdata stime −ydata a1
.chart element create b1 −color black −xdata stime −ydata b1
```

20.5 Compiling

TBW

Chapter 21

Example Circuits

This section starts with an ngspice example to walk you through the basic features of ngspice using its command line user interface. The operation of ngspice will be illustrated through several examples (chapters 18.1 to 18.6).

The first example uses the simple one-transistor amplifier circuit illustrated in Figure 21.1. This circuit is constructed entirely with SPICE3 compatible devices and is used to introduce basic concepts, including:

- Invoking the simulator:

- Running simulations in different analysis modes

- Printing and plotting analog results

- Examining status, including execution time and memory usage

- Exiting the simulator

The remainder of the section (from chapter 18.7 onwards) is designed to demonstrate XSPICE features. The second example circuit, shown in Figure C.2, models the circuit of Figure ... using the XSPICE gain block code model to substitute for the more complex and computationally expensive SPICE3 transistor model. This example illustrates one way in which XSPICE code models can be used to raise the level of abstraction in circuit modeling to improve simulation speed.

The third and final example, shown in Figure C.3, illustrates many of the more advanced features offered by: XSPICE. This circuit is a mixed-mode design incorporating digital data, analog data, and User-Defined Node data together in the same simulation. Some of the important features illustrated include:

- Creating and compiling Code Models

- Creating an XSPICE executable that incorporates these new models

- The use of "node bridge" models to translate data between the data types in the simulation

- Plotting analog and event-driven (digital and User-Defined Node) data

- Using the "eprint" command to print event-driven data

Throughout these examples, we assume that ngspice with XSPICE option has already been installed on your system and that your user account has been set up with the proper search path and environment variable data. If you experience problems, please see your system administrator for help.:

The examples also assume that you are running under UNIX and will use standard UNIX commands such as "cp" for copying files, etc. If you are using a different set up, with different operating system command names, you should be able to translate the commands shown into those suitable for your installation. Finally, file system pathnames given in the examples assume that ngspice + XSPICE has been installed on your system in directory "/usr/local/xspice-1-0". If your installation is different, you should substitute the appropriate root pathname where appropriate.
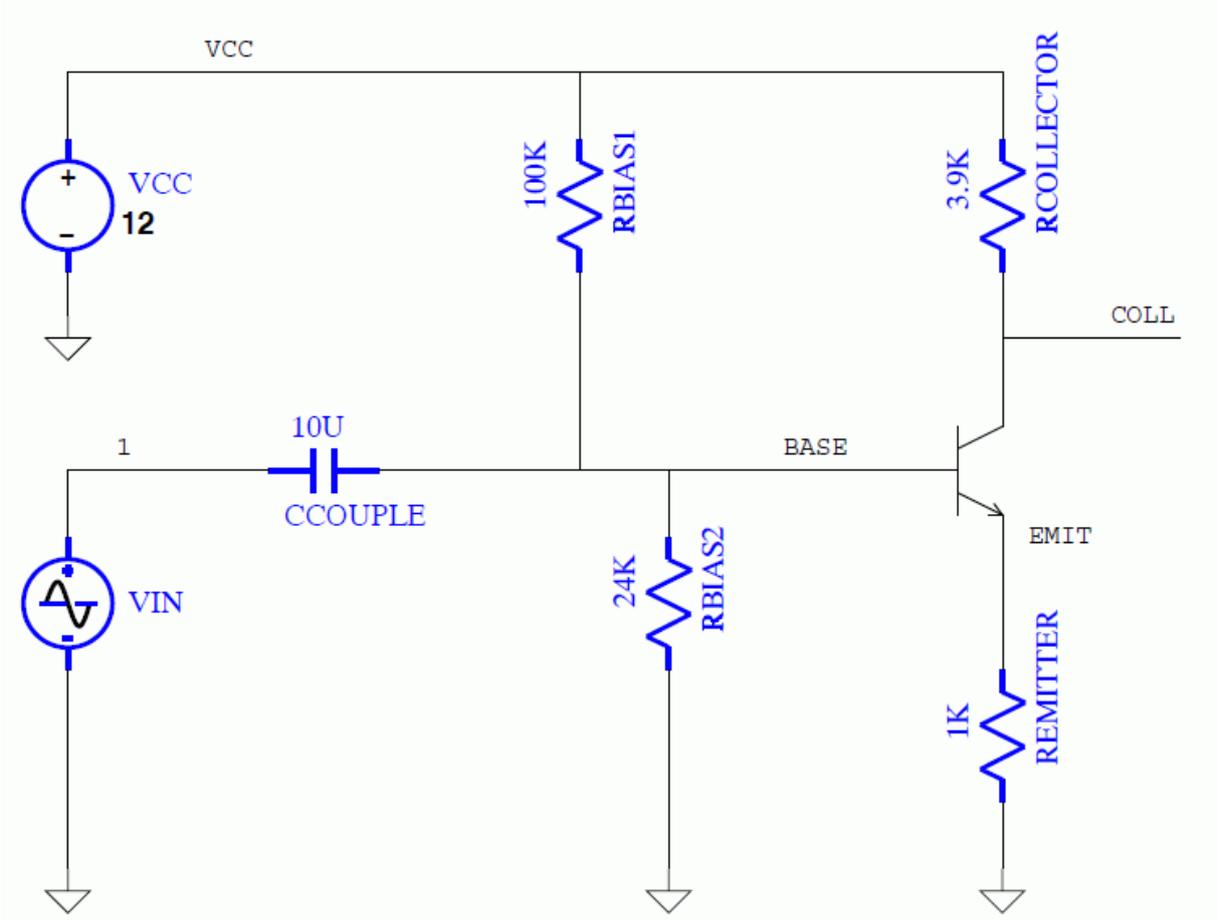
Figure 21.1: Transistor Amplifier Simulation Example

21.1   AC coupled transistor amplifier

The circuit shown in Figure C.1 is a simple one-transistor amplifier. The input signal is amplified with a gain of approximately -(Rc/Re) = -(3.9K/1K) = -3.9. The circuit description file for this example is shown below.

Example:

```
A Berkeley SPICE3 compatible circuit
*
* This circuit contains only Berkeley SPICE3 components.
*
* The circuit is an AC coupled transistor amplifier with
* a sinewave input at node "1", a gain of approximately -3.9,
* and output on node "coll".
*
.tran 1e-5 2e-3
*
vcc vcc 0 12.0
vin 1 0 0.0 ac 1.0 sin(0 1 1k)
ccouple 1 base 10uF
rbias1 vcc base 100k
rbias2 base 0 24k
q1 coll base emit generic
rcollector vcc coll 3.9k
remitter emit 0 1k
*
.model generic npn
*
.end
```

To simulate this circuit, move into a directory under your user account and copy the file `spice3.deck` from directory `/usr/local/xspice-1-0/lib/sim/examples`.

```
$ cp /usr/local/xspice-1-0/lib/sim/examples/spice3.deck spice3.deck
```

Now invoke the simulator on this circuit as follows:

```
$ ngspice spice3.deck
```

After a few moments, you should see the XSPICE prompt:

```
ngspice 1 ->
```

At this point, ngspice has read-in the circuit description and checked it for errors. If any errors had been encountered, messages describing them would have been output to your terminal. Since no messages were printed for this circuit, the syntax of the circuit description was correct.

To see the circuit description read by the simulator you can issue the following command:

```
ngspice 1 -> listing
```

The simulator shows you the circuit description currently in memory:

```
a berkeley spice3 compatible circuit
 1 : a berkeley spice3 compatible circuit
 2 : .global gnd
10 : .tran 1e-5 2e-3
12 : vcc vcc 0 12.0
13 : vin 1 0 0.0 ac 1.0 sin(0 1 1k)
14 : ccouple 1 base 10uf
15 : rbias1 vcc base 100k
16 : rbias2 base 0 24k
17 : q1 coll base emit generic
```

```
18 : rcollector vcc coll 3.9k
19 : remitter emit 0 1k
21 : .model generic npn
24 : .end
```

The title of this circuit is "A Berkeley SPICE3 compatible circuit". The circuit description contains a transient analysis control command `.TRAN 1E-5 2E-3` requesting a total simulated time of 2ms with a maximum timestep of 10us. The remainder of the lines in the circuit description describe the circuit of Figure C.1.

Before running this simulation, let's issue the "rusage" command to check the CPU time and memory used so far:

```
ngspice 2 -> rusage
Total elapsed time: 89.687 seconds.
Total DRAM available = 1535.480469 MB.
DRAM currently available = 1021.015625 MB.
Total ngspice program size = 4.804688 MB.
```

From this output we notice that the simulator used 1.3 seconds while reading in and parsing the circuit description and has used 237504 bytes of dynamically allocated memory so far (numbers may be somewhat different on your system).

Now, execute the simulation by entering the "run" command:

```
ngspice 3 -> run
```

The simulator will run the simulation and when execution is completed, will return with the ngspice prompt. When the prompt returns, issue the rusage command again to see how much time and memory has been used now.

```
ngspice 4 -> rusage
Total run time: 6.467 seconds.
Current data size = 270272,
Data limits: hard = 2147483647, soft = 2147483647.
Time since last call: 0.033 seconds.
```

From this information, we can compute that the total run time for this analysis was approximately (6.5 - 1.3) = 4.2 seconds and that (270272 - 237504) = 32768 additional bytes of dynamically allocated memory have been used.

To examine the results of this transient analysis, we can use the "plot" command. First we will plot the nodes labeled "1" and "base".

```
ngspice 5 -> plot v(1) base
```

The simulator responds by displaying an X Window System plot similar to that shown in Figure C.4.

Notice that we have named one of the nodes in the *circuit description* with a number ("1"), while the others are words ("base"). This was done to illustrate SPICE3's special requirements for plotting nodes labeled with numbers. Numeric labels are allowed in SPICE3 for backwards compatibility with SPICE2. However, they require special treatment in some commands such as "plot". The "plot" command is designed to allow expressions in its argument list in addition to names of results data to be plotted. For example, the expression `plot (base - 1)` would plot the result of subtracting 1 from the value of node "base".

If we had desired to plot the difference between the voltage at node "base" and node "1", we would need to enclose the node name "1" in the construction v( ) producing a command such as `plot (base - v(1))`.

Now, issue the following command to examine the voltages on two of the internal nodes of the transistor amplifier circuit:

```
ngspice 6 -> plot vcc coll
```

The plot shown in Figure C.5 should appear. Notice in the circuit description that the power supply voltage source and the node it is connected to both have the name "vcc". The plot command above has plotted the node voltage "vcc". However, it is also possible to plot branch currents through voltage sources in a circuit. SPICE3 always adds the special suffix "#branch" to voltage source names. Hence, to plot the current into the voltage source named "vcc", we would use a command such as plot vcc#branch.

Now let's run a simple DC simulation of this circuit and examine the bias voltages with the "print" command. One way to do this is to quit the simulator using the "quit" command, edit the input file to change the ".tran" line to ".op" (for 'operating point analysis'), reinvoke the simulator, and then issue the "run" command. However, ngspice allows analysis mode changes directly from the ngspice prompt. All that is required is to enter the control line, e.g. op (without the leading "."). ngspice will interpret the information on the line and start the new analysis run immediately, without the need to enter a new "run" command.

Figure C.5 Nutmeg Plot of VCC, Collector, and Emitter Voltages

To run the DC simulation of the transistor amplifier, issue the following command:

```
ngspice 7 -> op
```

After a moment the XSPICE prompt returns. Now issue the "print" command to examine the emitter, base, and collector DC bias voltages.

```
ngspice 8 -> print emit base coll
```

XSPICE responds with:

```
emit = 1.293993e+00 base = 2.074610e+00 coll = 7.003393e+00
```

To run an AC analysis, enter the following command:

```
ngspice 9 -> ac dec 10 0.01 100
```

This command runs a small-signal swept AC analysis of the circuit to compute the magnitude and phase responses. In this example, the sweep is logarithmic with "decade" scaling, 10 points per decade, and lower and upper frequencies of 0.01 Hz and 100 Hz. Since the command sweeps through a range of frequencies, the results are vectors of values and are examined with the plot command. Issue to the following command to plot the response curve at node "coll":

```
ngspice 10 -> plot coll
```

This plot shows the AC gain from input to the collector. (Note that our input source in the circuit description "vin" contained parameters of the form "AC 1.0" designating that a unit-amplitude AC signal was applied at this point.)

To produce a more traditional "Bode" gain phase plot with logarithmic scaling on the frequency axis, we use the expression capability of the "plot" command and the built-in Nutmeg functions db( ), log( ), and ph( ) together with the vs keyword:

```
ngspice 11 -> plot db(coll) ph(coll) vs log(frequency)
```

The last analysis supported by ngspice is a swept DC analysis. To perform this analysis, issue the following command:

```
ngspice 12 -> dc vcc 0 15 0.1
```

This command sweeps the supply voltage "vcc" from 0 to 15 volts in 0.1 volt increments. To plot the results, issue the command:

```
ngspice 13 -> plot emit base coll
```

Finally, to exit the simulator, use the "quit" command, and you will be returned to the operating system prompt.

```
ngspice 14 -> quit
```

So long.

21.2   Differential Pair

The following deck determines the dc operating point of a simple differential pair. In addition, the ac small-signal response is computed over the frequency range 1Hz to 100MEGHz.

Example:

```
SIMPLE DIFFERENTIAL PAIR
VCC  7  0  12
VEE  8  0  −12
VIN  1  0  AC  1
RS1  1  2  1K
RS2  6  0  1K
Q1  3  2  4  MOD1
Q2  5  6  4  MOD1
RC1  7  3  10K
RC2  7  5  10K
RE  4  8  10K
.MODEL MOD1 NPN BF=50 VAF=50 IS =1.E−12 RB=100 CJC=.5PF TF=.6NS
.TF  V(5)  VIN
.AC  DEC  10  1  100MEG
.END
```

21.3   MOSFET Characterization

The following deck computes the output characteristics of a MOSFET device over the range 0-10V for VDS and 0-5V for VGS.

Example:

```
MOS OUTPUT CHARACTERISTICS
.OPTIONS NODE NOPAGE
VDS  3  0
VGS  2  0
M1  1  2  0  0  MOD1  L=4U  W=6U  AD=10P  AS=10P
* VIDS MEASURES ID , WE COULD HAVE USED VDS, BUT ID WOULD BE NEGATIVE
VIDS  3  1
.MODEL MOD1 NMOS VTO=−2 NSUB=1.0E15 UO=550
.DC  VDS  0  10  .5  VGS  0  5  1
.END
```

21.4   RTL Inverter

The following deck determines the dc transfer curve and the transient pulse response of a simple RTL inverter. The input is a pulse from 0 to 5 Volts with delay, rise, and fall times of 2ns and a pulse width of 30ns. The transient interval is 0 to 100ns, with printing to be done every nanosecond.

Example:

```
SIMPLE RTL INVERTER
VCC  4  0  5
VIN  1  0  PULSE  0  5  2NS  2NS  2NS  30NS
RB  1  2  10K
Q1  3  2  0  Q1
RC  3  4  1K
.MODEL Q1 NPN BF 20 RB 100 TF .1NS CJC 2PF
.DC  VIN  0  5  0.1
.TRAN  1NS  100NS
.END
```

21.5 Four-Bit Binary Adder (Bipolar)

The following deck simulates a four-bit binary adder, using several subcircuits to describe various pieces of the overall circuit.

Example:

```
ADDER − 4 BIT ALL−NAND−GATE BINARY ADDER
*** SUBCIRCUIT DEFINITIONS
.SUBCKT NAND 1 2 3 4
* NODES: INPUT(2), OUTPUT, VCC
Q1 9 5 1 QMOD
D1CLAMP 0 1 DMOD
Q2 9 5 2 QMOD
D2CLAMP 0 2 DMOD
RB 4 5 4K
R1 4 6 1.6K
Q3 6 9 8 QMOD
R2 8 0 1K
RC 4 7 130
Q4 7 6 10 QMOD
DVBEDROP 10 3 DMOD
Q5 3 8 0 QMOD
.ENDS NAND

.SUBCKT ONEBIT 1 2 3 4 5 6
* NODES: INPUT(2), CARRY−IN, OUTPUT, CARRY−OUT, VCC
X1 1 2 7 6 NAND
X2 1 7 8 6 NAND
X3 2 7 9 6 NAND
X4 8 9 10 6 NAND
X5 3 10 11 6 NAND
X6 3 11 12 6 NAND
X7 10 11 13 6 NAND
X8 12 13 4 6 NAND
X9 11 7 5 6 NAND
.ENDS ONEBIT

.SUBCKT TWOBIT 1 2 3 4 5 6 7 8 9
* NODES: INPUT − BIT0(2) / BIT1(2), OUTPUT − BIT0 / BIT1,
* CARRY−IN, CARRY−OUT, VCC
X1 1 2 7 5 10 9 ONEBIT
X2 3 4 10 6 8 9 ONEBIT
.ENDS TWOBIT

.SUBCKT FOURBIT 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
* NODES: INPUT − BIT0(2) / BIT1(2) / BIT2(2) / BIT3(2),
* OUTPUT − BIT0 / BIT1 / BIT2 / BIT3, CARRY−IN, CARRY−OUT, VCC
X1 1 2 3 4 9 10 13 16 15 TWOBIT
X2 5 6 7 8 11 12 16 14 15 TWOBIT
.ENDS FOURBIT
```

Continue 4 Bit adder :

```
*** DEFINE NOMINAL CIRCUIT
.MODEL DMOD D
.MODEL QMOD NPN(BF=75 RB=100 CJE=1PF CJC=3PF)
VCC 99 0 DC 5V
VIN1A 1 0 PULSE(0 3 0 10NS 10NS 10NS 50NS)
VIN1B 2 0 PULSE(0 3 0 10NS 10NS 20NS 100NS)
VIN2A 3 0 PULSE(0 3 0 10NS 10NS 40NS 200NS)
VIN2B 4 0 PULSE(0 3 0 10NS 10NS 80NS 400NS)
VIN3A 5 0 PULSE(0 3 0 10NS 10NS 160NS 800NS)
VIN3B 6 0 PULSE(0 3 0 10NS 10NS 320NS 1600NS)
VIN4A 7 0 PULSE(0 3 0 10NS 10NS 640NS 3200NS)
VIN4B 8 0 PULSE(0 3 0 10NS 10NS 1280NS 6400NS)
X1 1 2 3 4 5 6 7 8 9 10 11 12 0 13 99 FOURBIT
RBIT0 9 0 1K
RBIT1 10 0 1K
RBIT2 11 0 1K
RBIT3 12 0 1K
RCOUT 13 0 1K


*** (FOR THOSE WITH MONEY (AND MEMORY) TO BURN)
.TRAN 1NS 6400NS
.END
```

21.6   Four-Bit Binary Adder (MOS)

The following deck simulates a four-bit binary adder, using several subcircuits to describe various pieces of the overall circuit.

Example:

```
   ADDER − 4 BIT ALL−NAND−GATE BINARY ADDER
*** SUBCIRCUIT DEFINITIONS
.SUBCKT NAND in1 in2 out VDD
*    NODES:  INPUT(2), OUTPUT, VCC
M1 out in2 Vdd Vdd p1 W=3u L=1u
M2 net.1 in2 0 0 n1 W=3u L=2u
M3 out in1 Vdd Vdd p1 W=3u L=1u
M4 out in1 net.1 0 n1 W=3u L=2u
.ENDS NAND
.SUBCKT ONEBIT 1 2 3 4 5 6 AND
X2   1  7  8  6    NAND
X3   2  7  9  6    NAND
X4   8  9 10  6    NAND
X5   3 10 11  6    NAND
X6   3 11 12  6    NAND
X7  10 11 13  6    NAND
X8  12 13  4  6    NAND
X9  11  7  5  6    NAND
.ENDS ONEBIT
.SUBCKT TWOBIT 1 2 3 4 5 6 7 8 9
*    NODES:  INPUT − BIT0(2) / BIT1(2), OUTPUT − BIT0 / BIT1,
*            CARRY−IN, CARRY−OUT, VCC
X1   1  2  7  5 10  9    ONEBIT
X2   3  4 10  6  8  9    ONEBIT
.ENDS TWOBIT
```

Continue 4 Bit adder MOS:

```
.SUBCKT FOURBIT 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
*NODES: INPUT - BIT0(2) / BIT1(2) / BIT2(2) / BIT3(2),
*        OUTPUT - BIT0 / BIT1 / BIT2 / BIT3, CARRY-IN, CARRY-OUT, VCC
X1    1   2   3   4   9  10  13  16  15     TWOBIT
X2    5   6   7   8  11  12  16  14  15     TWOBIT
.ENDS FOURBIT
*** DEFINE NOMINAL CIRCUIT
VCC    99   0    DC 3.3V
VIN1A  1   0    PULSE(0 3 0 10NS 10NS    10NS    50NS)
VIN1B  2   0    PULSE(0 3 0 10NS 10NS    20NS   100NS)
VIN2A  3   0    PULSE(0 3 0 10NS 10NS    40NS   200NS)
VIN2B  4   0    PULSE(0 3 0 10NS 10NS    80NS   400NS)
VIN3A  5   0    PULSE(0 3 0 10NS 10NS   160NS   800NS)
VIN3B  6   0    PULSE(0 3 0 10NS 10NS   320NS  1600NS)
VIN4A  7   0    PULSE(0 3 0 10NS 10NS   640NS  3200NS)
VIN4B  8   0    PULSE(0 3 0 10NS 10NS  1280NS  6400NS)
X1     1   2   3   4   5   6   7   8   9  10  11  12   0  13  99 FOURBIT
.option acct
.TRAN 1NS 1000NS
*.save VIN1A VIN1B VIN2A VIN2B VIN3A VIN3B VIN4A VIN4B
*.save V(1) V(2) V(3) V(4) V(5) V(6) V(7) V(8)

.include ./Modelcards/modelcard.nmos
.include ./Modelcards/modelcard.pmos

.END
```

21.7   Transmission-Line Inverter

The following deck simulates a transmission-line inverter. Two transmission-line elements are required since two propagation modes are excited. In the case of a coaxial line, the first line (T1) models the inner conductor with respect to the shield, and the second line (T2) models the shield with respect to the outside world.

Example:

```
TRANSMISSION-LINE INVERTER
V1 1 0 PULSE(0 1 0 0.1N)
R1 1 2 50
X1 2 0 0 4 TLINE
R2 4 0 50
.SUBCKT TLINE 1 2 3 4
T1 1 2 3 4 Z0=50 TD=1.5NS
T2 2 0 4 0 Z0=100 TD=1NS
.ENDS TLINE
.TRAN 0.1NS 20NS
.END
```

Chapter 22

Notes

22.1 Glossary

card A logical SPICE input line. A card may be extended through the use of the "+" sign in SPICE, thereby allowing it to take up multiple lines in a SPICE deck.

code model A model of a device, function, component, etc. which is based solely on a C programming language-based function. In addition to the code models included with the XSPICE simulator, you can use code models that you develop for circuit modeling.

deck A collection of SPICE cards which together specify all input information required in order to perform an analysis. A "deck" of "cards" will in fact be contained within a file on the host computer system.

element card A single, logical line in an XSPICE circuit description deck which describes a circuit element. Circuit elements are connected to each other to form circuits (e.g., a logical card which describes a resistor, such as R1 2 0 10K, is an element card).

instance A unique occurrence of a circuit element. See "element card", in which the instance "R1" is specified as a unique element (instance) in a hypothetical circuit description.

macro A macro, in the context of this document, refers to a Clanguage macro which supports the construction of userdefined models by simplifying input/output and parameterpassing operations within the Model Definition File.

.mod Refers to the Model Definition File. The file suffix reflects the filename of the model definition file: cfunc.mod.

.model Refers to a model card associated with an element card in XSPICE. A model card allows for data defining an instance to be conveniently located in the XSPICE deck such that the general layout of the elements is more readable.

Nutmeg The SPICE3C1 default post-processor. This provides a simple stand-alone simulator interface which can be used with the ATESSE simulator (see referenced documents section for additional information on Nutmeg).

subcircuit A "device" within an XSPICE deck which is defined in terms of a group of element cards and which can be referenced in other parts of the XSPICE deck through element cards.

22.2 Acronyms and Abbreviations

ATE Automatic Test Equipment

ATESSE Automatic Test Equipment Software Support Environment

CAE Computer-Aided Engineering

CCCS Current Controlled Current Source. In some cases, this is abbreviated ICIS.

CCVS Current Controlled Voltage Source. Also abbreviated as ICVS.

CSCI Computer Software Configuration Item

FET Field Effect Transistor

IDD Interface Design Document

IFS Refers to the Interface Specification File. The abbreviation reflects the filename of the Interface Specification File: ifspec.ifs.

MNA Modified Nodal Analysis

MOSFET Metal Oxide Semiconductor Field Effect Transistor

PWL Piece-Wise Linear

RAM Random Access Memory

ROM Read Only Memory

SDD Software Design Document

SI Simulator Interface

SIM The ATESSE Version 2.0 Simulator

SPICE Simulation Program with Integrated Circuit Emphasis. This program was developed at the University of California at Berkeley.

SPICE3 Version 3 of SPICE.

SRS Software Requirements Specification

SUM Software User's Manual

UCB University of California at Berkeley

UDN User-Defined Node(s)

VCCS Voltage Controlled Current Source. This is also sometimes abbreviated as VCIS.

VCIS Voltage Controlled Current Source.

VCVS Voltage Controlled Voltage Source

XSPICE Extended SPICE; synonymous with the ATESSE Version 2.0 Simulator.

Bibliography

[1] A. Vladimirescu and S. Liu, The Simulation of MOS Integrated Circuits Using SPICE2 ERL Memo No. ERL M80/7, Electronics Research Laboratory University of California, Berkeley, October 1980

[2] T. Sakurai and A. R. Newton, A Simple MOSFET Model for Circuit Analysis and its application to CMOS gate delay analysis and series-connected MOSFET Structure ERL Memo No. ERL M90/19, Electronics Research Laboratory, University of California, Berkeley, March 1990

[3] B. J. Sheu, D. L. Scharfetter, and P. K. Ko, SPICE2 Implementation of BSIM ERL Memo No. ERL M85/42, Electronics Research Laboratory University of California, Berkeley, May 1985

[4] J. R. Pierret, A MOS Parameter Extraction Program for the BSIM Model ERL Memo Nos. ERL M84/99 and M84/100, Electronics Research Laboratory University of California, Berkeley, November 1984

[5] Min-Chie Jeng, Design and Modeling of Deep Submicrometer MOSFETSs ERL Memo Nos. ERL M90/90, Electronics Research Labora tory University of California, Berkeley, October 1990

[6] Soyeon Park, Analysis and SPICE implementation of High Temperature Effects on MOSFET, Master's thesis, University of California, Berkeley, December 1986.

[7] Clement Szeto, Simulator of Temperature Effects in MOS FETs (STEIM), Master's thesis, University of California, Berkeley, May 1988.

[8] J.S. Roychowdhury and D.O. Pederson, Efficient Transient Simulation of Lossy Interconnect, Proc. of the 28th ACM/IEEE Design Automation Conference, June 17-21 1991, San Francisco

[9] A. E. Parker and D. J. Skellern, An Improved FET Model for Computer Simulators, IEEE Trans CAD, vol. 9, no. 5, pp. 551-553, May 1990.

[10] R. Saleh and A. Yang, Editors, Simulation and Modeling, IEEE Circuits and Devices, vol. 8, no. 3, pp. 7-8 and 49, May 1992.

[11] H.Statz et al., GaAs FET Device and Circuit Simulation in SPICE, IEEE Transactions on Electron Devices, V34, Number 2, February 1987, pp160-169.

[12] Weidong Liu et al., BSIM3v3.2.2 MOSFET Model Users' Manual, http://www-device.eecs.berkeley.edu/~bsim3/ftpv322/Mod_doc/V322manu.tar.Z

[13] Weidong Lui et al. BSIM3.v3.2.3 MOSFET Model Users' Manual, http://www-device.eecs.berkeley.edu/~bsim3/ftpv323/Mod_doc/BSIM323_manu.tar

[14] *SPICE3C.1 Nutmeg Programmer's Manual*, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California, April, 1987.

[15] *SPICE3 Version 3C1 User's Guide*, Thomas L. Quarles, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California, April, 1989.

[16] *The C Programming Language*, Second Edition, Brian Kernighan and Dennis Ritchie, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[17] "Code-Level Modeling in XSPICE", F.L. Cox, W.B. Kuhn, J.P. Murray, and S.D. Tynor, published in the Proceedings of the 1992 International Symposium on Circuits and Systems, San Diego, CA, May 1992, vol 2, pp. 871-874.

[18] "A Physically Based Compact Model of Partially Depleted SOI MOSFETs for Analog Circuit Simulation", Mike S. L. Lee, Bernard M. Tenbroek, William Redman-White, James Benson, and Michael J. Uren, IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 36, NO. 1, JANUARY 2001, pp. 110-121

[19] "A Realistic Large-signal MESFET Model for SPICE", A. E. Parker, and D. J. Skellern, IEEE Transactions on Microwave Theory and Techniques, vol. 45, no. 9, Sept. 1997, pp. 1563-1571.

Part II

XSPICE Software User's Manual

Chapter 23

XSPICE Basics

23.1 The XSPICE Code Model Subsystem

The second major component of XSPICE, the Code Model Subsystem, provides the tools needed to model the various parts of your system. While SPICE3 is targeted primarily at integrated circuit (IC) analysis, XSPICE includes features to model and simulate boardlevel and system-level designs as well. The Code Model Subsystem is central to this new capability, providing XSPICE with an extensive set of models to use in designs and allowing you to add your own models to this model set.

The SPICE3 simulator at the core of XSPICE includes built-in models for discrete components commonly found within integrated circuits. These "model primitives" include components such as resistors, capacitors, diodes, and transistors. The XSPICE Code Model Subsystem extends this set of primitives in two ways. First, it provides a library of over 40 additional primitives, including summers, integrators, digital gates, controlled oscillators, s-domain transfer functions, and digital state machines. Second, it provides a set of programming utilities to make it easy for you to create your own models by writing them in the C programming language.

23.2 XSPICE Top-Level Diagram

A top-level diagram of the XSPICE system outlined in the paragraphs above is shown in Figure 23.1. The XSPICE Simulator is made up of the SPICE3 core, the event-driven algorithm, circuit description syntax parser extensions, code model device routines, the Nutmeg user interface, and interprocess communications code used to integrate XSPICE with the ATESSE SI. The XSPICE Code Model Subsystem consists of the Code Model Toolkit, the Code Model Library, the Node Type Library, and interfaces to User-Defined Code Models and to User-Defined Node Types.
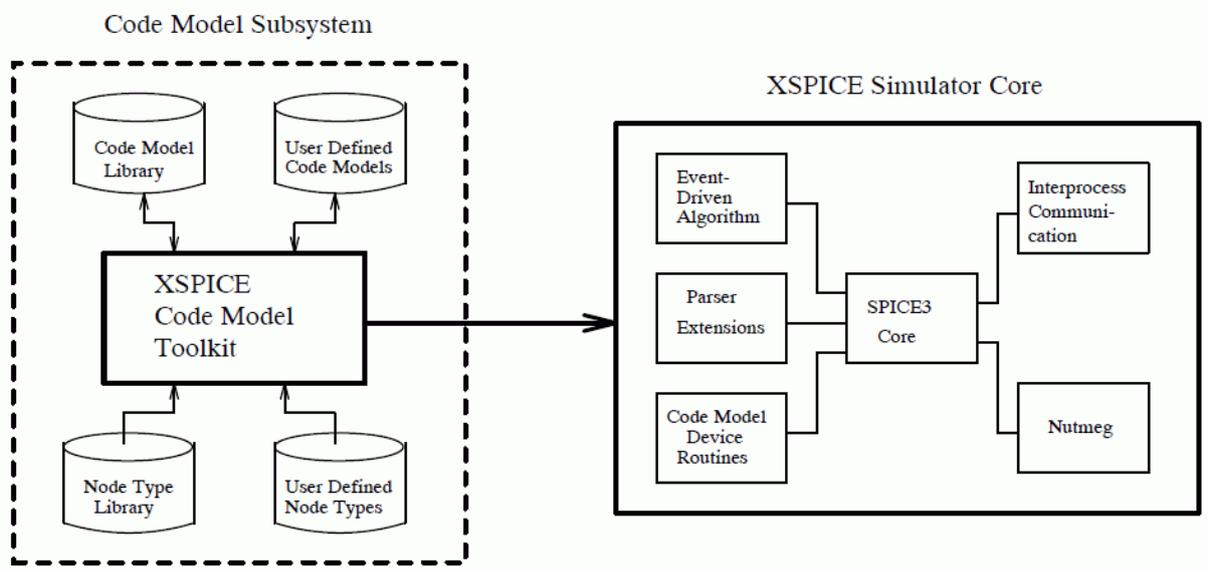
Figure 23.1: XSPICE Top-Level Diagram

Chapter 24

Execution Procedures

This chapter covers operation of the XSPICE simulator and the Code Model Subsystem. It begins with background material on simulation and modeling and then discusses the analysis modes supported in XSPICE and the circuit description syntax used for modeling. Detailed descriptions of the predefined Code Models and Node Types provided in the XSPICE libraries are also included.

24.1 Simulation and Modeling Overview

This section introduces the concepts of circuit simulation and modeling. It is intended primarily for users who have little or no previous experience with circuit simulators, and also for those who have not used circuit simulators recently. However, experienced SPICE users may wish to scan the material presented here since it provides background for the capabilities of XSPICE's new Code Model and User-Defined Node capabilities.

24.1.1 Describing the Circuit

This section provides an overview of the circuit description syntax expected by the XSPICE simulator.

If you are using the ATESSE SI user interface to the simulator, you do not need to create these circuit descriptions yourself. The ATESSE SI will automatically create a circuit description from your schematic and other information entered through menu selections. However, a general understanding of circuit description syntax will still be helpful to you should you encounter problems with your circuit and need to examine the simulator's error messages, or should you wish to develop your own models.

If you are using the Nutmeg user interface, this section will introduce you to the creation of circuit description input files.

In either case, note that this material is presented in an overview form. Details of circuit description syntax are given later in this chapter and in the references.

Example Circuit Description Input

Although different SPICE-based simulators may include various enhancements to the basic version from the University of California at Berkeley, most use a similar approach in describing circuits. This approach involves capturing the information present in a circuit schematic in the form of a text file that follows a defined format. This format requires the assignment of alphanumeric identifiers to each circuit node, the assignment of component identifiers to each circuit device, and the definition of the significant parameters for each device. For example, the circuit description below shows the equivalent input file for the circuit shown in Figure 24.1.

```
Small Signal Amplifier
*
*This circuit simulates a simple small signal amplifier.
*
Vin         Input 0                 0 SIN(0 .1 500Hz)
R_source    Input Amp_In            100
C1          Amp_In 0                1uF
R_Amp_Input Amp_In 0                1MEG
```
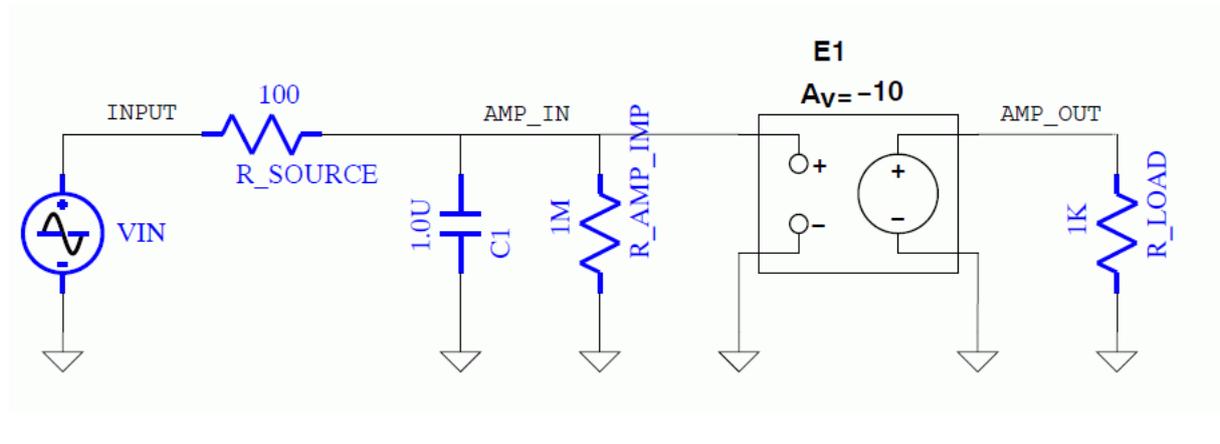
Figure 24.1: Example Circuit 1

```
E1              (Amp:Out 0) (Amp_In 0)    -10
R_Load          Amp_Out 0                 1000
*
.Tran 1.0u 0.01
*
.end
```

This file exhibits many of the most important properties common to all SPICE circuit description files including the following:

- The first line of the file is always interpreted as the title of the circuit. The title may consist of any text string.

- Lines which provide user comments, but no circuit information, are begun by an asterisk.

- A circuit device is specified by a device name, followed by the node(s) to which it is connected, and then by any required parameter information.

- The first character of a device name tells the simulator what kind of device it is (e.g. R = resistor, C = capacitor, E = voltage controlled voltage source).

- Nodes may be labeled with any alphanumeric identifier. The only specific labelling requirement is that 0 must be used for ground.

- A line that begins with a dot is a "control directive". Control directives are used most frequently for specifying the type of analysis the simulator is to carry out.

- An ".end" statement must be included at the end of the file.

- With the exception of the Title and .end statements, the order in which the circuit file is defined is arbitrary.

- All identifiers are case insensitive - the identifier 'npn' is equivalent to 'NPN' and to 'nPn'.

- Spaces and parenthesis are treated as white space.

- Long lines may be continued on a succeeding line by beginning the next line with a '+' in the first column.

In this example, the title of the circuit is 'Small Signal Amplifier'. Three comment lines are included before the actual circuit description begins. The first device in the circuit is voltage source 'Vin', which is connected between node 'Input' and '0' (ground). The parameters after the nodes specify that the source has an initial value of 0, a waveshape of 'SIN', and a DC offset, amplitude, and frequency of 0, .1, and 500 respectively. The next device in the circuit is resistor 'R_Source', which is connected between nodes 'Input' and 'Amp_In', with a value of 100 Ohms. The remaining device lines in the file are interpreted similarly.

The control directive that begins with '.Tran' specifies that the simulator should carry out a simulation using the Transient analysis mode. In this example, the parameters to the transient analysis control directive specify that the maximum timestep allowed is 1 microsecond and that the circuit should be simulated for 0.01 seconds of circuit time.

Other control cards are used for other analysis modes. For example, if a frequency response plot is desired, perhaps to determine the effect of the capacitor in the circuit, the following statement will instruct the simulator to perform a frequency analysis from 100 Hz to 10 MHz in decade intervals with ten points per decade.

```
.ac dec 10 100 10meg
```

To determine the quiescent operating point of the circuit, the following statement may be inserted in the file.

```
.op
```

A fourth analysis type supported by XSPICE is swept DC analysis. An example control statement for the analysis mode is

```
.dc Vin -0.1 0.2 .05
```

This statement specifies a DC sweep which varies the source Vin from -100 millivolts to +200 millivolts in steps of 50 millivolts.

Models and Subcircuits

The file discussed in the previous section illustrated the most basic syntax rules of a circuit description file. However, XSPICE (and other SPICE-based simulators) include many other features which allow for accurate modelling of semiconductor devices such as diodes and transistors and for grouping elements of a circuit into a macro or 'subcircuit' description which can be reused throughout a circuit description. For instance, the file shown below is a representation of the schematic shown in Figure 24.2.

```
Small Signal Amplifier with Limit Diodes
*
*This circuit simulates a small signal amplifier
*with a diode limiter.
*
.dc Vin -1 1 .05
*
Vin       Input 0 DC          0
R_source  Input Amp_In        100
*
D_Neg     0 Amp_In            1n4148
D_Pos     Amp_In 0            1n4148
*
C1        Amp_In 0            1uF
X1        Amp_In 0 Amp.Out    Amplifier
R_Load    Amp_Out 0           1000
*
.model 1n4148 D (is=2.495E-09 rs=4.755E-01 n=1.679E+00
+ tt=3.030E-09 cjo=1.700E-12 vj=1 m=1.959E-01 bv=1.000E+02
+ ibv=1.000E-04)
*
.subckt Amplifier Input Common Output
E1        (Output Common) (Input Common) -10
R_Input   Input           Common 1meg
.ends Amplifier
*
.end
```

This is the same basic circuit as in the initial example, with the addition of two components and some changes to the simulation file. The two diodes have been included to illustrate the use of device models, and the amplifier is implemented with a subcircuit. Additionally, this file shows the use of the swept DC control card.
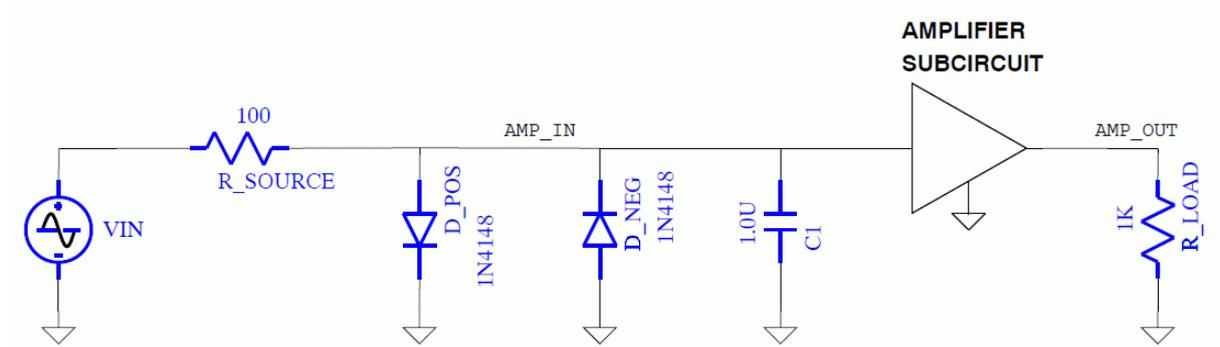
Figure 24.2: Example Circuit 2

3.1.2.2.1 Device Models   Device models allow you to specify, when required, many of the parameters of the devices being simulated. In this example, model statements are used to define the silicon diodes. Electrically, the diodes serve to limit the voltage at the amplifier input to values between about ±700 millivolts. The diode is simulated by first declaring the "instance" of each diode with a device statement. Instead of attempting to provide parameter information separately for both diodes, the label "1n4148" alerts the simulator that a separate model statement is included in the file which provides the necessary electrical specifications for the device ("1n4148" is the part number for the type of diode the model is meant to simulate).

The model statement that provides this information is:

```
.model 1n4148 D (is=2.495E-09 rs=4.755E-01 n=1.679E+00
+               tt=3.030E-09 cjo=1.700E-12 vj=1 m=1.959E-01
+               bv=1.000E+02 ibv=1.000E-04)
```

The model statement always begins with the string ".model" followed by an identifier and the model type (D for diode, NPN for NPN transistors, etc).

The optional parameters ('is', 'rs', 'n', 'etc.') shown in this example configure the simulator's mathematical model of the diode to match the specific behavior of a particular part (e.g. a "1n4148").

3.1.2.2.2 Subcircuits   In some applications, describing a device by imbedding the required elements in the main circuit file, as is done for the amplifier in Figure 24.1, is not desirable. A hierarchical approach may be taken by using subcircuits. An example of a subcircuit statement is shown in the second circuit file:

```
X1 Amp_In 0 Amp_Out
```

Amplifier Subcircuits are always identified by a device label beginning with "X". Just as with other devices, all of the connected nodes are specified. Notice, in this example, that three nodes are used. Finally, the name of the subcircuit is specified. Elsewhere in the circuit file, the simulator looks for a statement of the form:

```
.subckt <Name> <Node1> <Node2> <Node3> ...
```

This statement specifies that the lines that follow are part of the Amplifier subcircuit, and that the three nodes listed are to be treated wherever they occur in the subcircuit definition as referring, respectively, to the nodes on the main circuit from which the subcircuit was called. Normal device, model, and comment statements may then follow. The subcircuit definition is concluded with a statement of the form:

```
.ends <Name>
```

XSPICE Code Models

In the previous example, the specification of the amplifier was accomplished by using a SPICE Voltage Controlled Voltage Source device. This is an idealization of the actual amplifier. Practical amplifiers include numerous non-ideal effects, such as offset error voltages and non-ideal input and output impedances. The accurate simulation of complex, realworld components can lead to cumbersome subcircuit files, long

simulation run times, and difficulties in synthesizing the behavior to be modeled from a limited set of internal devices known to the simulator.

To address these problems, XSPICE allows you to create Code Models which simulate complex, non-ideal effects without the need to develop a subcircuit design. For example, the following file provides simulation of the circuit in Figure 24.2, but with the subcircuit amplifier replaced with a Code Model called 'Amp' that models several non-ideal effects including input and output impedance and input offset voltage.

```
Small Signal Amplifier
*
*This circuit simulates a small signal amplifier
*with a diode limiter.
*
.dc Vin -1 1 .05
*
Vin        Input 0              DC 0
R_source   Input Amp_In         100
*
D_Neg 0    Amp_In               1n4148
D_Pos      Amp_In 0             1n4148
*
C1         Amp_In 0             1uF
A1         Amp_In 0 Amp_Out     Amplifier
R_Load     Amp_Out 0            1000
*
.model 1n4148 D (is=2.495E-09 rs=4.755E-01 n=1.679E+00
+ tt=3.030E-09 cjo=1.700E-12 vj=1 m=1.959E-01 bv=1.000E+02
+ ibv=1.000E-04)
*
.model Amplifier Amp (gain = -10 in_offset = 1e-3
+                     rin = 1meg rout = 0.4)
*
.end
```

A statement with a device label beginning with "A" alerts the simulator that the device uses a Code Model. The model statement is similar in form to the one used to specify the diode. The model label 'Amp' directs XSPICE to use the code model with that name. Parameter information has been added to specify a gain of -10, an input offset of 1 millivolt, an input impedance of 1 meg ohm, and an output impedance of 0.4 ohm. Subsequent sections of this document detail the steps required to create such a Code Model and include it in the XSPICE simulator.

Node Bridge Models

When a mixed-mode simulator is used, some method must be provided for translating data between the different simulation algorithms. XSPICE's Code Model support allows you to develop models that work under the analog simulation algorithm, the event-driven simulation algorithm, or both at once.

In XSPICE, models developed for the express purpose of translating between the different algorithms or between different User-Defined Node types are called "Node Bridge" models. For translations between the built-in analog and digital types, predefined node bridge models are included in the XSPICE Code Model Library.

Practical Model Development

In practice, developing models often involves using a combination of SPICE passive devices, device models, subcircuits, and XSPICE Code Models. XSPICE's Code Models may be seen as an extension to the set of device models offered in standard SPICE. The collection of over 40 predefined Code Models included with XSPICE provides you with an enriched set of modeling primitives with which to build subcircuit models. In general, you should first attempt to construct your models from these available primitives. This is often the quickest and easiest method.

If you find that you cannot easily design a subcircuit to accomplish your goal using the available primitives, then you should turn to the code modeling approach. Because they are written in a general

purpose programming language (C), code models enable you to simulate virtually any behavior for which you can develop a set of equations or algorithms.

24.2   Circuit Description Syntax

If you are using the ATESSE system, you will enter most of the information about a circuit through graphical means, and the ATESSE Simulator Interface will automatically handle the translation of the graphical schematic into the XSPICE simulator's circuit description input language. Because of this, the circuit description language used by the simulator will rarely be visible to you unless you are developing models or need to examine error messages output by the simulator.

If you need to debug a simulation, if you are planning to develop your own models, or if you are using the XSPICE simulator through the Nutmeg user interface, you will need to become familiar with the circuit description language.

The previous sections presented example circuit description input files. The following sections provide more detail on XSPICE circuit descriptions with particular emphasis on the syntax for creating and using models. First, the language and syntax of the SPICE3 simulator are described and references to additional information are given. Next, XSPICE extensions to the SPICE3 syntax are detailed. Finally, various enhancements to SPICE operation are discussed including polynomial sources, arbitrary phase sources, supply ramping, matrix conditioning, convergence options, and debugging support.

24.2.1   XSPICE Syntax Extensions

In the preceding discussion, SPICE3C1 syntax was reviewed, and those features of SPICE3C1 that are specifically supported by the XSPICE simulator were enumerated. In addition to these features, there exist extensions to the SPICE3C1 capabilities that provide much more flexibility in describing and simulating a circuit. The following sections describe these capabilities, as well as the syntax required to make use of them

3.3.2.3.6 Convergence Debugging Support   When a simulation is failing to converge, the simulator can be asked to return convergence diagnostic information that may be useful in identifying the areas of the circuit in which convergence problems are occurring. When running the simulator with the ATESSE SI, these messages are included in the simulator text output. When running through the Nutmeg user interface, these messages are written directly to the terminal.

3.3.2.3.7 Digital Nodes   Support is included for digital nodes that are simulated by an event-driven algorithm. Because the event-driven algorithm is faster than the standard SPICE matrix solution algorithm, and because all "digital", "real", "int" and User-Defined Node types make use of the event-driven algorithm, reduced simulation time for circuits that include these models can be anticipated compared to simulation of the same circuit using analog code models and nodes.

3.3.2.3.8 User-Defined Nodes   Support is provided for User Defined Nodes that operate with the event-driven algorithm. These nodes allow the passing of arbitrary data structures among models. The real and integer node types supplied with XSPICE are actually predefined User-Defined Node types.

Chapter 25

Code Model Data TypeDefinitions

There are three data types which you can incorporate into a model and which have already been used extensively in the code model library included with the simulator. These are detailed below:

Boolean_t The Boolean type is an enumerated type which can take on values of FALSE (integer value 0) or TRUE (integer value 1). Alternative names for these enumerations are MIF FALSE and MIF TRUE, respectively.

Complex_t The Complex type is a structure composed of two double values. The first of these is the .real type, and the second is the .imag type. Typically these values are accessed as shown:
For complex value "data", the real portion is "data.real", and the imaginary portion is "data.imag".

Digital_State_t The Digital State type is an enumerated value which can be either ZERO (integer value 0), ONE (integer value 1), or UNKNOWN (integer value 2).

Digital_Strength_t The Digital Strength type is an enumerated value which can be either STRONG (integer value 0), RESISTIVE (integer value 1), HI IMPEDANCE (integer value 2) or UNDETERMINED (integer value 3).

Digital_t The Digital type is a composite of the Digital_State_t and Digital_Strength_t enumerated datatypes. The actual variable names within the Digital type are .state and .strength and are accessed as shown below:
For Digital_t value "data", the state portion is "data.state", and the strength portion is "data.strength".

Chapter 26

Example circuits

26.1 Amplifier with XSPICE model "gain"

The circuit shown in Figure C.2 is an abstract model of the circuit shown in Figure C.1, constructed using the XSPICE code model "gain". The ngspice circuit description for this circuit is shown below.
Example:

```
A transistor amplifier circuit
*
.tran 1e-5 2e-3
*
vin 1 0 0.0 ac 1.0 sin(0 1 1k)
*
ccouple 1 in 10uF
rzin in 0 19.35k
*
aamp in aout gain_block
.model gain_block gain (gain = -3.9 out_offset = 7.003)
*
rzout aout coll 3.9k
rbig coll 0 1e12
*
.end
```

Notice the component "aamp". This is an XSPICE code model device. All XSPICE code model devices begin with the letter "a" to distinguish them from other SPICE3 devices. The actual code model used is referenced through a user-defined identifier at the end of the line - in this case"gain_block". The type of code model used and its parameters appear on the associated .model card. In this example, the gain has been specified as -3.9 to approximate the gain of the transistor amplifier, and the output offset (out_offset) has been set to 7.003 according to the DC bias point information obtained from the DC analysis in Example 1.

Notice also that input and output impedances of the one-transistor amplifier circuit are modeled with the resistors "rzin" and "rzout", since the "gain" code model defaults to an ideal voltage-input, voltage-output device with infinite input impedance and zero output impedance.

Lastly, note that a special resistor "rbig" with value "1e12" has been included at the opposite side of the output impedance resistor "rzout". This resistor is required by SPICE3's matrix solution formula. Without it, the resistor "rzout" would have only one connection to the circuit, and an ill-formed matrix could result. One way to avoid such problems without adding resistors explicitly is to use the ngspice "rshunt" option described in this document under ngspice Syntax Extensions/General Enhancements.

To simulate this circuit, copy the file xspice.deck from the directory /usr/local/xspice-1-0/lib/sim/examples into a directory in your account.

```
$ cp /examples/xspice_c2.cir xspice_c2.cir
```

Invoke the simulator on this circuit:

```
$ ngspice xspice_c2.cir
```

After a few moments, you should see the XSPICE prompt:

```
ngspice 1 ->
```

Now issue the "run" command and when the prompt returns, issue the "plot" command to examine the voltage at the node "coll".

```
ngspice 1 -> run
ngspice 2 -> plot coll
```

The resulting waveform closely matches that from the original transistor amplifier circuit simulated in Example 1.

When you are done, enter the "quit" command to leave the simulator and return to the command line.

```
ngspice 3 -> quit
```

So long.

Using the "rusage" command, you can verify that this abstract model of the transistor amplfier runs somewhat faster than the full circuit of Example 1. This is because the code model is less complex computationally. This demonstrates one important use of XSPICE code models - to reduce run time by modeling circuits at a higher level of abstraction. Speed improvements vary and are most pronounced when a large amount of low-level circuitry can be replaced by a small number of code models and additional components.

26.2   XSPICE advanced usage

26.2.1   Circuit example C3

An equally important use of code models is in creating models for circuits and systems that do not easily lend themselves to synthesis using standard SPICE3 primitives (resistors, capacitors, diodes, transistors, etc.). This occurs often when trying to create models of ICs for use in simulating board-level designs. Creating models of operational amplfiers such as an LM741 or timer ICs such as an LM555 is greatly simplified through the use of XSPICE code models. Another example of code model use is shown in the next example where a complete sampled-data system is simulated using XSPICE analog, digital, and User-Defined Node types simultaneously.

C.3 Simulation Example 3   The circuit shown in Figure C.3 is designed to demonstrate several of the more advanced features of XSPICE. In this example, you will be introduced to the process of creating code models and linking them into a new version of the XSPICE simulator. You will also learn how to print and plot the results of event-driven analysis data. The XSPICE circuit description for this example is shown below.
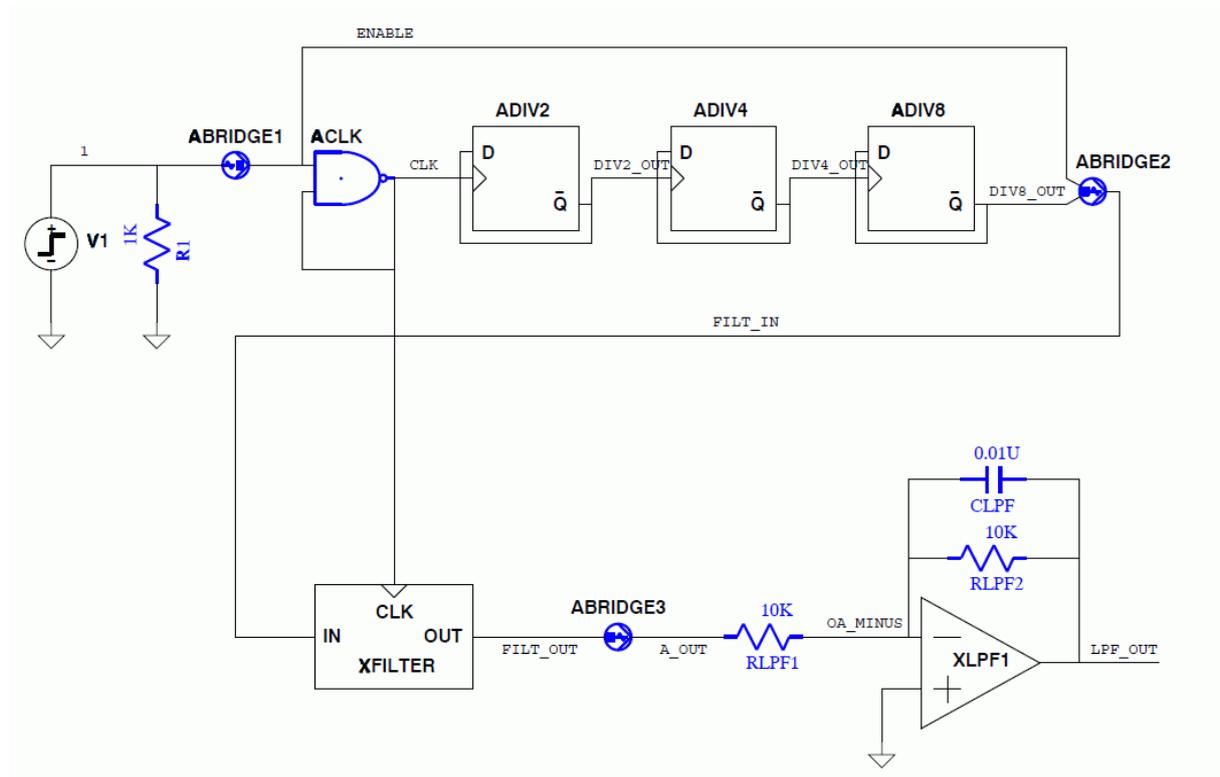
Figure 26.1: Example Circuit C3

Example:

```
Mixed IO types
* This circuit contains a mixture of IO types, including
* analog, digital, user-defined (real), and 'null'.
*
* The circuit demonstrates the use of the digital and
* user-defined node capability to model system-level designs
* such as sampled-data filters. The simulated circuit
* contains a digital oscillator enabled after 100us. The
* square wave oscillator output is divided by 8 with a
* ripple counter. The result is passed through a digital
* filter to convert it to a sine wave.
*
.tran 1e-5 1e-3
*
v1 1 0 0.0 pulse(0 1 1e-4 1e-6)
r1 1 0 1k
*
abridge1 [1] [enable] atod
.model atod adc_bridge
*
aclk [enable clk] clk nand
.model nand d_nand (rise_delay=1e-5 fall_delay=1e-5)
*
adiv2 div2_out clk NULL NULL NULL div2_out dff
adiv4 div4_out div2_out NULL NULL NULL div4_out dff
adiv8 div8_out div4_out NULL NULL NULL div8_out dff
.model dff d_dff
*
abridge2 div8_out enable filt_in node_bridge2
.model node_bridge2 d_to_real (zero=-1 one=1)
*
xfilter filt_in clk filt_out dig_filter
*
abridge3 filt_out a_out node_bridge3
.model node_bridge3 real_to_v
*
```

This circuit is a high-level design of a sampled-data filter. An analog step waveform (created from a SPICE3 "pulse" waveform) is introduced as "v1" and converted to digital by code model instance "abridge". This digital data is used to enable a Nand-Gate oscillator ("aclk") after a short delay. The Nand-Gate oscillator generates a squarewave clock signal with a period of approximately two times the gate delay, which is specified as 1e-5 seconds. This 50 KHz clock is divided by a series of D Flip Flops ("adiv2", "adiv4", "adiv8") to produce a squarewave at approximately 6.25 KHz. Note particularly the use of the reserved word "NULL" for certain nodes on the D Flip Flops. This tells the code model that there is no node connected to these ports of the flip flop.

The divide-by-8 and enable waveforms are converted by the instance "abridge2" to the format required by the User-Defined Node type "real", which expected real-valued data. The output of this instance on node "filt_in" is a real valued square wave which oscillates between values of -1 and 1. Note that the associated code model "d_to_real" is not part of the original XSPICE code model library but has been added later to ngspice.

This signal is then passed through subcircuit "xfilter" which contains a digital lowpass filter clocked by node "clk". The result of passing this squarewave through the digital lowpass filter is the production of a sampled sine wave (the filter passes only the fundamental of the squarewave input) on node "filt_out". This signal is then converted back to SPICE analog data on node "a_out" by node bridge instance "abridge3".

The resulting analog waveform is then passed through an opamp-based lowpass analog filter constructed around subcircuit "xlpf" to produce the final output at analog node "lpf_out".

26.2.2   How to create code models

The following insruction to create the missing code models is obsolete, because the code models used in the above example are part of ngspice. It may be instructive however to see how to create additional code models. More detailed information is available in chapter 27.

The four additional code models (compared to original XSPICE) used in the circuit are:

d_to_real, real_to_v, real_gain, real_delay.

To construct these models, we will use the XSPICE Code Model Toolkit. However, to avoid typing in all of the model code, we will be copying files from the "examples" directory once the model directories have been created.

First, create the code model "d_to_real". To do so, move into a directory under your user account and invoke the Code Model Toolkit's "mkmoddir" command:

```
$ mkmoddir d_to_real SPICE model name [d_to_real]: C function name [ucm_d_to_real]:
Model Directory "d_to_real" created.
```

Edit files "ifspec.ifs" and "cfunc.mod" to define your model. Then run "make" to preprocess and compile it.

and press RETURN to accept the defaults when prompted for data. This creates a model directory named "d_to_real" and installs three files in it - a "Makefile", an Interface Specification File, and a Model Definition File.

Now move into this new directory:

```
$ cd d_to_real
```

and examine the Interface Specification and Model Definition files (ifspec.ifs and cfunc.mod). As explained in Chapter 3 of this document, these files are used to specify the model's inputs, output, and parameters, and to code the models behavior in the C programming language with help from the Code Model Toolkit's "accessor macros" and function library.

To save time in this example, we will copy these files from the directory /usr/local/xspice-1-0/lib/sim/examples/d.to.rea

```
$ cp /usr/local/xspice-1-0/lib/sim/examples/d.to.real/ifspec.ifs ifspec.ifs
$ cp /usr/local/xspice-1-0/lib/sim/examples/d.to.real/cfunc.mod cfunc.mod
```

You may wish to examine these files once you have copied them. When you are done, issue the UNIX "make" command to send them through the XSPICE Code Model Preprocessor utility ("cmpp") and then through the C compiler to create the necessary object files to be bound in with the simulator.

```
$ make
/usr/local/xspice-1-0/bin/cmpp -ifs cc -g -I. -I/usr/local/xspice-1-0/include/sim -c ifspec.c /u
```

Now move back up to the parent directory:

```
$ cd ..
```

and repeat this process for the three remaining code models required for the simulation (real_to_v, real_gain, and real_delay). Note that the process of compiling code models is automated by the XSPICE Code Model Toolkit and the associated UNIX "make" command. Once the interface specifications and model definitions for the models are developed, the process of compiling the models is reduced to three easy-to-remember steps:

- Create the model with the "mkmoddir" command.

- Edit the template files "ifspec.ifs" and "cfunc.mod" to specify the model's interface and it's behavior.

- Build the model by typing "make".

The body of this document tells you how to go about designing and coding a model's interface and behavior. In addition, numerous examples can be found in the form of the models in the XSPICE code model library.

Now that our models are ready, the remaining step is to link them with the ngspice "core" to create a new XSPICE executable. For this, we use the Code Model Toolkit's "mksimdir" command. Move to the directory in which all the models were created if you are not already there. Then enter the command:

```
$ mksimdir mysim
```

Simulator directory "mysim" created. Edit files "modpath.lst" and "udnpath.lst" to specify desired models and node types respectively. Then run "make" to build the simulator executable.

When the operating system prompt returns, move into this new directory, and edit the file "modpath.lst". This file holds the pathnames to model directories containing models to be included in the simulator. The file initially includes all the models in the XSPICE Code Model Library. You may add and/or delete files from the list according to your anticipated needs. For this example, we will simply add the four models we have just compiled at the bottom of the file, with one pathname per line.

Add the following lines at the bottom of the file:

```
../d_to_real
../real_to_v
../real_gain
../real_delay
```

Save this edited file, return to the operating system prompt, and enter the UNIX "make" command (Note: Making the simulator may take a couple of minutes depending on the number of models included).

```
$ make
```

Running preprocessor on modpath.lst and udnpath.lst ... /usr/local/xspice-1-0/bin/cmpp -lst
    Compiling list of models and node types... cc -o temp.o -g -I. -I/usr/local/xspice-1-0/include/sim ...
    Linking XSPICE simulator ... cc -o xspice temp.o " /usr/local/xspice-1-0/lib/sim/object/core.o -L/usr/X11/lib ... /usr/local/xspice-1-0/src/cml/aswitch/*.o " /usr/local/xspice-1-0/src/cml/climit/*.o " /usr/local/xspice-1-0/src/cml/d.dt/*.o " /usr/local/xspice-1-0/src/cml/divide/*.o " /usr/local/xspice-1-0/src/cml/gain/*.o " /usr/local/xspice-1-0/src/cml/hyst/*.o " /usr/local/xspice-1-0/src/cml/ilimit/*.o "
    ... /usr/local/xspice-1-0/src/cml/d.dff/*.o " /usr/local/xspice-1-0/src/cml/d.jkff/*.o " /usr/local/xspice-1-0/src/cml/d.tff/*.o " /usr/local/xspice-1-0/src/cml/d.srff/*.o " /usr/local/xspice-1-0/src/cml/d.dlatch/*.o " /usr/local/xspice-1-0/src/cml/d.srlatch/*.o " /usr/local/xspice-1-0/src/cml/d.state/*.o " /usr/local/xspice-1-0/src/cml/d.osc/*.o " ../d.to.real/*.o " ../real.to.v/*.o " ../real.gain/*.o " ../real.delay/*.o " /usr/local/xspice-1-0/src/udnl/real/*.o " /usr/local/xspice-1-0/src/udnl/int/*.o
    Deleting temporary files ...
    XSPICE simulator created. Type: "xspice !input deck?" to run.

26.2.3   Running example C3

Now copy the file "mixed mode.deck" from directory /usr/local/xspice-1-0/lib/sim/examples into the "mysim" directory:

```
$ cp /examples/xspice_c3.cir xspice_c3.cir
```

and invoke the new simulator executable as you did in the previous examples.

```
$ ngspice xspice_c3.cir
```

Execute the simulation with the "run" command.

```
ngspice 1 -> run
```

After several seconds, the ngspice prompt should return. Results of this simulation are examined in the manner illustrated in the previous two examples. You can use the "plot" command to plot either analog nodes, event-driven nodes, or both. For example, you can plot the values of the sampled-data filter input node and the analog lowpass filter output node as follows:

Figure C.6 Nutmeg Plot of Filter Input and Output

```
ngspice 2 -> plot filt_in lpf_out
```

The plot shown in Figure C.6 should appear. You can also plot data from nodes inside a subcircuit. For example, to plot the data on node "x1a" in subcircuit "xfilter", create a pathname to this node in reverse order with a colon separator.

```
ngspice 3 -> plot x1a:xfilter
```

The output from this command is shown in Figure C.7. Note that the waveform contains vertical segments. These segments are caused by the non-zero delays in the "real gain" models used within the subcircuit. Each vertical segment is actually a step with a width equal to the model delay (1e-9 seconds).

Plotting nodes internal to subcircuits works for both analog and event-driven nodes. The reverse order format arises because instance names are expanded similarly when processing subcircuits and SPICE uses the first character of a name to determine the device type.

Figure C.7 Nutmeg Plot of Subcircuit Internal Node   Hence, by building the name in reverse order, the first character of the instance in the subcircuit is unchanged in the expanded name.

To examine data such as the closely spaced events inside the subcircuit at node "x1a:xfilter", it is often convenient to use the "eprint" command to produce a tabular listing of events. Try this by entering the following command:

```
ngspice 4 -> eprint x1a:xfilter
```

**** Results Data **** Time or Step x1a:xfilter
    0.000000000e+00 0.000000e+00 1.010030000e-04 2.000000e+00 1.010040000e-04 2.562500e+00 1.210020000e-04 2.812500e+00 1.210030000e-04 4.253906e+00
    232
    XSPICE Simulator Software User's Manual
    XSPICE/Nutmeg Simulation Examples
    Simulation Example 3
    1.410020000e-04 2.332031e+00 1.410030000e-04 3.283447e+00 1.610020000e-04 2.014893e+00 1.610030000e-04 1.469009e+00 1.810020000e-04 2.196854e+00 1.810030000e-04 1.176232e+00 1.810090000e-04 -2.823768e+00 1.810100000e-04 -3.948768e+00 2.010020000e-04 -3.087939e+00 2.010030000e-04 -6.135439e+00 2.210020000e-04 -2.072106e+00 2.210030000e-04 -3.302109e+00
    ... 9.010090000e-04 3.049473e+00 9.010100000e-04 4.174473e+00 9.210020000e-04 2.867375e+00 9.210030000e-04 5.380142e+00 9.410020000e-04 2.029786e+00 9.410030000e-04 2.707975e+00 9.610020000e-04 1.803723e+00 9.610030000e-04 3.006294e-01 9.810020000e-04 2.304755e+00 9.810030000e-04 9.506230e-01 9.810090000e-04 -3.049377e+00 9.810100000e-04 -4.174377e+00

**** Messages **** **** Statistics **** Operating point analog/event alternations: 1 Operating point load calls: 37 Operating point event passes: 2 Transient analysis load calls: 4299 Transient analysis timestep backups: 87

This command produces a tabular listing of event-times in the first column and node values in the second column. The 1 ns delays can be clearly seen in the fifth decimal place of the event times.

Note that the eprint command also gives statistics from the event-driven algorithm portion of XSPICE. For this example, the simulator alternated between the analog solution algorithm and the event-driven algorithm one time while performing the initial DC operating point solution prior to the start of the transient analysis. During this operating point analysis, 37 total calls were made to event-driven code model functions, and two separate event passes or iterations were required before the event nodes obtained stable values. Once the transient analysis commenced, there were 4299 total calls to event-driven code model functions. Lastly, the analog simulation algorithm performed 87 timestep backups that forced the event-driven simulator to backup its state data and its event queues.

A similar output is obtained when printing the values of digital nodes. For example, print the values of the node "div8 out" as follows:

```
ngspice 5 -> eprint div8.out
```

**** Results Data **** Time or Step div8.out

0.000000000e+00 1s 1.810070000e-04 0s 2.610070000e-04 1s 3.410070000e-04 0s 4.210070000e-04 1s 5.010070000e-04 0s 5.810070000e-04 1s 6.610070000e-04 0s 7.410070000e-04 1s 8.210070000e-04 0s 9.010070000e-04 1s 9.810070000e-04 0s

**** Messages **** **** Statistics **** Operating point analog/event alternations: 1 Operating point load calls: 37 Operating point event passes: 2 Transient analysis load calls: 4299 Transient analysis timestep backups: 87

From this printout, we see that digital node values are composed of a two character string. The first character (0, 1, or U) gives the state of the node (logic zero, logic one, or unknown logic state). The second character (s, r, z, u) gives the "strength" of the logic state (strong, resistive, hi-impedance, or undetermined).

If you wish, examine other nodes in this circuit with either the plot or eprint commands. When you are done, enter the "quit" command to exit the simulator and return to the operating system prompt:

```
ngspice 6 -> quit
```

So long.

Chapter 27

Code Models and User-Defined Nodes

The following sections explain the steps required to create code models and User-Defined Nodes (UDNs) and link them into the simulator. The XSPICE simulator includes libraries of predefined models and node types that span the analog and digital domains. These have been detailed earlier in this document (see Sections 13.2, 13.3, and 13.4). However, the real power of the XSPICE simulator is in its support for extending these libraries with new models written by users. In order to provide this capability, XSPICE includes a "Code Model Toolkit" that enables you to define new models and node data types to be passed between them. These models are handled by XSPICE in a manner analogous to its handling of SPICE devices and XSPICE Predefined Code Models.The basic steps required to create code models or User-Defined Nodes and link them into the XSPICE simulator are similar. They consist of 1) creating the code model or UserDefined Node (UDN) directory and its associated model or data files, and 2) creating a simulator directory (or returning to the existing simulator directory) and linking the new files into a new XSPICE simulator executable. Once the simulator executable has been created, instances of models, can be placed into any simulator deck that describes a circuit of interest and simulated along with all of the other components in that circuit.

27.1 Creating Code Models

The first step in creating a new code model within XSPICE is to create a model directory containing a UNIX 'Makefile' and the following template files: 'Makefile'.

- Interface Specification File

- Model Definition File

After this, the template Interface Specification File (ifspec.ifs) is edited to define the model's inputs, outputs, parameters, etc. You then edit the template Model Definition File (cfunc.mod) to include the C-language source code that defines the model behavior. Once this is done, the files are preprocessed by the XSPICE Code Model Toolkit under the direction of the UNIX Makefile and then compiled into object files ready to be linked into a simulator executable.

The first step in the process of producing a code model, that of creating the model directory and the associated template files, is handled automatically. You simply execute the "mkmoddir" command in a UNIX shell as follows:

```
mkmoddir <directory name>
```

This command creates the named directory, a "Makefile", and the two template files ifspec.ifs and cfunc.mod. You then edit the ifspec.ifs and cfunc.mod files to define your code model. A complete list of the steps taken to create a model follows:

1. In a UNIX shell, execute the command "mkmoddir" to create a directory containing a "Makefile" and templates for an ifspec.ifs file and a cfunc.mod file.

2. Move into the newly created directory using the UNIX command "cd".

3. Edit the Interface Specification template file (ifspec.ifs) to specify the model's name, ports, parameters, and static variables.

4. Edit the model definition template file (cfunc.mod) to include the C-language source code that defines the model's behavior.

5. Execute the UNIX command "make" to preprocess and compile the Interface Specification and Model Definition files.

The Interface Specification File is a text file that describes, in a tabular format, information needed for the code model to be properly interpreted by the simulator when it is placed with other circuit components into a SPICE deck. This information includes such things as the parameter names, parameter default values, and the name of the model itself. The specific format presented to you in the Interface Specification File template must be followed exactly, but is quite straightforward. A detailed description of the required syntax, along with numerous examples, is included in Section 27.2.

The Model Definition File contains a C programming language function definition. This function specifies the operations to be performed within the model on the data passed to it by the simulator. Special macros are provided that allow the function to retrieve input data and return output data. Similarly, macros are provided to allow for such things as storage of information between iteration timepoints and sending of error messages. Section 27.3 describes the form and function of the Model Definition File in detail and lists the support macros provided within the simulator for use in code models.

27.2   Creating User-Defined Nodes

In addition to providing the capability of adding new models to the simulator, a facility exists which allows node types other than those found in standard SPICE to be created. Models may be constructed which pass information back and forth via these nodes. Such models are constructed in the manner described in the previous sections, with appropriate changes to the Interface Specification and Model Definition Files.

Because of the need of electrical engineers to have ready access to both digital and analog simulation capabilities, the "digital" node type is provided as a built-in node type along with standard SPICE analog nodes. For "digital" nodes, extensive support is provided in the form of macros and functions so that you can treat this node type as a standard type analogous to standard SPICE analog nodes when creating and using code models. In addition to "analog" and "digital" nodes, the node types "real" and "int" are also provided with the simulator. These were created using the User-Defined Node (UDN) creation facilities described below.

The first step in creating a new node type within XSPICE is to set up a node type directory along with the appropriate template files needed. After this, the UDN Definition File (cfunc.udn) is edited to provide the node-specific functions which will be needed to support code models using this node type.

The first step in the process of producing a UDN type, that of creating the UDN directory and the associated template files, is handled automatically. You simply execute the "mkudndir" command in a UNIX shell as follows:

```
mkudndir <directory name>
```

This command creates the named directory, a "Makefile", and the template file udnfunc.c. You may then edit the template file as described below. A complete list of the steps necessary to create a User-Defined Node type follows:

1. In a UNIX shell, execute the command "mkudndir" to create a directory containing a "Makefile" and a template for a udnfunc.c file.

2. Move into the newly created directory using the UNIX command "cd".

3. Edit the udnfunc.c template file to code the required C functions (see Section 27.4 for details).

4. Execute the UNIX command "make" to preprocess and compile the node functions.

The UDN Definition File contains a set of C language functions. These functions perform operations such as allocating space for data structures, initializing them, and comparing them to each other. Section 27.4 describes the form and function of the User-Defined Node Definition File in detail and includes an example UDN Definition File.

27.3 Compiling and Linking the Simulator

The concept of creating a new "version" of XSPICE whenever a code model needs to be added is probably foreign to most users. However, the advantages gained from taking this approach are considerable. Most mixed-mode simulators are closed systems; the set of models they provide cannot be extended by the average user. In many cases, even the creators of the original software are not in a position to easily add to the set of models. Consequently, when the need arises for new models they must be defined as subcircuits based on the built-in models.

For simple devices, the synthesis of models from the set of built-in models does not necessarily lead to a degradation of simulator performance. However, if you wish to build up a model that does not lend itself readily to such a synthesis, penalties in the form of increased simulation time and lower modeling accuracy can result.

With this in mind, XSPICE was constructed so that you can readily add to the lowest level of simulator functionality simply by creating a new model or User-Defined Node type and linking it into a new simulator executable. This process is described in the following paragraphs.

The first step in creating a new instatnce of the XSPICE simulator is to set up a simulation directory along with the appropriate model-list template file and User-Defined Node-list template file. After this, the model-list template and User-Defined Node list template files should be edited to specify the path names to directories containing the code models to be incorporated into the simulator, and the path names to any User-Defined Node type definitions required by the simulator, respectively. Once this is done, the files are preprocessed by the Code Model Toolkit and the executable is built.

The first step in the process of producing a working version of the simulator, that of creating the simulator directory and the associated template files, is handled automatically for you. You simply execute the "mksimdir" command in a UNIX shell as follows:

```
mksimdir <directory name>
```

This command creates the named directory, a "Makefile", and the model-list and UserDefined Node-list template files. The template files may then be edited as described below. A complete list of the steps necessary to create a new version of the simulator follows:

1. In a UNIX shell, execute the command "mksimdir" to create a directory containing a "Makefile" and the template files for the model list file and the UDN-list file.

2. Move into the newly created directory using the UNIX command "cd".

3. Edit the model list template file to indicate the path names to directories containing the desired code models.

4. Edit the User-Defined Node list template file to indicate the path names to directories containing the required User-Defined Node types.

5. Type "make xspice" or "make atesse_xspice" to preprocess the list files, link the specified models and node types, and create the desired simulator executable.

Making "xspice" will create a stand-alone simulator that incorporates the Nutmeg user interface. Making "atesse xspice" creates a version of the simulator suitable for use with the ATESSE SI user interface.

27.4 Interface Specification File

The Interface Specification (IFS) file is a text file that describes the model's naming information, its expected input and output ports, its expected parameters, and any variables within the model that are to be used for storage of data across an entire simulation. These four types of data are described to the simulator in IFS file sections labeled NAME TABLE, PORT TABLE, PARAMETER TABLE and STATIC VAR TABLE, respectively. An example IFS file is given below. The example is followed by detailed descriptions of each of the entries, what they signify, and what values are acceptable for them. Keywords are case insensitive.

```
NAME_TABLE:
C_Function_Name:   ucm_xfer
Spice_Model_Name:  xfer
```

```
Description:           "arbitrary transfer function"
PORT_TABLE:
Port_Name:             in              out
Description:           "input"         "output"
Direction:             in              out
Default_Type:          v               v
Allowed_Types:         [v,vd,i,id]     [v,vd,i,id]
Vector: no no
Vector_Bounds:         -               -
Null_Allowed:          no              no
PARAMETER_TABLE:
Parameter_Name:        in_offset       gain
Description:           "input offset" "gain"
Data_Type:             real            real
Default_Value:         0.0             1.0
Limits:                -               -
Vector:                no              no
Vector_Bounds:         -               -
Null_Allowed:          yes             yes
PARAMETER_TABLE:
Parameter_Name:        num_coeff
Description:           "numerator polynomial coefficients"
Data_Type:             real
Default_Value:         -
Limits:                -
Vector:                yes
Vector_Bounds:         [1 -]
Null_Allowed:          no
PARAMETER_TABLE:
Parameter_Name:        den_coeff
Description:           "denominator polynomial coefficients"
Data_Type:             real
Default_Value:         -
Limits:                -
Vector:                yes
Vector_Bounds:         [1 -]
Null_Allowed:          no
PARAMETER_TABLE:
Parameter_Name:        int_ic
Description:           "integrator stage initial conditions"
Data_Type:             real
Default_Value:         0.0
Limits:                -
Vector:                yes
Vector_Bounds:         den_coeff
Null_Allowed:          yes
STATIC_VAR_TABLE:
Static_Var_Name:       x
Data_Type:             pointer
Description:           "x-coefficient array"
```

27.4.1   The Name Table

The name table is introduced by the "Name_Table:" keyword. It defines the code model's C function name, the name used on a .MODEL card, and an optional textual description. The following sections define the valid fields that may be specified in the Name Table.

C Function Name

The C function name is a valid C identifier which is the name of the function for the code model. It is introduced by the "C_Function_Name:" keyword followed by a valid C identifier. To reduce the chance of name conflicts, it is recommended that user-written code model names use the prefix "ucm_" for this entry. Thus, in the example given above, the model name is "xfer", but the C function is "ucm_xfer". Note that when you subsequently write the model function in the Model Definition File, this name must agree with that of the function (i.e., "ucm_xfer"), or an error will result in the linking step.

SPICE Model Name

The SPICE model name is a valid SPICE identifier that will be used on SPICE .MODEL cards to refer to this code model. It may or may not be the same as the C function name. It is introduced by the "Spice_Model_Name:" keyword followed by a valid SPICE identifier.

Description The description string is used to describe the purpose and function of the code model. It is introduced by the "Description:" keyword followed by a C string literal.

27.4.2 The Port Table

The port table is introduced by the "Port_Table:" keyword. It defines the set of valid ports available to the code model. The following sections define the valid fields that may be specified in the port table.

Port Name

The port name is a valid SPICE identifier. It is introduced by the "Port_Name:" keyword followed by the name of the port. Note that this port name will be used to obtain and return input and output values within the model function. This will be discussed in more detail in the next section.

Description

The description string is used to describe the purpose and function of the code model. It is introduced by the "Description:" keyword followed by a C string literal.

Direction

The direction of a port specifies the dataflow direction through the port. A direction must be one of "in", "out", or "inout". It is introduced by the "Direction:" keyword followed by a valid direction value.

Default Type

The Default_Type field specifies the type of a port. These types are identical to those used to define the port types on a SPICE deck instance card (see Table 13.1), but without the percent sign (%) preceding them. Table 27.1 summarizes the allowable types.

Allowed Types

A port must specify the types it is allowed to assume. An allowed type value must be a list of type names (a blank or comma separated list of names delimited by square brackets, e.g. "[v vd i id]" or "[d]"). The type names must be taken from those listed in Table 27.1.

Vector

A port which is a vector can be thought of as a bus. The Vector field is introduced with the "Vector:" keyword followed by a boolean value: "YES", "TRUE", "NO" or "FALSE".
    The values "YES" and "TRUE" are equivalent and specify that this port is a vector. Likewise, "NO" and "FALSE" specify that the port is not a vector. Vector ports must have a corresponding vector bounds field that specifies valid sizes of the vector port.

Default Types		
Type	Description	Valid Directions
d	digital	in or out
g	conductance (VCCS)	inout
gd	differential conductance (VCCS)	inout
h	resistance (CCVS)	inout
hd	differential resistance (CCVS)	inout
i	current	in or out
id	differential current	in or out
v	voltage	in or out
vd	differential voltage	in or out
<identifier>	user-defined type	in or out

Table 27.1: Port Types

Vector Bounds

If a port is a vector, limits on its size must be specified in the vector bounds field. The Vector Bounds field specifies the upper and lower bounds on the size of a vector. The Vector Bounds field is usually introduced by the "Vector_Bounds:" keyword followed by a range of integers (e.g. "[1 7]" or "[3, 20]"). The lower bound of the vector specifies the minimum number of elements in the vector; the upper bound specifies the maximum number of elements. If the range is unconstrained, or the associated port is not a vector, the vector bounds may be specified by a hyphen ("-"). Using the hyphen convention, partial constraints on the vector bound may be defined (e.g., "[2, -]" indicates that the least number of port elements allowed is two, but there is no maximum number).

Null Allowed

In some cases, it is desirable to permit a port to remain unconnected to any electrical node in a circuit. The Null_Allowed field specifies whether this constitutes an error for a particular port. The Null_Allowed field is introduced by the "Null_Allowed:" keyword and is followed by a boolean constant: "YES", "TRUE", "NO" or "FALSE". The values "YES" and "TRUE" are equivalent and specify that it is legal to leave this port unconnected. "NO" or "FALSE" specify that the port must be connected.

27.4.3   The Parameter Table

The parameter table is introduced by the "Parameter_Table:" keyword. It defines the set of valid parameters available to the code model. The following sections define the valid fields that may be specified in the parameter table.

Parameter Name

The parameter name is a valid SPICE identifier which will be used on SPICE .MODEL cards to refer to this parameter. It is introduced by the "Parameter_Name:" keyword followed by a valid SPICE identifier.

Description

The description string is used to describe the purpose and function of the parameter. It is introduced by the "Description:" keyword followed by a string literal.

Data Type

The parameter's data type is specified by the Data Type field. The Data Type field is introduced by the keyword "Data_Type:" and is followed by a valid data type. Valid data types include boolean, complex, int, real, and string.

Null Allowed

The Null_Allowed field is introduced by the "Null_Allowed:" keyword and is followed by a boolean literal. A value of "TRUE" or "YES" specify that it is valid for the corresponding SPICE .MODEL card to omit a value for this parameter. If the parameter is omitted, the default value is used. If there is no default value, an undefined value is passed to the code model, and the PARAM_NULL( ) macro will return a value of "TRUE" so that defaulting can be handled within the model itself. If the value of Null_Allowed is "FALSE" or "NO", then the simulator will flag an error if the SPICE .MODEL card omits a value for this parameter.

Default Value

If the Null_Allowed field specifies "TRUE" for this parameter, then a default value may be specified. This value is supplied for the parameter in the event that the SPICE .MODEL card does not supply a value for the parameter. The default value must be of the correct type. The Default Value field is introduced by the "Default_Value:" keyword and is followed by a numeric, boolean, complex, or string literal, as appropriate.

Limits

Integer and real parameters may be constrained to accept a limited range of values. The following range syntax is used whenever such a range of values is required. A range is specified by a square bracket followed by a value representing a lower bound separated by space from another value representing an upper bound and terminated with a closing square bracket (e.g."[0 10]"). The lower and upper bounds are inclusive. Either the lower or the upper bound may be replaced by a hyphen ("-") to indicate that the bound is unconstrained (e.g. "[10 -]" is read as "the range of values greater than or equal to 10"). For a totally unconstrained range, a single hyphen with no surrounding brackets may be used. The parameter value limit is introduced by the "Limits:" keyword and is followed by a range.

Vector

The Vector field is used to specify whether a parameter is a vector or a scalar. Like the PORT TABLE Vector field, it is introduced by the "Vector:" keyword and followed by a boolean value. "TRUE" or "YES" specify that the parameter is a vector. "FALSE" or "NO" specify that it is a scalar.

Vector Bounds

The valid sizes for a vector parameter are specified in the same manner as are port sizes (see Section 27.4.2). However, in place of using a numeric range to specify valid vector bounds it is also possible to specify the name of a port. When a parameter's vector bounds are specified in this way, the size of the vector parameter must be the same as the associated vector port.

27.4.4   Static Variable Table

The Static Variable table is introduced by the "Static_Var_Table:" keyword. It defines the set of valid static variables available to the code model. These are variables whose values are retained between successive invocations of the code model by the simulator. The following sections define the valid fields that may be specified in the Static Variable Table.

Name

The Static variable name is a valid C identifier that will be used in the code model to refer to this static variable. It is introduced by the "Static_Var_Name:" keyword followed by a valid C identifier.

Description

The description string is used to describe the purpose and function of the static variable. It is introduced by the "Description:" keyword followed by a string literal.

Data Type

The static variable's data type is specified by the Data Type field. The Data Type field is introduced by the keyword "Data_Type:" and is followed by a valid data type. Valid data types include boolean, complex, int, real, string and pointer.

Note that pointer types are used to specify vector values; in such cases, the allocation of memory for vectors must be handled by the code model through the use of the malloc() or calloc() C function. Such allocation must only occur during the initialization cycle of the model (which is identified in the code model by testing the INIT macro for a value of TRUE). Otherwise, memory will be unnecessarily allocated each time the model is called.

Following is an example of the method used to allocate memory to be referenced by a static pointer variable "x" and subsequently use the allocated memory. The example assumes that the value of "size" is at least 2, else an error would result. The references to `STATIC_VAR(x)` that appear in the example illustrate how to set the value of, and then access, a static variable named "x". In order to use the variable "x" in this manner, it must be declared in the Static Variable Table of the code model's Interface Specification File.

```
/* Define local pointer variable */
double *local.x;

/* Allocate storage to be referenced by the static variable x. */
/* Do this only if this is the initial call of the code model. */
if (INIT == TRUE) {
    STATIC_VAR(x) = calloc(size, sizeof(double));
}

/* Assign the value from the static pointer value to the local */
/* pointer variable. */
local_x = STATIC_VAR(x);

/* Assign values to first two members of the array */
local_x[0] = 1.234;
local_x[1] = 5.678;
```

27.5   Model Definition File

The Model Definition File is a C source code file that defines a code model's behavior given input values which are passed to it by the simulator. The file itself is always given the name "cfunc.mod". In order to allow for maximum flexibility, passing of input, output, and simulator-specific information is handled through accessor macros, which are described below. In addition, certain predefined library functions (e.g. smoothing interpolators, complex arithmetic routines) are included in the simulator in order to ease the burden of the code model programmer. These are also described below.

27.5.1   Macros

The use of the accessor macros is illustrated in the following example. Note that the argument to most accessor macros is the name of a parameter or port as defined in the Interface Specification File. Note also that all accessor macros except "ARGS" resolve to an lvalue (C language terminology for something that can be assigned a value). Accessor macros do not implement expressions or assignments.

```
void code.model(ARGS) /* private structure accessed by
                              accessor macros                    */
{
  /* The following code fragments are intended to show how     */
  /* information in the argument list is accessed. The reader   */
  /* should not attempt to relate one fragment to another.      */
  /* Consider each fragment as a separate example.              */
```

```
double p,/* variable for use in the following code fragments */
      x,   /* variable for use in the following code fragments */
      y;   /* variable for use in the following code fragments */


int i,   /* indexing variable for use in the following     */
      j;   /* indexing variable for use in the following     */

UDN_Example_Type *a_ptr, /* A pointer used to access a User-
                               Defined Node type */
                 *y_ptr; /* A pointer used to access a User-
                               Defined Node type */

/* Initializing and outputting a User-Defined Node result */
if(INIT) {
   OUTPUT(y) = malloc(sizeof(user.defined.struct));
   y_ptr = OUTPUT(y);
   y_ptr->component1 = 0.0;
   y_ptr->component2 = 0.0;
}
else {
   y_ptr = OUTPUT(y);
   y_ptr->component1 = x1;
   y_ptr->component2 = x2;
}

/* Determining analysis type */
if(ANALYSIS == AC) {
    /* Perform AC analysis-dependent operations here */
}

/* Accessing a parameter value from the .model card */
p = PARAM(gain);

/* Accessing a vector parameter from the .model card */
for(i = 0; i < PARAM_SIZE(in_offset); i++)
   p = PARAM(in_offset[i]);

/* Accessing the value of a simple real-valued input */
x = INPUT(a);

/* Accessing a vector input and checking for null port */
if( ! PORT_NULL(a))
   for(i = 0; i < PORT_SIZE(a); i++)
       x = INPUT(a[i]);

/* Accessing a digital input */
x = INPUT(a);

/* Accessing the value of a User-Defined Node input...    */
/* This node type includes two elements in its definition. */
a_ptr = INPUT(a);
x = a_ptr->component1;
y = a_ptr->component2;

/* Outputting a simple real-valued result */
OUTPUT(out1) = 0.0;
```

```
    /* Outputting a vector result and checking for null */
    if( ! PORT_NULL(a))
        for(i = 0; i < PORT.SIZE(a); i++)
            OUTPUT(a[i]) = 0.0;


    /* Outputting the partial of output out1 w.r.t. input a */
    PARTIAL(out1,a) = PARAM(gain);

  /* Outputting the partial of output out2(i) w.r.t. input b(j) */
    for(i = 0; i < PORT_SIZE(out2); i++) {
        for(j = 0; j < PORT_SIZE(b); j++) {
            PARTIAL(out2[i],b[j]) = 0.0;
        }
    }

  /* Outputting gain from input c to output out3 in an
     AC analysis */
    complex_gain_real = 1.0;
    complex_gain_imag = 0.0;
    AC_GAIN(out3,c) = complex_gain;

    /* Outputting a digital result */
    OUTPUT_STATE(out4) = ONE;

    /* Outputting the delay for a digital or user-defined output */
    OUTPUT_DELAY(out5) = 1.0e-9;
}
```

Macro Definitions

The full set of accessor macros is listed below. Arguments shown in parenthesis are examples only.
Explanations of the accessor macros are provided in the subsections below.

```
Circuit Data:
    ARGS
    CALL_TYPE
    INIT
    ANALYSIS
    FIRST_TIMEPOINT
    TIME
    T(n)
    RAD_FREQ
    TEMPERATURE
Parameter Data:
    PARAM(gain)
    PARAM_SIZE(gain)
    PARAM_NULL(gain)
Port Data:
    PORT_SIZE(a)
    PORT_NULL(a)
    LOAD(a)
    TOTAL_LOAD(a)
Input Data:
    INPUT(a)
    INPUT_STATE(a)
    INPUT_STRENGTH(a)
Output Data:
    OUTPUT(y)
```

```
        OUTPUT_CHANGED(a)
        OUTPUT_DELAY(y)
        OUTPUT_STATE(a)
        OUTPUT_STRENGTH(a)
    Partial Derivatives:
        PARTIAL(y,a)
    AC Gains:
        AC_GAIN(y,a)
    Static Variable:
        STATIC_VAR(x)
```

Circuit Data

```
    ARGS
    CALL_TYPE
    INIT
    ANALYSIS
    FIRST_TIMEPOINT
    TIME
    T(n)
    RAD_FREQ
    TEMPERATURE
```

ARGS is a macro which is passed in the argument list of every code model. It is there to provide a way of referencing each model to all of the remaining macro values. It must be present in the argument list of every code model; it must also be the only argument present in the argument list of every code model.

CALL_TYPE is a macro which returns one of two possible symbolic constants. These are EVENT and ANALOG. Testing may be performed by a model using CALL TYPE to determine whether it is being called by the analog simulator or the event-driven simulator. This will, in general, only be of value to a hybrid model such as the adc bridge or the dac bridge.

INIT is an integer (int) that takes the value 1 or 0 depending on whether this is the first call to the code model instance or not, respectively.

ANALYSIS is an enumerated integer that takes values of DC, AC, or TRANSIENT. FIRST TIME-POINT is an integer that takes the value 1 or 0 depending on whether this is the first call for this instance at the current analysis step (i.e., timepoint) or not, respectively.

TIME is a double representing the current analysis time in a transient analysis. $T(n)$ is a double vector giving the analysis time for a specified timepoint in a transient analysis, where n takes the value 0 or 1. $T(0)$ is equal to TIME. $T(1)$ is the last accepted timepoint. $(T(0) - T(1))$ is the timestep (i.e., the delta-time value) associated with the current time.

RAD_FREQ is a double representing the current analysis frequency in an AC analysis expressed in units of radians per second.

TEMPERATURE is a double representing the current analysis temperature.

Parameter Data

```
    PARAM(gain)
    PARAM_SIZE(gain)
    PARAM_NULL(gain)
```

PARAM(gain) resolves to the value of the scalar (i.e., non-vector) parameter "gain" which was defined in the Interface Specification File tables. The type of "gain" is the type given in the ifspec.ifs file. The same accessor macro can be used regardless of type. If "gain" is a string, then PARAM(gain) would resolve to a pointer. PARAM(gain[n]) resolves to the value of the nth element of a vector parameter "gain".

PARAM_SIZE(gain) resolves to an integer (int) representing the size of the "gain" vector (which was dynamically determined when the SPICE deck was read). PARAM_SIZE(gain) is undefined if gain is a scalar.

PARAM_NULL(gain) resolves to an integer with value 0 or 1 depending on whether a value was specified for gain, or whether the value is defaulted, respectively.

Port Data

```
PORT_SIZE(a)
PORT_NULL(a)
LOAD(a)
TOTAL_LOAD(a)
```

PORT_SIZE(a) resolves to an integer (int) representing the size of the "a" port (which was dynamically determined when the SPICE deck was read). PORT_SIZE(a) is undefined if gain is a scalar.

PORT_NULL(a) resolves to an integer (int) with value 0 or 1 depending on whether the SPICE deck has a node specified for this port, or has specified that the port is null, respectively.

LOAD(a) is used in a digital model to post a capacitive load value to a particular input or output port during the INIT pass of the simulator. All values posted for a particular event-driven node using the LOAD() macro are summed, producing a total load value which

TOTAL_LOAD(a) returns a double value which represents the total capacitive load seen on a specified node to which a digital code model is connected. This information may be used after the INIT pass by the code model to modify the delays it posts with its output states and strengths. Note that this macro can also be used by non-digital event-driven code models (see LOAD(), above).

Input Data

```
INPUT(a)
INPUT_STATE(a)
INPUT_STRENGTH(a)
```

INPUT(a) resolves to the value of the scalar input "a" that was defined in the Interface Specification File tables ("a" can be either a scalar port or a port value from a vector; in the latter case, the notation used would be "a[i]", where "i" is the index value for the port). The type of "a" is the type given in the ifspec.ifs file. The same accessor macro can be used regardless of type.

INPUT_STATE(a) resolves to the state value defined for digital node types. These will be one of the symbolic constants ZERO, ONE, or UNKNOWN.

INPUT_STRENGTH(a) resolves to the strength with which a digital input node is being driven. This is determined by a resolution algorithm which looks at all outputs to a node and determines its final driven strength. This value in turn is passed to a code model when requested by this macro. Possible strength values are:
1. STRONG
2. RESISTIVE
3. HI_IMPEDANCE
4. UNDETERMINED

Output Data

```
OUTPUT(y)
OUTPUT_CHANGED(a)
OUTPUT_DELAY(y)
OUTPUT_STATE(a)
OUTPUT_STRENGTH(a)
```

OUTPUT(y) resolves to the value of the scalar output "y" that was defined in the Interface Specification File tables. The type of "y" is the type given in the ifspec.ifs file. The same accessor macro can be used regardless of type. If "y" is a vector, then OUTPUT(y) would resolve to a pointer.

OUTPUT_CHANGED(a) may be assigned one of two values for any particular output from a digital code model. If assigned the value TRUE (the default), then an output state, strength and delay must be posted by the model during the call. If, on the other hand, no change has occurred during that pass, the OUTPUT_CHANGED(a) value for an output can be set to FALSE. In this case, no state, strength or delay values need subsequently be posted by the model. Remember that this macro applies to a single output port. If a model has multiple outputs that have not changed, OUTPUT_CHANGED(a) must be set to FALSE for each of them.

OUTPUT_DELAY(y) may be assigned a double value representing a delay associated with a particular digital or User-Defined Node output port. Note that this macro must be set for each digital or User-Defined Node output from a model during each pass, unless the OUTPUT_CHANGED(a) macro is invoked (see above). Note also that a non-zero value must be assigned to OUTPUT_DELAY(). Assigning a value of zero (or a negative value) will cause an error.

OUTPUT_STATE(a) may be assigned a state value for a digital output node. Valid values are ZERO, ONE, and UNKNOWN. This is the normal way of posting an output state from a digital code model.

OUTPUT_STRENGTH(a) may be assigned a strength value for a digital output node. This is the normal way of posting an output strength from a digital code model. Valid values are:
1. STRONG
2. RESISTIVE
3. HI_IMPEDANCE
4. UNDETERMINED

Partial Derivatives

```
PARTIAL(y,a)
PARTIAL(y[n],a)
PARTIAL(y,a[m])
PARTIAL(y[n],a[m])
```

PARTIAL(y,a) resolves to the value of the partial derivative of scalar output "y" with respect to scalar input "a". The type is always double since partial derivatives are only defined for nodes with real valued quantities (i.e., analog nodes).

The remaining uses of PARTIAL are shown for the cases in which either the output, the input, or both are vectors.

Partial derivatives are required by the simulator to allow it to solve the non-linear equations that describe circuit behavior for analog nodes. Since coding of partial derivatives can become difficult and error-prone for complex analog models, you may wish to consider using the cm_analog_auto_partial() code model support function instead of using this macro.

AC Gains

```
AC_GAIN(y,a)
AC_GAIN(y[n],a)
AC_GAIN(y,a[m])
AC_GAIN(y[n],a[m])
```

AC_GAIN(y,a) resolves to the value of the AC analysis gain of scalar output "y" from scalar input "a". The type is always a structure ("Complex_t") defined in the standard code model header file:

```
typedef struct Complex_s {
double real; /* The real part of the complex number */
double imag; /* The imaginary part of the complex number */
}Complex_t;
```

The remaining uses of AC_GAIN are shown for the cases in which either the output, the input, or both are vectors.

Static Variables

```
STATIC_VAR(x)
```

STATIC_VAR(x) resolves to an lvalue or a pointer which is assigned the value of some scalar code
model result or state defined in the Interface Spec File tables, or a pointer to a value or a vector
of values. The type of "x" is the type given in the Interface Specification File. The same accessor
macro can be used regardless of type since it simply resolves to an lvalue. If "x" is a vector, then
STATIC_VAR(x) would resolve to a pointer. In this case, the code model is responsible for allocat-
ing storage for the vector and assigning the pointer to the allocated storage to STATIC_VAR(x).

Accessor Macros

Table 27.2 describes the accessor macros available to the Model Definition File programmer and their C
types. The PARAM and STATIC_VAR macros, whose types are labeled CD (context dependent), return
the type defined in the Interface Specification File. Arguments listed with "[i]" take an optional square
bracket delimited index if the corresponding port or parameter is a vector. The index may be any C
expression - possibly involving calls to other accessor macros (e.g.," `OUTPUT(out[PORT_SIZE(out)-1])`")

27.5.2   Function Library

Overview

Aside from the accessor macros, the simulator also provides a library of functions callable from within
code models. The header file containing prototypes to these functions is automatically inserted into the
Model Definition File for you. The complete list of available functions follows:

```
Smoothing Functions:
    void cm_smooth_corner
    void cm_smooth_discontinuity
    double cm_smooth_pwl
Model State Storage Functions:
    void cm_analog_alloc
    void cm_event_alloc
    void *cm_analog_get_ptr
    void *cm_event_get_ptr
Integration and Convergence Functions:
    int cm_analog_integrate
    int cm_analog_converge
    void cm_analog_not_converged
    void cm_analog_auto_partial
    double cm_analog_ramp_factor
Message Handling Functions:
    char *cm_message_get_errmsg
    void cm_message_send
Breakpoint Handling Functions:
    int cm_analog_set_temp_bkpt
    int cm_analog_set_perm_bkpt
    int cm_event_queue
Special Purpose Functions:
    void cm_climit_fcn
    double cm_netlist_get_c
    double cm_netlist_get_l
Complex Math Functions:
    complex_t cm_complex_set
    complex_t cm_complex_add
    complex_t cm_complex_sub
    complex_t cm_complex_mult
    complex_t cm_complex_div
```

Name	Type	Args	Description
AC_GAIN	Complex_t	y[i],x[i]	AC gain of output y with respect to input x.
ANALYSIS	enum	<none>	Type of analysis: DC, AC, TRANSIENT.
ARGS	Mif_Private_t	<none>	Standard argument to all code model function.
CALL_TYPE	enum	<none>	Type of model evaluation call: ANALOG or EVENT.
INIT	Boolean_t	<none>	Is this the first call to the model?
INPUT	double or void*	name[i]	Value of analog input port, or value of structure pointer for User-Defined Node port.
INPUT_STATE	enum	name[i]	State of a digital input: ZERO, ONE, or UNKNOWN.
INPUT_STRENGHT	enum	name[i]	Strength of digital input: STRONG, RESISTIVE, HI IMPEDANCE, or UNDETERMINED.
INPUT_TYPE	char*	name[i]	The port type of the input.
LOAD	double	name[i]	The digital load value placed on a port by this model.
MESSAGE	char*	name[i]	A message output by a model on an event-driven node.
OUTPUT	double or void*	name[i]	Value of the analog output port or value of structure pointer for User-Defined Node port.
OUTPUT_CHANGED	Boolean_t	name[i]	Has a new value been assigned to this event-driven output by the model?
OUTPUT_DELAY	double	name[i]	Delay in seconds for an event-driven output.
OUTPUT_STATE	enum	name[i]	State of a digital output: ZERO, ONE, or UNKNOWN.
OUTPUT_STRENGTH	enum	name[i]	Strength of digital output: STRONG, RESISTIVE, HI_IMPEDANCE, or UNDETERMINED.
OUTPUT_TYPE	char*	name[i]	The port type of the output.
PARAM	CD	name[i]	Value of the parameter.
PARAM_NULL	Boolean_t	name[i]	Was the parameter not included on the SPICE .model card ?
PARAM_SIZE	int	name	Size of parameter vector.
PARTIAL	double	y[i],x[i]	Partial derivative of output y with respect to input x.
PORT_NULL	Mif_Boolean_t	name	Has this port been specified as unconnected?
PORT_SIZE	int	name	Size of port vector.
RAD_FREQ	double	<none>	Current analysis frequency in radians per second.
STATIC_VAR	CD	name	Value of a static variable.
STATIC_VAR_SIZE	int	name	Size of static var vector (currently unused).
T(n)	int	index	Current and previous analysis times (T(0) = TIME = current analysis time, T(1) = previous analysis time).
TEMPERATURE	double	<none>	Current analysis temperature.
TIME	double	<none>	Current analysis time (same as T(0)).
TOTAL_LOAD	double	name[i]	The total of all loads on the node attached to this event driven port.

Table 27.2: Accessor Macros

Smoothing Functions

```
    void
    cm_smooth_corner(x_input, x_center, y_center, domain,
                     lower_slope, upper_slope, y_output, dy_dx)

        double x_input;      /* The value of the x input */
        double x_center;     /* The x intercept of the two slopes */
        double y_center;     /* The y intercept of the two slopes */
        double domain;       /* The smoothing domain */
        double lower_slope;  /* The lower slope */
        double upper_slope;  /* The upper slope */
        double *y_output;    /* The smoothed y output */
        double *dy_dx;       /* The partial of y wrt x */


    void
    cm_smooth_discontinuity(x_input, x_lower, y_lower, x_upper, y_upper
                            y_output, dy_dx)

        double x_input;    /* The x value at which to compute y */
        double x_lower;    /* The x value of the lower corner */
        double y_lower;    /* The y value of the lower corner */
        double x_upper;    /* The x value of the upper corner */
        double y_upper;    /* The y value of the upper corner */
        double *y_output;  /* The computed smoothed y value */
        double *dy_dx;     /* The partial of y wrt x */


    double
    cm_smooth_pwl(x_input, x, y, size, input_domain, dout_din)

        double x_input;       /* The x input value */
        double *x;            /* The vector of x values */
        double *y;            /* The vector of y values */
        int size;             /* The size of the xy vectors */
        double input_domain;  /* The smoothing domain */
        double *dout_din;     /* The partial of the output wrt the input */
```

cm_smooth_corner() automates smoothing between two arbitrarily-sloped lines that meet at a single center point. You specify the center point (x_center, y_center), plus a domain (x-valued delta) above and below x_center. This defines a smoothing region about the center point. Then, the slopes of the meeting lines outside of this smoothing region are specified (lower_slope, upper_slope). The function then interpolates a smoothly-varying output (*y_output) and its derivative (*dy_dx) for the x_input value. This function helps to automate the smoothing of piecewise-linear functions, for example. Such smoothing aids the simulator in achieving convergence.

cm_smooth_discontinuity() allows you to obtain a smoothly-transitioning output (*y_output) that varies between two static values (y_lower, y_upper) as an independent variable (x_input) transitions between two values (x_lower, x_upper). This function is useful in interpolating between resistances or voltage levels that change abruptly between two values.

cm_smooth_pwl() duplicates much of the functionality of the predefined pwl code model. The cm smooth pwl() takes an input value plus x-coordinate and y-coordinate vector values along with the total number of coordinate points used to describe the piecewise linear transfer function and returns the interpolated or extrapolated value of the output based on that transfer function. More detail is available by looking at the description of the pwl code model. Note that the output value is the function's returned value.

Model State Storage Functions

```
    void cm_analog_alloc(tag, size)
```

```
     int tag;  /* The user-specified tag for this block of memory */
     int size; /* The number of bytes to allocate */

void cm_event_alloc(tag, size)

     int tag; /* The user-specified tag for the memory block */
     int size; /* The number of bytes to be allocated */

void *cm_analog_get_ptr(tag, timepoint)

     int tag; /* The user-specified tag for this block of memory */
     int timepoint; /* The timepoint of interest - 0=current 1=previous */

void *cm_event_get_ptr(tag, timepoint)

     int tag; /* The user-specified tag for the memory block */
     int timepoint; /* The timepoint - 0=current, 1=previous */
```

cm_analog_alloc() and **cm_event_alloc()** allow you to allocate storage space for analog and event-driven model state information. The storage space is not static, but rather represents a storage vector of two values which rotate with each accepted simulator timepoint evaluation. This is explained more fully below. The "tag" parameter allows you to specify an integer tag when allocating space. This allows more than one rotational storage location per model to be allocated. The "size" parameter specifies the size in bytes of the storage (computed by the C language "sizeof()" operator). Both cm_analog_alloc() and cm_event_alloc() will **not** return pointers to the allocated space, as has been available (and buggy) from the original XSPICE code. cm_analog_alloc() should be used by an analog model; cm_event_alloc() should be used by an event-driven model.

*cm_analog_get_ptr() and ***cm_event_get_ptr()** retrieve the pointer location of the rotational storage space previously allocated by cm_analog_alloc() or cm_event_alloc(). **Important notice:** These functions must be called only after **all** memory allocation (all calls to cm_analog_alloc() or cm_event_alloc()) have been done. All pointers returned between calls to memory allocation will become obsolete (point to freed memory because of an internal realloc). The functions take the integer "tag" used to allocate the space, and an integer from 0 to 1 which specifies the timepoint with which the desired state variable is associated (e.g. timepoint = 0 will retrieve the address of storage for the current timepoint; timepoint = 1 will retrieve the address of storage for the last accepted timepoint). **Note that once a model is exited, storage to the current timepoint state storage location (i.e., timepoint = 0) will, upon the next timepoint iteration, be rotated to the previous location (i.e., timepoint = 1).** When rotation is done, a copy of the old "timepoint = 0" storage value is placed in the new "timepoint = 0" storage location. Thus, if a value does not change for a particular iteration, specific writing to "timepoint = 0" storage is not required. These features allow a model coder to constantly know which piece of state information is being dealt with within the model function at each timepoint.

Integration and Convergence Functions

```
int cm_analog_integrate(integrand, integral, partial)

     double integrand; /* The integrand */
     double *integral; /* The current and returned value of integral */
     double *partial;  /* The partial derivative of integral wrt integrand */

int cm_analog_converge(state)

     double *state; /* The state to be converged */

void cm_analog_not_converged()
void cm_analog_auto_partial()

double cm_ramp_factor()
```

cm_analog_integrate() takes as input the integrand (the input to the integrator) and produces as output the integral value and the partial of the integral with respect to the integrand. The integration itself is with respect to time, and the pointer to the integral value must have been previously allocated using cm_analog_alloc() and *cm_analog_get_ptr(). This is required because of the need for the integrate routine itself to have access to previously-computed values of the integral.

cm_analog_converge() takes as an input the address of a state variable that was previously allocated using cm_analog_alloc() and *cm_analog_get_ptr(). The function itself serves to notify the simulator that for each timestep taken, that variable must be iterated upon until it converges.

cm_analog_not_converged() is a function that can and should be called by an analog model whenever it performs internal limiting of one or more of its inputs to aid in reaching convergence. This causes the simulator to call the model again at the current timepoint and continue solving the circuit matrix. A new timepoint will not be attempted until the code model returns without calling the cm_analog_not_converged() function. For circuits which have trouble reaching a converged state (often due to multiple inputs changing too quickly for the model to react in a reasonable fashion), the use of this function is virtually mandatory.

cm_analog_auto_partial() may be called at the end of a code model function in lieu of calculating the values of partial derivatives explicitly in the function. When this function is called, no values should be assigned to the PARTIAL macro since these values will be computed automatically by the simulator. The automatic calculation of partial derivatives can save considerable time in designing and coding a model, since manual computation of partial derivatives can become very complex and error-prone for some models. However, the automatic evaluation may also increase simulation run time significantly. Function cm_analog_auto_partial() causes the model to be called N additional times (for a model with N inputs) with each input varied by a small amount (1e-6 for voltage inputs and 1e-12 for current inputs). The values of the partial derivatives of the outputs with respect to the inputs are then approximated by the simulator through divided difference calculations.

cm_analog_ramp_factor() will then return a value from 0.0 to 1.0, which indicates whether or not a ramp time value requested in the SPICE analysis deck (with the use of .option ramptime=<duration>) has elapsed. If the RAMPTIME option is used, then cm_analog_ramp_factor returns a 0.0 value during the DC operating point solution and a value which is between 0.0 and 1.0 during the ramp. A 1.0 value is returned after the ramp is over or if the RAMPTIME option is not used. This value is intended as a multiplication factor to be used with all model outputs which would ordinarily experience a "power-up" transition. Currently, all sources within the simulator are automatically ramped to the "final" time-zero value if a RAMPTIME option is specified.

Message Handling Functions

```
char *cm_message_get_errmsg()
int cm_message_send(char *msg)
char *msg; /* The message to output. */
```

*cm_message_get_errmsg() is a function designed to be used with other library functions to provide a way for models to handle error situations. More specifically, whenever a library function which returns type "int" is executed from a model, it will return an integer value, n. If this value is not equal to zero (0), then an error condition has occurred (likewise, functions which return pointers will return a NULL value if an error has occurred). At that point, the model can invoke *cm_message_get_errmsg to obtain a pointer to an error message. This can then in turn be displayed to the user or passed to the simulator interface through the cm_message_send() function. The C code required for this is as follows:

```
err = cm_analog_integrate(in, &out, &dout_din);
if (err) {
    cm_message_send(cm_message_get_errmsg());
}
else { ...
```

cm_message_send() sends messages to either the standard output screen or to the simulator interface, depending on which is in use.

Breakpoint Handling Functions

```
int cm_analog_set_perm_bkpt(time)
```

```
        double time; /* The time of the breakpoint to be set */

    int cm_analog_set_temp_bkpt(time)

        double time; /* The time of the breakpoint to be set */

    int cm_event_queue(time)

        double time; /* The time of the event to be queued */
```

cm_analog_set_perm_bkpt() takes as input a time value. This value is posted to the analog simulator algorithm and is used to force the simulator to choose that value as a breakpoint at some time in the future. The simulator may choose as the next timepoint a value less than the input, but not greater. Also, regardless of how many timepoints pass before the breakpoint is reached, it will not be removed from posting. Thus, a breakpoint is guaranteed at the passed time value. Note that a breakpoint may also be set for a time prior to the current time, but this will result in an error if the posted breakpoint is prior to the last accepted time (i.e., T(1)).

cm_analog_set_temp_bkpt() takes as input a time value. This value is posted to the simulator and is used to force the simulator, for the next timestep only, to not exceed the passed time value. The simulator may choose as the next timepoint a value less than the input, but not greater. In addition, once the next timestep is chosen, the posted value is removed regardless of whether it caused the break at the given timepoint. This function is useful in the event that a timepoint needs to be retracted after its first posting in order to recalculate a new breakpoint based on new input data (for controlled oscillators, controlled one-shots, etc), since temporary breakpoints automatically "go away" if not reposted each timestep. Note that a breakpoint may also be set for a time prior to the current time, but this will result in an error if the posted breakpoint is prior to the last accepted time (i.e., T(1)).

cm_event_queue() is similar to cm_analog_set_perm_bkpt(), but functions with event-driven models. When invoked, this function causes the model to be queued for calling at the specified time. All other details applicable to cm_analog_set_perm_bkpt() apply to this function as well.

Special Purpose Functions

```
    void
    cm_climit_fcn(in, in_offset, cntl_upper, cntl_lower, lower_delta, upper_delta,
                  limit_range, gain, fraction, out_final, pout_pin_final,
                  pout_pcntl_lower_final, pout_pcntl_upper_final)

        double in;                      /* The input value */
        double in-offset;               /* The input offset */
        double cntl_upper;              /* The upper control input value */
        double cntl_lower;              /* The lower control input value */
        double lower_delta;             /* The delta from control to limit value */
        double upper_delta;             /* The delta from control to limit value */
        double limit_range;             /* The limiting range */
        double gain;                    /* The gain from input to output */
        int percent;                    /* The fraction vs. absolute range flag */
        double *out_final;              /* The output value */
        double *pout_pin_final;         /* The partial of output wrt input */
        double *pout_pcntl_lower_final; /* The partial of output wrt lower control input */
        double *pout_pcntl_upper:final; /* The partial of output wrt upper control input */

    double cm_netlist_get_c()

    double cm_netlist_get_l()
```

cm_climit_fcn() is a very specific function that mimics the behavior of the climit code model (see the Predefined Models section). In brief, the cm_climit_fcn() takes as input an "in" value, an offset, and controlling upper and lower values. Parameter values include delta values for the controlling inputs, a

smoothing range, gain, and fraction switch values. Outputs include the final value, plus the partial deriva-
tives of the output with respect to signal input, and both control inputs. These all operate identically to
the similarly-named inputs and parameters of the climit model.

The function performs a limit on the "in" value, holding it to within some delta of the controlling
inputs, and handling smoothing, etc. The cm_climit_fcn() was originally used in the ilimit code model
to handle much of the primary limiting in that model, and can be used by a code model developer to
take care of limiting in larger models that require it. See the detailed description of the climit model for
more in-depth description.

cm_netlist_get_c() and **cm_netlist_get_l()** functions search the analog circuitry to which
their input is connected, and total the capacitance or inductance, respectively, found at that node. The
functions, as they are currently written, assume they are called by a model which has only one single-ended
analog input port.

Complex Math Functions

```
    Complex_t cm_complex_set (real_part, imag_part)

        double real_part; /* The real part of the complex number */
        double imag_part; /* The imaginary part of the complex number */


    Complex_t cm_complex_add (x, y)

        Complex_t x; /* The first operand of x + y */
        Complex_t y; /* The second operand of x + y */


    Complex_t cm_complex_sub (x, y)

        Complex_t x; /* The first operand of x - y */
        Complex_t y; /* The second operand of x - y */


    Complex_t cm_complex_mult (x, y)

        Complex_t x; /* The first operand of x * y */
        Complex_t y; /* The second operand of x * y */


    Complex_t cm_complex_div (x, y)

        Complex_t x; /* The first operand of x / y */
        Complex_t y; /* The second operand of x / y */
```

cm_complex_set() takes as input two doubles, and converts these to a Complex_t. The first double
is taken as the real part, and the second is taken as the imaginary part of the resulting complex value.

cm_complex_add(), **cm_complex_sub()**, **cm_complex_mult()**, and **cm_complex_div()**
each take two complex values as inputs and return the result of a complex addition, subtraction, multi-
plication, or division, respectively.

27.6   User-Defined Node Definition File

The User-Defined Node Definition File (udnfunc.c) defines the C functions which implement basic op-
erations on user-defined nodes such as data structure creation, initialization, copying, and comparison.
Unlike the Model Definition File which uses the Code Model Preprocessor to translate Accessor Macros,
the User-Defined Node Definition file is a pure C language file. This file uses macros to isolate you
from data structure definitions, but the macros are defined in a standard header file (EVTudn.h), and
translations are performed by the standard C Preprocessor.

When a directory is created for a new User-Defined Node with 'mkudndir', a structure of type
'Evt_Udn_Info_t' is placed at the bottom of the User-Defined Node Definition File.

This structure contains the type name for the node, a description string, and pointers to each of the
functions that define the node. This structure is complete except for a text string that describes the node
type. This string is stubbed out and may be edited by you if desired.

Name	Type	Description
MALLOCED_PTR	void *	Assign pointer to alloced structure to this macro
STRUCT_PTR	void *	A pointer to a structure of the defined type
STRUCT_PTR_1	void *	A pointer to a structure of the defined type
STRUCT_PTR_2	void *	A pointer to a structure of the defined type
EQUAL	Mif_Boolean_t	Assign TRUE or FALSE to this macro according to the results of structure comparison
INPUT_STRUCT_PTR	void *	A pointer to a structure of the defined type
OUTPUT_STRUCT_PTR	void *	A pointer to a structure of the defined type
INPUT_STRUCT_PTR_ARRAY	void **	An array of pointers to structures of the defined type
INPUT_STRUCT_PTR_ARRAY_SIZE	int	The size of the array
STRUCT_MEMBER_ID	char *	A string naming some part of the structure
PLOT_VAL	double	The value of the specified structure member for plotting purposes
PRINT_VAL	char *	The value of the specified structure member for printing purposes

Table 27.3: User-Defined Node Macros

27.6.1 Macros

You must code the functions described in the following section using the macros appropriate for the particular function. You may elect whether not to provide the optional functions.

It is an error to use a macro not defined for a function. Note that a review of the sample directories for the "real" and "int" UDN types will make the function usage clearer.

The macros used in the User-Defined Node Definition File to access and assign data values are defined in Table 27.3. The translations of the macros and of macros used in the function argument lists are defined in the document Interface Design Document for the XSPICE Simulator of the Automatic Test Equipment Software Support Environment (ATESSE).

27.6.2 Function Library

The functions (required and optional) that define a User-Defined Node are listed below. For optional functions, the function "stub" can be deleted from the udnfunc.c file template created by "mkudndir", and the pointer in the Evt_Udn_Info_t structure can be changed to NULL.

Required functions:

```
create      Allocate data structure used as inputs and outputs to
            code models.

initialize  Set structure to appropriate initial value for first use as
            model input.

copy        Make a copy of the contents into created but possibly
            uninitialized structure.

compare     Determine if two structures are equal in value.
```

Optional functions:

```
dismantle   Free allocations inside structure (but not structure itself).

invert      Invert logical value of structure.

resolve     Determine the resultant when multiple outputs are connected
            to a node.

plot_val    Output a real value for specified structure component for
            plotting purposes.

print_val   Output a string value for specified structure component for
            printing.

ipc_val     Output a binary representation of the structure suitable
            for sending over the IPC channel.
```

The required actions for each of these functions are described in the following subsections. In each function, "mkudndir" replaces the XXX with the node type name specified by you when mkudndir is invoked. The macros used in implementing the functions are described in a later section.

Function udn_XXX_create

Allocate space for the data structure defined for the User-Defined Node to pass data between models. Then assign pointer created by the storage allocator (e.g. malloc) to MALLOCED_PTR.

Function udn_XXX_initialize

Assign STRUCT_PTR to a pointer variable of defined type and then initialize the value of the structure.

Function udn_XXX_compare

Assign STRUCT_PTR_1 and STRUCT_PTR_2 to pointer variables of the defined type. Compare the two structures and assign either TRUE or FALSE to EQUAL.

Function udn_XXX_copy

Assign INPUT_STRUCT_PTR and OUTPUT_STRUCT_PTR to pointer variables of the defined type and then copy the elements of the input structure to the output structure.

Function udn_XXX_dismantle

Assign STRUCT_PTR to a pointer variable of defined type and then free any allocated substructures (but not the structure itself!). If there are no substructures, the body of this function may be left null.

Function udn_XXX_invert

Assign STRUCT_PTR to a pointer variable of the defined type, and then invert the logical value of the structure.

Function udn XXX resolve

Assign INPUT_STRUCT_PTR_ARRAY to a variable declared as an array of pointers of the defined type - e.g.:

```
<type> **struct_array;
struct_array = INPUT_STRUCT_PTR_ARRAY;
```

Then, the number of elements in the array may be determined from the integer valued INPUT_STRUCT_PTR_ARRAY_ macro.
    Assign OUTPUT_STRUCT_PTR to a pointer variable of the defined type. Scan through the array of structures, compute the resolved value, and assign it into the output structure.

Function udn_XXX_plot_val

Assign STRUCT_PTR to a pointer variable of the defined type. Then, access the member of the structure specified by the string in STRUCT_MEMBER_ID and assign some real valued quantity for this member to PLOT_VALUE.

Function udn_XXX_print_val

Assign STRUCT_PTR to a pointer variable of the defined type. Then, access the member of the structure specified by the string in STRUCT_MEMBER_ID and assign some string valued quantity for this member to PRINT_VALUE.

If the string is not static, a new string should be allocated on each call. Do not free the allocated strings.

Function udn XXX ipc val

Use STRUCT_PTR to access the value of the node data. Assign to IPC_VAL a binary representation of the data. Typically this can be accomplished by simply assigning STRUCT_PTR to IPC_VAL.

Assign to IPC_VAL_SIZE an integer representing the size of the binary data in bytes.

27.6.3 Example UDN Definition File

The following is an example UDN Definition File which is included with the XSPICE system. It illustrates the definition of the functions described above for a User-Defined Node type which is included with the XSPICE system: in this case, the "int" (for "integer") node type.

```
#include "EVTudn.h"

void *malloc(unsigned);

/* ----------------------------------------------------------- */

void udn_int_create(CREATE_ARGS)
{
    /* Malloc space for an int */
    MALLOCED_PTR = malloc(sizeof(int));
}

/* ----------------------------------------------------------- */

void udn_int_dismantle(DISMANTLE_ARGS)
{
    /* Do nothing. There are no internally malloc'ed things to dismantle */
}

/* ----------------------------------------------------------- */

void udn_int_initialize(INITIALIZE_ARGS)
{
    int *int_struct = STRUCT_PTR;

    /* Initialize to zero */
    *int.struct = 0;
}

/* ----------------------------------------------------------- */

void udn_int_invert(INVERT_ARGS)
{
```

```
        int *int_struct = STRUCT_PTR;

        /* Invert the state */
        *int_struct = -(*int.struct);
    }

/* ------------------------------------------------------------ */

void udn_int_copy(COPY_ARGS)
{
        int *int_from_struct = INPUT_STRUCT_PTR;
        int *int_to_struct   = OUTPUT_STRUCT_PTR;

        /* Copy the structure */
        *int_to_struct = *int_from_struct;
    }

/* ------------------------------------------------------------ */

void udn_int_resolve(RESOLVE_ARGS)
{
        int **array     = INPUT_STRUCT_PTR_ARRAY;
        int *out        = OUTPUT_STRUCT_PTR;
        int num.struct = INPUT_STRUCT_PTR_ARRAY_SIZE;
        int sum;
        int i;

        /* Sum the values */
        for(i = 0, sum = 0; i ! num_struct; i++)
            sum += *(array[i]);

        /* Assign the result */
        *out = sum;
    }

/* ------------------------------------------------------------ */

void udn_int_compare(COMPARE_ARGS)
{
        int *int_struct1 = STRUCT_PTR_1;
        int *int_struct2 = STRUCT_PTR_2;

        /* Compare the structures */
        if((*int_struct1) == (*int_struct2))
            EQUAL = TRUE;
        else
            EQUAL = FALSE;
    }

/* ------------------------------------------------------------ */

void udn_int_plot_val(PLOT_VAL_ARGS)
{
        int *int_struct = STRUCT_PTR;

        /* Output a value for the int struct */
        PLOT_VAL = *int_struct;
    }
```

```
/* ---------------------------------------------------------- */

void udn_int_print_val(PRINT_VAL_ARGS)
{
    int *int_struct = STRUCT_PTR;

    /* Allocate space for the printed value */
    PRINT_VAL = malloc(30);

    /* Print the value into the string */
    sprintf(PRINT_VAL, "%8d", *int_struct);
}

/* ---------------------------------------------------------- */

void udn_int_ipc_val(IPC_VAL_ARGS)
{
    /* Simply return the structure and its size */
    IPC_VAL = STRUCT_PTR;
    IPC_VAL_SIZE = sizeof(int);
}

/* ---------------------------------------------------------- */

Evt_Udn_Info_t udn_int_info = {

    "int",
    "integer valued data",

    udn_int_create,
    udn_int_dismantle,
    udn_int_initialize,
    udn_int_invert,
    udn_int_copy,
    udn_int_resolve,
    udn_int_compare,
    udn_int_plot_val,
    udn_int_print_val,
    udn_int_ipc_val

};
```

Chapter 28

Error Messages

Error messages may be subdivided into three categories. These are:

1. Error messages generated during the development of a code model (Preprocessor Error Messages).

2. Error messages generated by the simulator during a simulation run (Simulator Error Messages).

3. Error messages generated by individual code models (Code Model Error Messages).

These messages will be explained in detail in the following subsections.

28.1   Preprocessor Error Messages

The following is a list of error messages that may be encountered when invoking the directory-creation and code modeling preprocessor tools. These are listed individually, and explanations follow the name/listing.

    Usage: cmpp [-ifs] [-mod [<filename>]] [-lst]

The Code Model Preprocessor (cmpp) command was invoked incorrectly.

    ERROR - Too few arguments

The Code Model Preprocessor (cmpp) command was invoked with too few arguments.

    ERROR - Too many arguments

The Code Model Preprocessor (cmpp) command was invoked with too many arguments.

    ERROR - Unrecognized argument

The Code Model Preprocessor (cmpp) command was invoked with an invalid argument.

    ERROR - File not found: s<filename>

The specified file was not found, or could not be opened for read access.

    ERROR - Line <line number> of <filename> exceeds XX characters

The specified line was too long.

    ERROR - Pathname on line <line number> of <filename>
            exceeds XX characters.

The specified line was too long.

    ERROR - No pathnames found in file: <filename>

The indicated modpath.lst file does not have pathnames properly listed.

```
ERROR - Problems reading ifspec.ifs in directory <pathname>
```

The Interface Specification File (ifspec.ifs) for the code model could not be read.

```
ERROR - Model name <model name> is same as internal SPICE model name
```

A model has been given the same name as an intrinsic SPICE device.

```
ERROR - Model name '<model name>' in directory: <pathname>
        is same as
        model name '<model name>' in directory: <pathname>
```

Two models in different directories have the same name.

```
ERROR - C function name '<function name>' in directory: <pathname>,
        is same as
        C function name '<function name>' in directory: <pathname>
```

Two C language functions in separate model directories have the same names; these would cause a collision when linking the final executable.

```
ERROR - Problems opening CMextrn.h for write
```

The temporary file CMextern.h used in building the XSPICE simulator executable could not be created or opened. Check permissions on directory.

```
ERROR - Problems opening CMinfo.h for write
```

The temporary file CMinfo.h used in building the XSPICE simulator executable could not be created or opened. Check permissions on directory.

```
ERROR - Problems opening objects.inc file for write
```

The temporary file objects.inc used in building the XSPICE simulator executable could not be created or opened. Check permissions on directory.

```
ERROR - Could not open input .mod file: <filename>
```

The Model Definition File that contains the definition of the Code Model's behavior (usually cfunc.mod) was not found or could not be read.

```
ERROR - Could not open output .c: <filename>
```

The indicated C language file that the preprocessor creates could not be created or opened. Check permissions on directory.

```
Error parsing .mod file: <filename>
```

Problems were encountered by the preprocessor in interpreting the indicated Model Definition File.

```
ERROR - File not found: <filename>
```

The indicated file was not found or could not be opened.

```
Error parsing interface specification file
```

Problems were encountered by the preprocessor in interpreting the indicated Interface Specification File.

```
ERROR - Can't create file: <filename>
```

The indicated file could not be created or opened. Check permissions on directory.

```
ERROR - write.port.info() - Number of allowed types cannot be zero
```

There must be at least one port type specified in the list of allowed types.

```
illegal quoted character in string (expected "\" or "\\")
```

A string was found with an illegal quoted character in it.

```
unterminated string literal
```

A string was found that was not terminated.

```
Unterminated comment
```

A comment was found that was not terminated.

```
Port '<port name>' not found
```

The indicated port name was not found in the Interface Specification File.

```
Port type 'vnam' is only valid for 'in' ports
```

The port type 'vnam' was used for a port with direction 'out' or 'inout'. This type is only allowed on 'in' ports.

```
Port types 'g', 'gd', 'h', 'hd' are only valid for 'inout' ports
```

Port type 'g', 'gd', 'h', or 'hd' was used for a port with direction 'out' or 'in'. These types are only allowed on 'inout' ports.

```
Invalid parameter type - POINTER type valid only for STATIC_VARs
```

The type POINTER was used in a section of the Interface Specification file other than the STATIC_VAR section.

```
Port default type is not an allowed type
```

A default type was specified that is not one of the allowed types for the port.

```
Incompatible port types in 'allowed_types' clause
```

Port types listed under 'Allowed_Types' in the Interface Specification File must all have the same underlying data type. It is illegal to mix analog and eventdriven types in a list of allowed types.

```
Invalid parameter type (saw <parameter type 1> - expected <parameter type 2>)
```

A parameter value was not compatible with the specified type for the parameter.

```
Named range not allowed for limits
```

A name was found where numeric limits were expected.

```
Direction of port '<port number>' in <port name>()
 is not <IN or OUT> or INOUT
```

A problem exists with the direction of one of the elements of a port vector.

```
Port '<port name>' is an array - subscript required
```

A port was referenced that is specified as an array (vector) in the Interface Specification File. A subscript is required (e.g. myport[i])

```
Parameter '<parameter name>' is an array - subscript required
```

A parameter was referenced that is specified as an array (vector) in the Interface Specification File. A subscript is required (e.g. myparam[i])

```
Port '<port name>' is not an array - subscript prohibited
```

A port was referenced that is not specified as an array (vector) in the Interface Specification File.   A subscript is not allowed.

```
    Parameter '<parameter name>' is not an array - subscript prohibited
```

A parameter was referenced that is not specified as an array (vector) in the Interface Specification File. A subscript is not allowed.


```
    Static variable '<static variable name>' is not an array - subscript prohibited
```

Array static variables are not supported. Use a POINTER type for the static variable.

```
    Buffer overflow - try reducing the complexity of CM-macro array subscripts
```

The argument to a code model accessor macro was too long.

```
    Unmatched )
```

An open (was found with no corresponding closing).

```
    Unmatched ]
```

An open [was found with no corresponding closing].

28.2   Simulator Error Messages

The following is a list of error messages that may be encountered while attempting to run a simulation (with the exception of those error messages generated by individual code models). Most of these errors are generated by the simulator while attempting to parse a SPICE deck. These are listed individually, and explanations follow the name/listing.

```
    ERROR - Scalar port expected, [ found
```

A scalar connection was expected for a particular port on the code model, but the symbol [which is used to begin a vector connection list was found.

```
    ERROR - Unexpected ]
```

A] was found where not expected. Most likely caused by a missing [.

```
    ERROR - Unexpected [ - Arrays of arrays not allowed
```

A [character was found within an array list already begun with another [character.

```
    ERROR - Tilde not allowed on analog nodes
```

The tilde character ~was found on an analog connection. This symbol, which performs state inversion, is only allowed on digital nodes and on User-Defined Nodes only if the node type definition allows it.

```
    ERROR - Not enough ports
```

An insufficient number of node connections was supplied on the instance line.   Check the Interface Specification File for the model to determine the required connections and their types.

```
    ERROR - Expected node/instance identifier
```

A special token (e.g. [ ] < > ...) was found when not expected.

```
    ERROR - Expected node identifier
```

A special token (e.g. [ ] < > ...) was found when not expected.

```
    ERROR - unable to find definition of model <name>
```

A .model line for the referenced model was not found.

```
ERROR - model: %s - Array parameter expected - No array delimiter found
```

An array (vector) parameter was expected on the .model card, but enclosing [ ] characters were not found to delimit its values.

```
ERROR - model: %s - Unexpected end of model card
```

The end of the indicated .model line was reached before all required information was supplied.

```
ERROR - model: %s - Array parameter must have at least one value
```

An array parameter was encountered that had no values.

```
ERROR - model: %s - Bad boolean value
```

A bad values was supplied for a Boolean. Value used must be TRUE, FALSE, T, or F.

```
ERROR - model: %s - Bad integer, octal, or hex value
```

A badly formed integer value was found.

```
ERROR - model: %s - Bad real value
```

A badly formed real value was found.

```
ERROR - model: %s - Bad complex value
```

A badly formed complex number was found. Complex numbers must be enclosed in $<\ >$ delimiters.

28.3 Code Model Error Messages

The following is a list of error messages that may be encountered while attempting to run a simulation with certain code models. These are listed alphabetically based on the name of the code model, and explanations follow the name and listing.

28.3.1 Code Model aswitch

```
cntl_error:
    *****ERROR*****
    ASWITCH: CONTROL voltage delta less than 1.0e-12
```

This message occurs as a result of the cntl_off and cntl_on values--with-editline=yes being less than 1.0e-12 volts/amperes apart.

28.3.2 Code Model climit

```
climit_range_error:
    **** ERROR ****
    * CLIMIT function linear range less than zero. *
```

This message occurs whenever the difference between the upper and lower control input values are close enough that there is no effective room for proper limiting to occur; this indicates an error in the control input values.

28.3.3   Code Model core

```
allocation_error:
    ***ERROR***
    CORE: Allocation calloc failed!
```

This message is a generic message related to allocating sufficient storage for the H and B array values.

```
limit_error:
    ***ERROR***
    CORE: Violation of 50% rule in breakpoints!
```

This message occurs whenever the input domain value is an absolute value and the H coordinate points are spaced too closely together (overlap of the smoothing regions will occur unless the H values are redefined).

28.3.4   Code Model d_osc

```
d_osc_allocation_error:
    **** Error ****
    D_OSC: Error allocating VCO block storage
```

Generic block storage allocation error.

```
d_osc_array_error:
    **** Error ****
    D_OSC: Size of control array different than frequency array
```

Error occurs when there is a different number of control array members than frequency array members.

```
d_osc_negative_freq_error:
    **** Error ****
    D_OSC: The extrapolated value for frequency
    has been found to be negative...
     Lower frequency level has been clamped to 0.0 Hz.
```

Occurs whenever a control voltage is input to a model which would ordinarily (given the specified control/freq coordinate points) cause that model to attempt to generate an output oscillating at zero frequency. In this case, the output will be clamped to some DC value until the control voltage returns to a more reasonable value.

28.3.5   Code Model d_source

```
loading_error:
    ***ERROR***
    D_SOURCE: source.txt file was not read successfully.
```

This message occurs whenever the d source model has experienced any difficulty in loading the source.txt (or user-specified) file. This will occur with any of the following problems:

- Width of a vector line of the source file is incorrect.

- A Timepoint value is duplicated or is otherwise not monotonically increasing.

- One of the output values was not a valid 12-State value (0s, 1s, Us, 0r, 1r, Ur, 0z, 1z, Uz, 0u, 1u, Uu).

28.3.6   Code Model d_state

```
loading_error:
    ***ERROR***
    D_STATE: state.in file was not read successfully.
    The most common cause of this problem is a trailing
    blank line in the state.in file
```

This error occurs when the state.in file (or user-named state machine input file) has not been read successfully. This is due to one of the following:

- The counted number of tokens in one of the file's input lines does not equal that required to define either a state header or a continuation line (Note that all comment lines are ignored, so these will never cause the error to occur).

- An output state value was defined using a symbol which was invalid (i.e., it was not one of the following: 0s, 1s, Us, 0r, 1r, Ur, 0z, 1z, Uz, 0u, 1u, Uu).

- An input value was defined using a symbol which was invalid (i.e., it was not one of the following: 0, 1, X, or x).

```
index_error:
    ***ERROR***
    D_STATE: An error exists in the ordering of states values
    in the states->state[] array. This is usually caused
    by non-contiguous state definitions in the state.in file
```

This error is caused by the different state definitions in the input file being non-contiguous. In general, it will refer to the different states not being defined uniquely, or being "broken up" in some fashion within the state.in file.

28.3.7   Code Model oneshot

```
oneshot_allocation_error:
    **** Error ****
    ONESHOT: Error allocating oneshot block storage
```

Generic storage allocation error.

```
oneshot_array_error:
    **** Error ****
    ONESHOT: Size of control array different than pulse-width array
```

This error indicates that the control array and pulse-width arrays are of different sizes.

```
oneshot_pw_clamp:
    **** Warning ****
    ONESHOT: Extrapolated Pulse-Width Limited to zero
```

This error indicates that for the current control input, a pulse-width of less than zero is indicated. The model will consequently limit the pulse width to zero until the control input returns to a more reasonable value.

28.3.8   Code Model pwl

```
allocation_error:
    ***ERROR***
    PWL: Allocation calloc failed!
```

Generic storage allocation error.

```
limit_error:
    ***ERROR***
    PWL: Violation of 50% rule in breakpoints!
```

This error message indicates that the pwl model has an absolute value for its input domain, and that the x_array coordinates are so close together that the required smoothing regions would overlap. To fix the problem, you can either spread the x_array coordinates out or make the input domain value smaller.

28.3.9   Code Model s_xfer

```
num_size_error:
    ***ERROR***
    S_XFER: Numerator coefficient array size greater than
    denominator coefficient array size.
```

This error message indicates that the order of the numerator polynomial specified is greater than that of the denominator. For the s_xfer model, the orders of numerator and denominator polynomials must be equal, or the order of the denominator polynomial must be greater than that or the numerator.

28.3.10   Code Model sine

```
allocation_error:
    **** Error ****
    SINE: Error allocating sine block storage
```

Generic storage allocation error.

```
sine_freq_clamp:
    **** Warning ****
    SINE: Extrapolated frequency limited to 1e-16 Hz
```

This error occurs whenever the controlling input value is such that the output frequency ordinarily would be set to a negative value. Consequently, the output frequency has been clamped to a near-zero value.

```
array_error:
    **** Error ****
    SINE: Size of control array different than frequency array
```

This error message normally occurs whenever the controlling input array and the frequency array are different sizes.

28.3.11   Code Model square

```
square_allocation_error:
    **** Error ****
    SQUARE: Error allocating square block storage
```

Generic storage allocation error.

```
square_freq_clamp:
    **** WARNING ****
    SQUARE: Frequency extrapolation limited to 1e-16
```

This error occurs whenever the controlling input value is such that the output frequency ordinarily would be set to a negative value. Consequently, the output frequency has been clamped to a near-zero value.

```
square_array_error:
    **** Error ****
    SQUARE: Size of control array different than frequency array
```

This error message normally occurs whenever the controlling input array and the frequency array are different sizes.

28.3.12 Code Model triangle

```
triangle_allocation_error:
    **** Error ****
    TRIANGLE: Error allocating triangle block storage
```

Generic storage allocation error.

```
triangle_freq_clamp:
    **** Warning ****
    TRIANGLE: Extrapolated Minimum Frequency Set to 1e-16 Hz
```

This error occurs whenever the controlling input value is such that the output frequency ordinarily would be set to a negative value. Consequently, the output frequency has been clamped to a near-zero value.

```
triangle_array_error:
    **** Error ****
    TRIANGLE: Size of control array different than frequency array
```

This error message normally occurs whenever the controlling input array and the frequency array are different sizes.

Part III

CIDER

Chapter 29

CIDER User's Manual

The CIDER User's Manual that follows is organized as a series of individual UNIX-style manual pages. At the end of the manual, several examples illustrating the use of CIDER are given.

29.1 SPECIFICATION

Overview of numerical-device specification

The input to CIDER consists of a SPICE-like description of a circuit, its analyses and its compact device models, and PISCES-like descriptions of numerically analyzed device models. For a description of the SPICE input format, consult the SPICE3 Users Manual [JOHN92].

To simulate devices numerically, two types of input must be added to the input file. The first is a model description in which the common characteristics of a device class are collected. In the case of numerical models, this provides all the information needed to construct a device cross-section, such as, for example, the doping profile. The second tpye of input consists of one or more element lines that specify instances of a numerical model, describe their connection to the rest of the circuit, and provide additional element-specific information such as device layout dimensions ans initial bias information.

The format of a numerical device model description differs from the standard approach used for SPICE3 compact models. It begins the same way with one line containing the .MODEL keywor followed by the name of the model, device type and modeling level. However, instead of providing a single long list of parameters and their values, numerical model parameters are grouped onto **cards**. Each type of card has its own set of valid paramters. In all cases, the relative ordering of different types of cards is unimportant. However, for cards of the same type (such as mesh-specification cards), their order in the input file can be important in determining the device structure.

Each card begins on a separate line of the input file. In order to let CIDER know that card lines are continuations of a numerical model description, each must begin with the continuation character "+". If there are too many parameters on a given card to allow it fit on a single line, the card can be continued by adding a second "+" to the beginning of the next line. However, the name and value of a parameter should always appear on the same line.

Several features are provided to make the numerical model format more convenient.

Blank space can follow the initial "+" to separate it from the name of a card or the card continuation "+". Blank lines are also permitted, as long as they also begin with an initial "+". Parentheses and commas can be used to visually group or separate parameter definitions. In addition, while it is common to add an equal sign between a parameter and its value, this is not strictly necessary.

The name of any card can be abbreaviated, provided that the abbreviation is unique. Parameter name abbreviations can also be used if they are unique in the list of a card's parameters. Numeric parameter values are treated identically as in SPICE3, so exponential notation, engineering scale facotrs and units can be attached to parameter values: `tau=10ns, nc=3.0e19cm^-3`. In SPICE3, the value of a FLAG model parameter is changed to TRUE simply by listing its name on the model line. In CIDER, the value of a numerical model FLAG parameter can be turned back to FALSE by preceding it by a caret "^". This minimizes the amount of input cahnge needed when features such as debugging are turned on and off. In certain cases it is necessary to include filenames in the input description and these names may contain capital letters. If the filename is part of an element line, the inout parser will convert these capitals to lowercase letters. To protect capitalization at any tyme, simply enclose the string in double quotes "".

The remainder of this manual describes how numerically anayzed elements anf models can be used in CIDER simulations. The manual consists of three parts. First, all of the model cards and their parameters are described. This is followed by a section describing the three basic types of numerical models and their corresponding element lines. In the final section, several complete examples of CIDER simulations are presented.

Several conventions are used in the card descriptions. In the card synopses, the name of a card is followed by a list of parameter classes. Each class is represented by a section in the card parameter table, in the same order as it appears in the synopsis line. Classes which contain optional parameters are surrounded by brackets: [...]. Sometimes it only makes sense for a single parameter to take effect. (For example, a material can not simultaneously be both Si and SiO2.) In such cases, the various choices are listed sequentially, separated by colons. The same paramter often has a number of different acceptable names, some of which are listed in the parameter tables.[1] These aliases are separated by vertical bars: "|". Finally, in the card examples, the model continuation pluses have been removed from the card lines for clarity's sake.

29.1.1   Examples

The model description for a two-dimensional numerical diode might look something like what follows. This example demonstrates many of the features of the input format described above. Notice how the .MODEL line and the leading pluses form a border around the model description:

Example: Numerical diode

```
.MODEL M–NUMERICAL NUPD LEVEL=2
+ cardnamel numberl−vall (number? valz), (number3 = va13)
+ cardname2 numberl=vall string1 = name1
+
+ cardname3 numberl=vall, flagl, ^flag2
+ + number2=val2, flag3
```

The element line for an instance of this model might look something like the following. Double quotes are used to protect the filename from decapitalization:

```
dl 1 2 M–NUMERICAL area=lOOpm"2 ic.file = "diode.IC" '
```

29.2   BOUNDARY, INTERFACE

Specify properties of a domain boundary or the interface between two boundaries
SYNOPSIS

```
boundary domain lbounding−box1 [properties]
interface domain neighbor lbounding−box1 [properties]
```

29.2.1   DESCRIPTION

The boundary and interface cards are used to set surface physics parameters along the boundary of a specified domain. Normally, the parameters apply to the entire boundary, but there are two ways to restrict the area of interest. If a neighboring domain is also specified, the parameters are only set on the interface between the two domains. In addition, if a bounding box is given, only that portion of the boundary or interface inside the bounding box will be set.

If a semiconductor-insulator interface is specified, then an estimate of the width of any inversion or accumulation layer that may form at the interface can be prdvided. If the surface mobility model (cf. **models** card) is enabled, then the model will apply to all semiconductor portions of the device within this estimated distance of the interface. If a point lies within the estimated layer width of more than one interface, it belong to the interface specified first in the input file. If the layer width given is less than or

---

[1]Some of the possibilities are not listed in order to shorten the lengths of the parameter tables. This makes the use of parameter abbreviations somewhat troublesome since an unlisted parameter may abbreviate to the same name as one that is listed. CIDER will produce a warning when this occurs. Many of the undocumented parameter names are the PISCES names for the same parameters. The adventurous soul can discover these names by delving through the "cards" directory of the source code distribution looking for the C parameter tables.

equal to zero, it is automatically replaced by an estimate calculated from the doping near the interface. As a consequence, if the doping varies so will the layer width estimate.

Each edge of the bounfing box can be specified in terms of its location or its mesh-index in the relevant dimension, or defaulted to the respective boundary of the simulation mesh.

29.2.2 PARAMETERS

Name	Type	Description	Units
Domain	Integer	ID number of primary domain	
Neighbor	Integer	ID number of neighboring domain	
X.Low	Real	Lowest X location of bounding box	μm
: IX.Low	Integer	Lowest X mesh-index of bounding box	
X.High	Real	Highest X location of bounding box	μm
: IX.High	Integer	Highest X mesh-index of bounding box	
Y.Low	Real	Lowest Y location of bounding box	μm
: IY.Low	Integer	Lowest Y mesh-index of bounding box	
Y.High	Real	Highest Y location of bounding box	μm
:IY.High	Integer	Highest Y mesh-index of bounding box	
Qf	Real	Fixed interface charge	C/cm^2
SN	Real	Surface recombination velocity - electrons	cm/s
SP	Real	Surface recombination velocity - holes	cm/s
Layer.Width	Real	Width of surface layer	μm

29.2.3 EXAMPLES

The following shows how the surface recombination velocities at an Si-SiO2 interface might be set:

interface dom=1 neigh=2 sn=1.Oe4 sp=1.Oe4

In a MOSFET with a $2.0\mu m$ gate width and $0.1\mu m$ source and drain overlap, the surfacechannel can be restricted to the region between the metallurgical junctions and within 100 $\mathring{A}$ (0.01 μm) of the interface:

interface dom=1 neigh=2 x.l=1.1 x.h=2.9 layer.w=0.01

The inversion layer width in the previous example can be automatically determined by setting the estimate to 0.0:

interface dom=1 neigh=% x.l=1.1 x.h=2.9 layer.w=0.0

29.3 COMMENT

Add explanatory comments to a device definition
    SYNOPSIS

    comment [text]
    * [text]
    $ [text]
    # [text]

29.3.1 DESCRIPTION

Annotations can be added to a device definition using the comment card. All text on a comment card is ignored. Several popular commenting characters are also supported as aliases: '*' from SPICE, '$' from PISCES, and '#' from UNIX shell scripts.

29.3.2 EXAMPLES

A SPICE-like comment is followed by a PISCES-like comment and shell script comment:

```
* CIDER and SPICE would ignore this input line
$ CIDER and PISCES would ignore this , but SPICE wouldn't
# CIDER and UNIX Shell scripts would ignore this input line
```

29.4   CONTACT

Specify properties of an electrode
    SYNOPSIS

```
contact number [workfunction]
```

29.4.1   DESCRIPTION

The properties of an electrode can be set using the contact card. The only changeable property is the workfunction of the electrode material and this only affects contacts made to an insulating material. All contacts to semiconductor material are assumed to be ohmic in nature.

29.4.2   PARAMETERS

Name	Type	Description
Number	Integer	ID number of the electrode
Workfunction	Real	Workfunction of electrode material. (eV)

29.4.3   EXAMPLES

The following shows how the workfunction of the gate contact of a MOSFET might be changed to a value appropriate for a P+ polysilicon gate:

```
contact num=2 workf=5.29
```

29.4.4   SEE ALSO

electrode, material

29.5   DOMAIN, REGION

Identify material-type for section of a device
    SYNOPSIS

```
domain number material [position]
region number material [position]
```

29.5.1   DESCRIPTION

A device is divided into one or more rectilinear domains, each of which has a unique identification number and is composed of a particular material.

Domain (aka region) cards are used to build up domains by associating a material type with a box-shaped section of the device. A single domain may be the union 0f multiple boxes. When multiple domain cards overlap in space, the one occurring last in the input file will determine the ID number and material type of the overlapped region.

Each edge of a domain box can be specified in terms of its location or mesh-index in the relevant dimension, or defaulted to the respective boundary of the simulation mesh.

29.5.2 PARAMETERS

Name	Type	Description
Number	Integer	ID number of this domain
Material	Integer	ID number of material used by this domain
X.Low	Real	Lowest X location of domain box, (μm)
:IX.Low	Integer	Lowest X mesh-index of domain box
X.High	Real	Highest X location of domain box, (μm)
:IX-High	Integer	Highest X mesh-index of domain box
Y.Low	Real	Lowest Y location of domain box, (μm)
:IY.Low	Integer	Lowest Y mesh-index of domain box
Y.High	Real	Highest Y location of domain box, (μm)
:IY.High	Integer	Highest Y mesh-index of domain box

29.5.3 EXAMPLES

Create a 4.0 pm wide by 2.0 pm high domain out of material #1:

    domain num=1 material=1 x.l=O.O x.h=4.0 y.l=O.O y.h=2.0

The next example defines the two domains that would be typical of a planar MOSFET simulation. One occupies all of the mesh below y = 0 and the other occupies the mesh above y = 0. Because the x values are left unspecified, the low and high x boundaries default to the edges of the mesh:

    domain n=1 m=1 y.l=O.O
    domain n=2 m=2 y.h=O.O

29.5.4 SEE ALSO

x.mesh, material

29.6 DOPING

Add dopant to regions of a device
   SYNOPSIS

    doping [domains] profile−type [lateral−profile−type] [axis] [impurity−type1 [constant-

29.6.1 DESCRIPTION

Doping cards are used to add impurities to the various domains of a device. Initially each domain is dopant-free. Each new doping card creates a new doping profile that defines the dopant concentration as a function of position. The doping at a particular location is then the sum over all profiles of the concentration values at that position. Each profile can be restricted to a subset of a device's domains by supplying a list of the desired domains.

Otherwise, all domainsare doped by each profile.

A profile has uniform concentration inside the constant box. Outside this region, it varies according to the primary an lateral profile shapes. In 1D devices the lateral shape is unused and in 2D devices the y-axis is the default axis for the primary profile. Several analytici functions can be used to define the primary profile shape. Alternatively, empirical or simulated profile data can be extracted froma a file. For the analytic profiles, the doping is the prduct of a profile function (e.g. Gaussian) and a reference concentration, which is either the constat concentration of a uniform profile, or the peak concentration for any of the other functions. If concentration data is used instead take from an ASCII file containing a list of location-concentration pairs or a SUPREM3 exported file, the name of the file must be provided. If necessary, the final concentration at a point is then found by multiplying the primary profile concentration by the value of the lateral profile function at that point. Empirical profiles must first be normalized by the value at 0.0 to provide a usable profile functions. Alternatively, the second dimension can be included
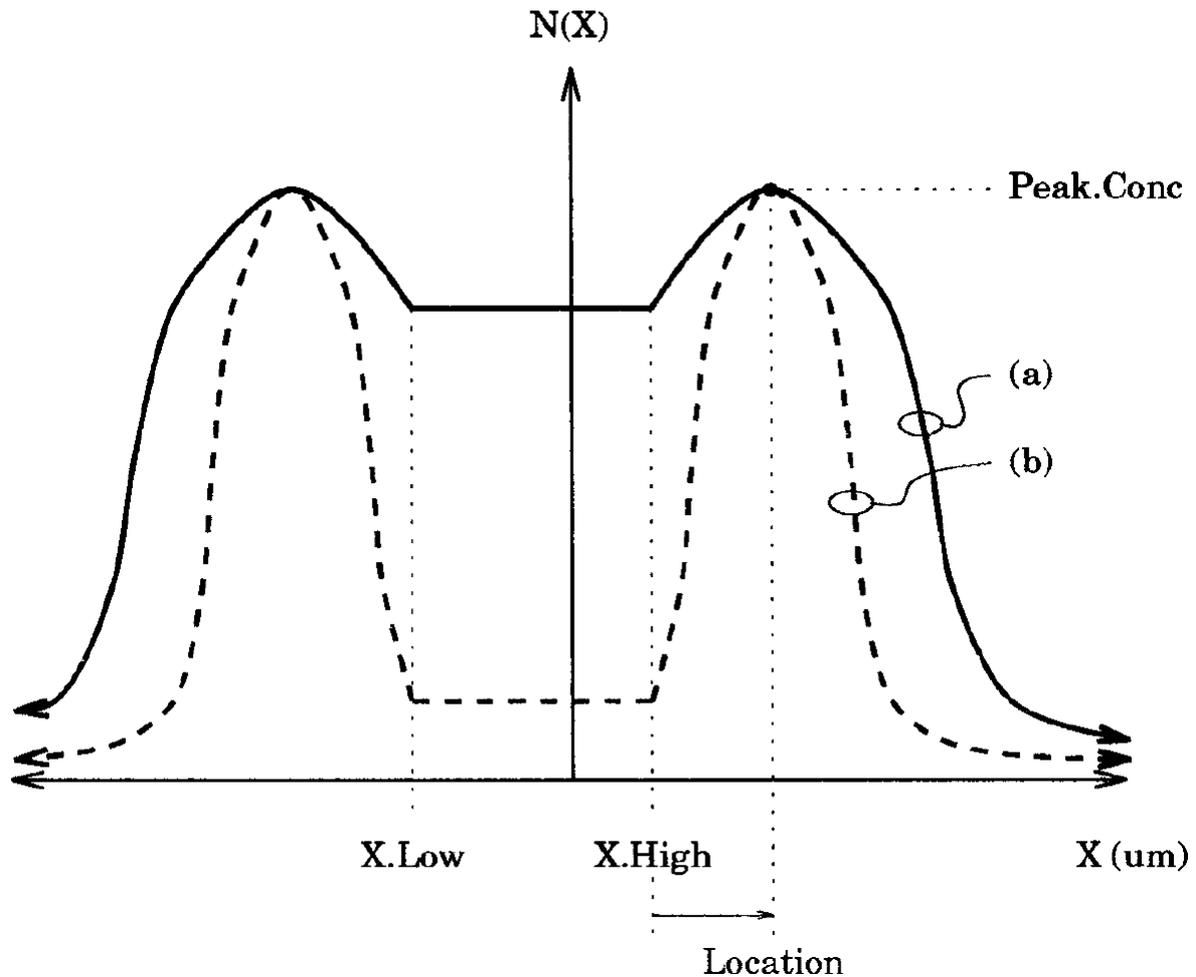
Figure 29.1: 1D doping profiles with location > 0.

by assigning the same concentration to all points equidistant from the edges of the constant box. The contours of the profile are the circular.

Unless otherwise specified, the added impurities are assumes to be N type. However, the name of a specific dopant species is needed when extracting concentration information for that impurity from a SUPREM3 exported file.

Several parameters are used to adjust the basic shape of a profile functions so that the final, constructed profile, matches the doping profile in the real device. The constant box region should coincide with a region of constant concentration in the device. For uniform profiles its boundaries default to the mesh boundaries. For the other profiles the constant box starts as a point and only acquires width or height if both the appropriate edges are specified. The location of the peak of the primary profile can be moved away from the edge of the constat box. A positive location places the peak outside the constant box (cf. Fig. 29.1), and a negative value puts it inside the constant box (cf. Fig. 29.2). The concentration in the constant box is then equal to the value of the profile when it intersects the edge of the constant box. The argument of the profile function is a distance expressed in terms of the characteristic length (by default equal to $1\mu m$). The longer this length, the more gradually the profile will change. For example, in Fig. A.1 and Fig A.2, the profiles marked (a) have characteristic lengths twice those of the profiles marked (b). The location and characteristic length for the lateral profile are multiplied by the lateral atio. This allows th use of different length scales for the primary and lateral profilles. For rotated profiles, this scaling is taken into account, and the profile contours are elliptical rather tha circular.
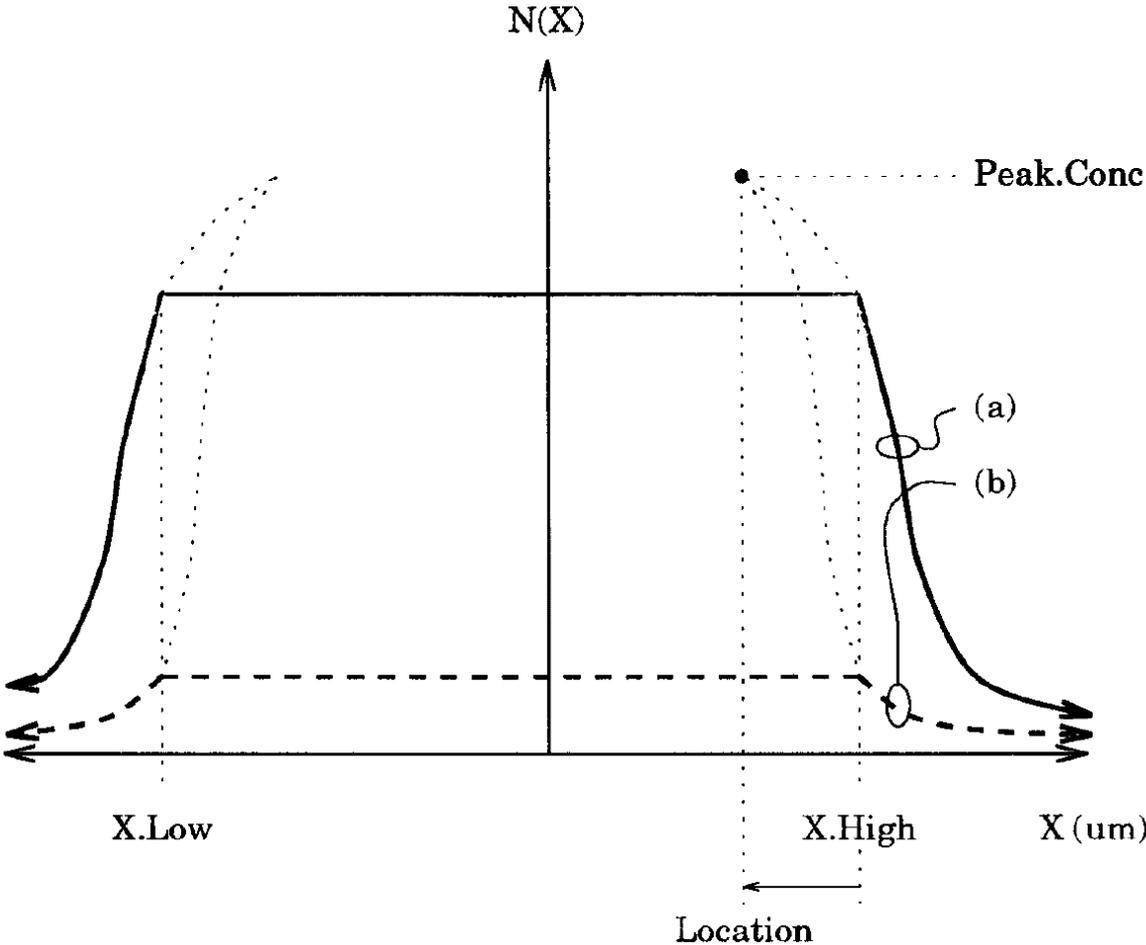
Figure 29.2: 1D doping profiles with location < 0.

29.6.2 PARAMETERS

Name	Type	Description
Domains	Int List	List of domains to dope
Uniform: Linear: Erfc: Exponential: Suprem3: Ascii: Ascii Suprem3	Flag	Primary profile type
InFile	String	Name of Suprem3, Ascii or Ascii Suprem3 input file
Lat.Rotate: Lat.Unif: Lat.Lin: Lat.Gauss: Lat.Erfc: Lat.Exp	Flag	Lateral profile type
X.Axis:Y.Axis	Flag	Primary profile direction
N.Type: P.Type: Donor: Acceptor: Phosphorus: Arsenic: Antimony: Boron	Flag	Impurity type
X.Low	Real	Lowest X location of constant box, (μm)
X.High	Real	Highest X location of constant box, (μm)
Y.Low	Real	Lowest Y location of constant box, (μm)
Y.High	Real	Highest Y location of constant box, (μm)
Conc | Peak.conc	Real	Dopant concentration, (cm^{-3})
Location | Range	Real	Location of profile edge/peak, (μm)
Char.Length	Real	Characteristic length of profile, (μm)
Ratio.Lat	Real	Ratio of lateral to primary distances

29.6.3 EXAMPLES

This first example adds a uniform background P-type doping of $1.0 \times 10^{16} cm^{-3}$ to an entire device:

```
doping uniform p.type conc=1.0e16
```

A gaussian implantation with rotated lateral falloff, such as might be used for a MOSFET source, is then added:

```
doping gauss lat.rotate n.type conc=1.Oe19
+ x.l=O.O x.h=0.5 y.l=O.O y.h=0.2 ratio=0.7
```

Alternatively, an error-function falloff could be used:

```
doping gauss lat.crfc conc=1.Oe19
+ x.l=O.O x.h=O.5 y.l=O.O y.h=0.2 ratio=0.7
```

Finally, the MOSFET channel implant is extracted from an ASCII-format SUPREM3 file. The lateral profile is uniform, so that the implant is confined between $X = 1\mu m$ and $X = 3\mu m$. The profile begins at $Y = 0\mu m$ (the high Y value defaults equal to the low Y value):

```
doping ascii suprem3 infile=implant.s3 lat.unif boron
+ x.l=1.0 x.h=3.0 y.l=O.O
```

29.6.4 SEE ALSO

domain, mobility, contact, boundary

29.7 ELECTRODE

Set location of a contact to the device
    SYNOPSIS

electrode [number] [position]

29.7.1 DESCRIPTION

Each device has several electrodes which are used to connect the device to the rest of the circuit. The number of electrodes depends on the type of device. For example, a MOSFET needs 4 electrodes. A particular electrode can be identified by its position in the list of circuit nodes on the device element line. For example, the drain node of a MOSFET is electrode number 1, while the bulk node is electrode number 4. Electrodes fo which an ID number has not been specified are assigned values sequentially in the order they appear in the input file.

For lD devices, the positions of two of the electrodes are predefined to be at the ends of the simulation mesh. The first electrode is at the low end of the mesh, and the last electrode is at the high end. The position of the special lD BJT base contact is set on the options card. Thus, electrode cards are used exclusively for 2D devices.

Each card associates a portion of the simulation mesh with a particular electrode. In contrast to domains, which are specified only in terms of boxes, electrodes can also be specified in terms of line segments. Boxes and segments for the same electrode do not have to overlap. If they don't, it is assumed that the electrode is wired together outside the area covered by the simulation mesh. However, pieces of different electrodes <u>must not</u> overlap, since this would represent a short circuit. Each electrode box or segment can be specified in terms of the locations or mesh-indices of its boundaries. A missing value defaults to the corresponding mesh boundary.

29.7.2 PARAMETERS

Name	Type	Description
Number	Integer	ID number of this domain
X.Low	Real	Lowest X location of electrode, (μm)
:IX.Low	Integer	Lowest X mesh-index of electrode
X.High	Real	Highest X location of electrode, (μm)
:IX.High	Integer	Highest X mesh-index of electrode
Y.Low	Real	Lowest Y location of electrode, (μm)
:IY.Low	Integer	Lowest Y mesh-index of electrode
Y.High	Real	Highest Y location of electrode, (μm)
:IY.High	Integer	Highest Y mesh-index of electrode

29.7.3 EXAMPLES

The following shows how the four contacts of a MOSFET might be specified:

```
* DRAIN
electrode x.l=O.O x.h=0.5 y.l=O.O y.h=O.O
* GATE
electrode x.1=1.0 x.h=3.0 iy.l=O iy.h=O
* SOURCE
electrode x.1=3.0 x.h=4.0 y.l=O.O y.h=O.O
* BULK
electrode x.l=O.O x.h=4.0 y.l=2.0 y.h=2.0
```

The numbering option can be used when specifying bipolar transistors with dual base contacts:

```
* EMITTER
electrode num=3 x.1=1.0 x.h=2.0 y.l=O.O y.h=O.O
* BASE
electrode num=2 x.l=O.O x.h=0.5 y.l=O.O y.h=O.O
electrode num=2 x.1=2.5 x.h=3.0 y.l=O.O y.h=O.O
* COLLECTOR
electrode num=l x.l=O.O x.h=3.0 y.l=l.O y.h−1.0
```

29.7.4   SEE ALSO

domain, contact

29.8   END

Terminate processing of a device definition

   SYNOPSIS

   end

29.8.1   DESCRIPTION

The end card stops processing of a device definition. It may appear anywhere within a definition.
Subsequent continuation lines of the definition will be ignored. If nq end card is supplied, all the cards
will be processed.

29.9   MATERIAL

Specify physical properties of a material

   SYNOPSIS

   material number type [physical−constants]

29.9.1   DESCRIPTION

The material card is used to create an entry in the list of materials used in a device. Each entry needs
a unique identification number and the type of the material. Default values are assigned to the physical
properties of the material. Most material parameters are accessible either here or on the `mobility` or
`contact` cards. However, some parameters remain inaccessible (e.g. the ionization coefficient parameters).
Parameters for most physical effect models are collected here. Mobility parameters are handled separately
by the `mobility` card. Properties of electrode materials are set using the `contact` card.

29.9.2   PARAMETERS

Name	Type	Description
Number	Integer	ID number of this material
Semiconductor : Silicon : Polysilicon : GaAs : Insulator : Oxide : Nitride	Flag	Type of this material
Affininity	Real	Electron affinity (eV)
Permittivity	Real	Dielectric permittivity (F/cm)
Nc	Real	Conduction band density (cm^{-3})
Nv	Real	Valence band density (cm^{-3})
Eg	Real	Energy band gap (eV)
dEg.dT	Real	Bandgap narrowing with temperature ($eV/°K$)
Eg.Tref	Real	Bandgap reference temperature, (°K)
dEg.dN	Real	Bandgap narrowing with N doping, (eV/cm^{-3})
Eg.Nref	Real	Bandgap reference concentration - N type, (cm^{-3})
dEg.dP	Real	Bandgap narrowing with P doping, (eV/cm^{-3})
Eg.Pref	Real	Bandgap reference concentration - P type, (cm^{-3})
TN	Real	SRH lifetime - electrons, (sec)
SRH.Nref	Real	SRH reference concentration - electrons (cm^{-3})
TP	Real	SRH lifetime - holes, (sec)
SRH.Pref	Real	SRH reference concentration - holes (cm^{-3})
CN	Real	Auger coefficient - electrons (cm^6/sec)
CP	Real	Auger coefficient - holes (cm^6/sec)
ARichN	Real	Richardson constant - electrons, ($A/\frac{cm^2}{°K^2}$)
ARichP	Real	Richardson constant - holes, ($A/\frac{cm^2}{°K^2}$)

29.9.3   EXAMPLES

Set the type of material #1 to silicon, then adjust the values of the temperature-dependent bandgap model parameters:

    material num=1 silicon eg=1.12 deg.dt=4.7e-4 eg.tref=640.0

The recombination lifetimes can be set to extremely short values to simulate mperfect semiconductor material:

    material num=2 silicon tn=1ps tp=1ps

29.9.4   SEE ALSO

domain, mobility, contact, boundary

29.10   METHOD

Choose types and parameters of numerical methods
    SYNOPSIS

    method [types] [parameters]

29.10.1   DESCRIPTION

The method card controls which numerical methods are used during a simulation and the parameters of these methods. Most of these methods are optimizations that r time, but may sacrifice accuracy or reliable convergence.

For majority-carrier devices such as MOSFETs, one carrier simulations ca to save simulation time. The systems of equations in AC analysis may be sol either direct or successive-over-relaxation techniques.

Successive-over-relaatior but at high frequencies, it may fail to converge or may converge to the wrong a some cases, it is desirable to obtain AC parameters as functions of DC bias c If necessary, a one-point AC analysis is performed at a predefined frequency ii obtain these small-signal parameters. The default for this frequency is 1 Hz. Thc matrix for DC and transient analyses can be simplified by ignoring the derivati mobility with respect to the solution variables. However, the resulting analysis convergence problems. Additionally, if they are ignored during AC analyses, results may be obtained. A damped Newton method is used as the primary solution technique for the dt partial differential equations. This algorithm is based on an iterative loop that tl when the error in the solution is small enough or the iteration limit is reach1 tolerances are used when determining if the error is "small enough". The toler expressed in terms of an absolute, solution-independent error and a relative, dependent error. The absolute-error limit can be set on this card. The relati1 computed by multiplying the size of the solution by the circuit level SPICE I RELTOL.

Part IV

Appendices

Chapter 30

Model and Device Parameters

The following tables summarize the parameters available on each of the devices and models in ngspice. There are two tables for each type of device supported by ngspice. Input parameters to instances and models are parameters that can occur on an instance or model definition line in the form `keyword=value` where `keyword` is the parameter name as given in the tables. Default input parameters (such as the resistance of a resistor or the capacitance of a capacitor) obviously do not need the keyword specified.

Output parameters are those additional parameters which are available for many types of instances for the output of operating point and debugging information. These parameters are specified as `@device[keyword]` and are available for the most recent point computed or, if specified in a `.save` statement, for an entire simulation as a normal output vector. Thus, to monitor the gate-to-source capacitance of a MOSFET, a command

```
save @m1[cgs]
```

given before a transient simulation causes the specified capacitance value to be saved at each timepoint, and a subsequent command such as

```
plot @m1[cgs]
```

produces the desired plot. (Note that the `show` command does not use this format).

Some variables are listed as both input and output, and their output simply returns the previously input value, or the default value after the simulation has been run. Some parameter are input only because the output system can not handle variables of the given type yet, or the need for them as output variables has not been apparent. Many such input variables are available as output variables in a different format, such as the initial condition vectors that can be retrieved as individual initial condition values. Finally, internally derived values are output only and are provided for debugging and operating point output purposes.

Please note that these tables do not provide the detailed information available about the parameters provided in the section on each device and model, but are provided as a quick reference guide.

30.1   Elementary Devices

30.1.1   Resistor

Resistor instance parameters

#	Name	Direction	Type	Description
1	resistance	InOut	real	Resistance
10	ac	InOut	real	AC resistance value
8	temp	InOut	real	Instance operating temperature
14	dtemp	InOut	real	Instance temperature difference with the rest of the circuit
3	l	InOut	real	Length
2	w	InOut	real	Width
12	m	InOut	real	Multiplication factor
16	tc	InOut	real	First order temp. coefficient
16	tc1	InOut	real	First order temp. coefficient
17	tc2	InOut	real	Second order temp. coefficient
13	scale	InOut	real	Scale factor
15	noisy	InOut	integer	Resistor generate noise
5	sens_resist	In	flag	flag to request sensitivity WRT resistance
6	i	Out	real	Current
7	p	Out	real	Power
206	sens_dc	Out	real	dc sensitivity
201	sens_real	Out	real	dc sensitivity and real part of ac sensitivity
202	sens_imag	Out	real	dc sensitivity and imag part of ac sensitivity
203	sens_mag	Out	real	ac sensitivity of magnitude
204	sens_ph	Out	real	ac sensitivity of phase
205	sens_cplx	Out	complex	ac sensitivity

Resistor model parameters

#	Name	Direction	Type	Description
103	rsh	InOut	real	Sheet resistance
105	narrow	InOut	real	Narrowing of resistor
105	dw	InOut	real	
108	short	InOut	real	Shortening of resistor
108	dlr	InOut	real	
101	tc1	InOut	real	First order temp. coefficient
102	tc2	InOut	real	Second order temp. coefficient
104	defw	InOut	real	Default device width
104	w	InOut	real	
109	kf	InOut	real	Flicker noise coefficient
110	af	InOut	real	Flicker noise exponent
107	tnom	InOut	real	Parameter measurement temperature
106	r	In	flag	Device is a resistor model

30.1.2   Capacitor - Fixed capacitor

Capacitor instance parameters

#	Name	Direction	Type	Description
1	capacitance	InOut	real	Device capacitance
1	cap	InOut	real	Device capacitance
1	c	InOut	real	Device capacitance
2	ic	InOut	real	Initial capacitor voltage
8	temp	InOut	real	Instance operating temperature
9	dtemp	InOut	real	Instance temperature difference from the rest of the circuit
3	w	InOut	real	Device width
4	l	InOut	real	Device length
11	m	InOut	real	Parallel multiplier
10	scale	InOut	real	Scale factor
5	sens_cap	In	flag	flag to request sens. WRT cap.
6	i	Out	real	Device current
7	p	Out	real	Instantaneous device power
206	sens_dc	Out	real	dc sensitivity
201	sens_real	Out	real	real part of ac sensitivity
202	sens_imag	Out	real	dc sens. & imag part of ac sens.
203	sens_mag	Out	real	sensitivity of ac magnitude
204	sens_ph	Out	real	sensitivity of ac phase
205	sens_cplx	Out	complex	ac sensitivity

Capacitor model parameters

#	Name	Direction	Type	Description
112	cap	InOut	real	Model capacitance
101	cj	InOut	real	Bottom Capacitance per area
102	cjsw	InOut	real	Sidewall capacitance per meter
103	defw	InOut	real	Default width
113	defl	InOut	real	Default length
105	narrow	InOut	real	width correction factor
106	short	InOut	real	length correction factor
107	tc1	InOut	real	First order temp. coefficient
108	tc2	InOut	real	Second order temp. coefficient
109	tnom	InOut	real	Parameter measurement temperature
110	di	InOut	real	Relative dielectric constant
111	thick	InOut	real	Insulator thickness
104	c	In	flag	Capacitor model

30.1.3   Inductor - Fixed inductor

Inductor instance parameters

#	Name	Direction	Type	Description
1	inductance	InOut	real	Inductance of inductor
2	ic	InOut	real	Initial current through inductor
5	sens_ind	In	flag	flag to request sensitivity WRT inductance
9	temp	InOut	real	Instance operating temperature
10	dtemp	InOut	real	Instance temperature difference with the rest of the circuit
8	m	InOut	real	Multiplication Factor
11	scale	InOut	real	Scale factor
12	nt	InOut	real	Number of turns
3	flux	Out	real	Flux through inductor
4	v	Out	real	Terminal voltage of inductor
4	volt	Out	real	
6	i	Out	real	Current through the inductor
6	current	Out	real	
7	p	Out	real	instantaneous power dissipated by the inductor
206	sens_dc	Out	real	dc sensitivity sensitivity
201	sens_real	Out	real	real part of ac sensitivity
202	sens_imag	Out	real	dc sensitivity and imag part of ac sensitivty
203	sens_mag	Out	real	sensitivity of AC magnitude
204	sens_ph	Out	real	sensitivity of AC phase
205	sens_cplx	Out	complex	ac sensitivity

Inductor model parameters

#	Name	Direction	Type	Description
100	ind	InOut	real	Model inductance
101	tc1	InOut	real	First order temp. coefficient
102	tc2	InOut	real	Second order temp. coefficient
103	tnom	InOut	real	Parameter measurement temperature
104	csect	InOut	real	Inductor cross section
105	length	InOut	real	Inductor length
106	nt	InOut	real	Model number of turns
107	mu	InOut	real	Relative magnetic permeability
108	l	In	flag	Inductor model

30.1.4   Mutual - Mutual Inductor

Mutual instance parameters

#	Name	Direction	Type	Description
401	k	InOut	real	Mutual inductance
401	coefficient	InOut	real	
402	inductor1	InOut	instance	First coupled inductor
403	inductor2	InOut	instance	Second coupled inductor
404	sens_coeff	In	flag	flag to request sensitivity WRT coupling factor
606	sens_dc	Out	real	dc sensitivity
601	sens_real	Out	real	real part of ac sensitivity
602	sens_imag	Out	real	dc sensitivity and imag part of ac sensitivty
603	sens_mag	Out	real	sensitivity of AC magnitude
604	sens_ph	Out	real	sensitivity of AC phase
605	sens_cplx	Out	complex	mutual model parameters:

30.2   Voltage and current sources

30.2.1   ASRC - Arbitrary source

ASRC instance parameters

#	Name	Direction	Type	Description
2	i	In	parsetree	Current source
1	v	In	parsetree	Voltage source
7	i	Out	real	Current through source
6	v	Out	real	Voltage across source
3	pos_node	Out	integer	Positive Node
4	neg_node	Out	integer	Negative Node

30.2.2 Isource - Independent current source

Isource instance parameters

#	Name	Direction	Type	Description
1	dc	InOut	real	DC value of source
2	acmag	InOut	real	AC magnitude
3	acphase	InOut	real	AC phase
5	pulse	In	real vector	Pulse description
6	sine	In	real vector	Sinusoidal source description
6	sin	In	real vector	Sinusoidal source description
7	exp	In	real vector	Exponential source description
8	pwl	In	real vector	Piecewise linear description
9	sffm	In	real vector	Single freq. FM description
21	am	In	real vector	Amplitude modulation description
10	neg_node	Out	integer	Negative node of source
11	pos_node	Out	integer	Positive node of source
12	acreal	Out	real	AC real part
13	acimag	Out	real	AC imaginary part
14	function	Out	integer	Function of the source
15	order	Out	integer	Order of the source function
16	coeffs	Out	real vector	Coefficients of the source
20	v	Out	real	Voltage across the supply
17	p	Out	real	Power supplied by the source
4	ac	In	real vector	AC magnitude,phase vector
1	c	In	real	Current through current source
22	current	Out	real	Current in DC or Transient mode
18	distof1	In	real vector	f1 input for distortion
19	distof2	In	real vector	f2 input for distortion

30.2.3   Vsource - Independent voltage source

Vsource instance parameters

#	Name	Direction	Type	Description
1	dc	InOut	real	D.C. source value
3	acmag	InOut	real	A.C. Magnitude
4	acphase	InOut	real	A.C. Phase
5	pulse	In	real vector	Pulse description
6	sine	In	real vector	Sinusoidal source description
6	sin	In	real vector	Sinusoidal source description
7	exp	In	real vector	Exponential source description
8	pwl	In	real vector	Piecewise linear description
9	sffm	In	real vector	Single freq. FM descripton
22	am	In	real vector	Amplitude modulation descripton
16	pos_node	Out	integer	Positive node of source
17	neg_node	Out	integer	Negative node of source
11	function	Out	integer	Function of the source
12	order	Out	integer	Order of the source function
13	coeffs	Out	real vector	Coefficients for the function
14	acreal	Out	real	AC real part
15	acimag	Out	real	AC imaginary part
2	ac	In	real vector	AC magnitude, phase vector
18	i	Out	real	Voltage source current
19	p	Out	real	Instantaneous power
20	distof1	In	real vector	f1 input for distortion
21	distof2	In	real vector	f2 input for distortion
23	r	In	real	pwl repeat start time value
24	td	In	real	pwl delay time value

30.2.4 CCCS - Current controlled current source

CCCS instance parameters

#	Name	Direction	Type	Description
1	gain	InOut	real	Gain of source
2	control	InOut	instance	Name of controlling source
6	sens_gain	In	flag	flag to request sensitivity WRT gain
4	neg_node	Out	integer	Negative node of source
3	pos_node	Out	integer	Positive node of source
7	i	Out	real	CCCS output current
9	v	Out	real	CCCS voltage at output
8	p	Out	real	CCCS power
206	sens_dc	Out	real	dc sensitivity
201	sens_real	Out	real	real part of ac sensitivity
202	sens_imag	Out	real	imag part of ac sensitivity
203	sens_mag	Out	real	sensitivity of ac magnitude
204	sens_ph	Out	real	sensitivity of ac phase
205	sens_cplx	Out	complex	ac sensitivity

30.2.5 CCVS - Current controlled voltage source

CCVS instance parameters

#	Name	Direction	Type	Description
1	gain	InOut	real	Transresistance (gain)
2	control	InOut	instance	Controlling voltage source
7	sens_trans	In	flag	flag to request sens. WRT transimpedance
3	pos_node	Out	integer	Positive node of source
4	neg_node	Out	integer	Negative node of source
8	i	Out	real	CCVS output current
10	v	Out	real	CCVS output voltage
9	p	Out	real	CCVS power
206	sens_dc	Out	real	dc sensitivity
201	sens_real	Out	real	real part of ac sensitivity
202	sens_imag	Out	real	imag part of ac sensitivity
203	sens_mag	Out	real	sensitivity of ac magnitude
204	sens_ph	Out	real	sensitivity of ac phase
205	sens_cplx	Out	complex	ac sensitivity

30.2.6   VCCS - Voltage controlled current source

VCCS instance parameters

#	Name	Direction	Type	Description
1	gain	InOut	real	Transconductance of source (gain)
8	sens_trans	In	flag	flag to request sensitivity WRT transconductance
3	pos_node	Out	integer	Positive node of source
4	neg_node	Out	integer	Negative node of source
5	cont_p_node	Out	integer	Positive node of contr. source
6	cont_n_node	Out	integer	Negative node of contr. source
2	ic	In	real	Initial condition of controlling source
9	i	Out	real	Output current
11	v	Out	real	Voltage across output
10	p	Out	real	Power
206	sens_dc	Out	real	dc sensitivity
201	sens_real	Out	real	real part of ac sensitivity
202	sens_imag	Out	real	imag part of ac sensitivity
203	sens_mag	Out	real	sensitivity of ac magnitude
204	sens_ph	Out	real	sensitivity of ac phase
205	sens_cplx	Out	complex	ac sensitivity

30.2.7   VCVS - Voltage controlled voltage source

VCVS instance parameters

#	Name	Direction	Type	Description
1	gain	InOut	real	Voltage gain
9	sens_gain	In	flag	flag to request sensitivity WRT gain
2	pos_node	Out	integer	Positive node of source
3	neg_node	Out	integer	Negative node of source
4	cont_p_node	Out	integer	Positive node of contr. source
5	cont_n_node	Out	integer	Negative node of contr. source
7	ic	In	real	Initial condition of controlling source
10	i	Out	real	Output current
12	v	Out	real	Output voltage
11	p	Out	real	Power
206	sens_dc	Out	real	dc sensitivity
201	sens_real	Out	real	real part of ac sensitivity
202	sens_imag	Out	real	imag part of ac sensitivity
203	sens_mag	Out	real	sensitivity of ac magnitude
204	sens_ph	Out	real	sensitivity of ac phase
205	sens_cplx	Out	complex	ac sensitivity

30.3 Transmission Lines

30.3.1 CplLines - Simple Coupled Multiconductor Lines

CplLines instance parameters

#	Name	Direction	Type	Description
1	pos_nodes	InOut	string vector	in nodes
2	neg_nodes	InOut	string vector	out nodes
3	dimension	InOut	integer	number of coupled lines
4	length	InOut	real	length of lines

CplLines model parameters

#	Name	Direction	Type	Description
101	r	InOut	real vector	resistance per length
104	l	InOut	real vector	inductance per length
102	c	InOut	real vector	capacitance per length
103	g	InOut	real vector	conductance per length
105	length	InOut	real	length
106	cpl	In	flag	Device is a cpl model

30.3.2   LTRA - Lossy transmussion line

LTRA instance parameters

#	Name	Direction	Type	Description
6	v1	InOut	real	Initial voltage at end 1
8	v2	InOut	real	Initial voltage at end 2
7	i1	InOut	real	Initial current at end 1
9	i2	InOut	real	Initial current at end 2
10	ic	In	real vector	Initial condition vector:v1,i1,v2,i2
13	pos_node1	Out	integer	Positive node of end 1 of t-line
14	neg_node1	Out	integer	Negative node of end 1 of t.line
15	pos_node2	Out	integer	Positive node of end 2 of t-line
16	neg_node2	Out	integer	Negative node of end 2 of t-line

LTRA model parameters

#	Name	Direction	Type	Description
0	ltra	InOut	flag	LTRA model
1	r	InOut	real	Resistance per metre
2	l	InOut	real	Inductance per metre
3	g	InOut	real	
4	c	InOut	real	Capacitance per metre
5	len	InOut	real	length of line
11	rel	Out	real	Rel. rate of change of deriv. for bkpt
12	abs	Out	real	Abs. rate of change of deriv. for bkpt
28	nocontrol	InOut	flag	No timestep control
32	steplimit	InOut	flag	always limit timestep to 0.8*(delay of line)
33	nosteplimit	InOut	flag	don't always limit timestep to 0.8*(delay of line)
34	lininterp	InOut	flag	use linear interpolation
35	quadinterp	InOut	flag	use quadratic interpolation
36	mixedinterp	InOut	flag	use linear interpolation if quadratic results look unacceptable
46	truncnr	InOut	flag	use N-R iterations for step calculation in LTRAtrunc
47	truncdontcut	InOut	flag	don't limit timestep to keep impulse response calculation errors low
42	compactrel	InOut	real	special reltol for straight line checking
43	compactabs	InOut	real	special abstol for straight line checking

30.3.3   Tranline - Lossless transmission line

Tranline instance parameters

#	Name	Direction	Type	Description
1	z0	InOut	real	Characteristic impedance
1	zo	InOut	real	
4	f	InOut	real	Frequency
2	td	InOut	real	Transmission delay
3	nl	InOut	real	Normalized length at frequency given
5	v1	InOut	real	Initial voltage at end 1
7	v2	InOut	real	Initial voltage at end 2
6	i1	InOut	real	Initial current at end 1
8	i2	InOut	real	Initial current at end 2
9	ic	In	real vector	Initial condition vector:v1,i1,v2,i2
10	rel	Out	real	Rel. rate of change of deriv. for bkpt
11	abs	Out	real	Abs. rate of change of deriv. for bkpt
12	pos_node1	Out	integer	Positive node of end 1 of t. line
13	neg_node1	Out	integer	Negative node of end 1 of t. line
14	pos_node2	Out	integer	Positive node of end 2 of t. line
15	neg_node2	Out	integer	Negative node of end 2 of t. line
18	delays	Out	real vector	Delayed values of excitation

30.3.4   TransLine - Simple Lossy Transmission Line

TransLine instance parameters

#	Name	Direction	Type	Description
1	pos_node	In	integer	Positive node of txl
2	neg_node	In	integer	Negative node of txl
3	length	InOut	real	length of line

TransLine model parameters

#	Name	Direction	Type	Description
101	r	InOut	real	resistance per length
104	l	InOut	real	inductance per length
102	c	InOut	real	capacitance per length
103	g	InOut	real	conductance per length
105	length	InOut	real	length
106	txl	In	flag	Device is a txl model

30.3.5   URC - Uniform R. C. line

URC instance parameters

#	Name	Direction	Type	Description
1	l	InOut	real	Length of transmission line
2	n	InOut	real	Number of lumps
3	pos_node	Out	integer	Positive node of URC
4	neg_node	Out	integer	Negative node of URC
5	gnd	Out	integer	Ground node of URC

URC model parameters

#	Name	Direction	Type	Description
101	k	InOut	real	Propagation constant
102	fmax	InOut	real	Maximum frequency of interest
103	rperl	InOut	real	Resistance per unit length
104	cperl	InOut	real	Capacitance per unit length
105	isperl	InOut	real	Saturation current per length
106	rsperl	InOut	real	Diode resistance per length
107	urc	In	flag	Uniform R.C. line model

30.4   BJTs

30.4.1   BJT - Bipolar Junction Transistor

BJT instance parameters

#	Name	Direction	Type	Description
2	off	InOut	flag	Device initially off
3	icvbe	InOut	real	Initial B-E voltage
4	icvce	InOut	real	Initial C-E voltage
1	area	InOut	real	(Emitter) Area factor
10	areab	InOut	real	Base area factor
11	areac	InOut	real	Collector area factor
9	m	InOut	real	Parallel Multiplier
5	ic	In	real vector	Initial condition vector
6	sens_area	In	flag	flag to request sensitivity WRT area
202	colnode	Out	integer	Number of collector node
203	basenode	Out	integer	Number of base node
204	emitnode	Out	integer	Number of emitter node
205	substnode	Out	integer	Number of substrate node
206	colprimenode	Out	integer	Internal collector node
207	baseprimenode	Out	integer	Internal base node
208	emitprimenode	Out	integer	Internal emitter node
211	ic	Out	real	Current at collector node
212	ib	Out	real	Current at base node
236	ie	Out	real	Emitter current
237	is	Out	real	Substrate current
209	vbe	Out	real	B-E voltage
210	vbc	Out	real	B-C voltage
215	gm	Out	real	Small signal transconductance
213	gpi	Out	real	Small signal input conductance - pi
214	gmu	Out	real	Small signal conductance - mu
225	gx	Out	real	Conductance from base to internal base
216	go	Out	real	Small signal output conductance
227	geqcb	Out	real	d(Ibe)/d(Vbc)
228	gccs	Out	real	Internal C-S cap. equiv. cond.
229	geqbx	Out	real	Internal C-B-base cap. equiv. cond.
239	cpi	Out	real	Internal base to emitter capactance
240	cmu	Out	real	Internal base to collector capactiance
241	cbx	Out	real	Base to collector capacitance
242	ccs	Out	real	Collector to substrate capacitance
218	cqbe	Out	real	Cap. due to charge storage in B-E jct.
220	cqbc	Out	real	Cap. due to charge storage in B-C jct.
222	cqcs	Out	real	Cap. due to charge storage in C-S jct.
224	cqbx	Out	real	Cap. due to charge storage in B-X jct.
226	cexbc	Out	real	Total Capacitance in B-X junction
217	qbe	Out	real	Charge storage B-E junction
219	qbc	Out	real	Charge storage B-C junction
221	qcs	Out	real	Charge storage C-S junction
223	qbx	Out	real	Charge storage B-X junction
238	p	Out	real	Power dissipation
235	sens_dc	Out	real	dc sensitivity
230	sens_real	Out	real	real part of ac sensitivity
231	sens_imag	Out	real	dc sens. & imag part of ac sens.
232	sens_mag	Out	real	sensitivity of ac magnitude
233	sens_ph	Out	real	sensitivity of ac phase

234	sens_cplx	Out	complex	ac sensitivity
7	temp	InOut	real	instance temperature
8	dtemp	InOut	real	instance temperature delta from circuit

BJT model parameters

#	Name	Direction	Type	Description
309	type	Out	string	NPN or PNP
101	npn	InOut	flag	NPN type device
102	pnp	InOut	flag	PNP type device
103	is	InOut	real	Saturation Current
104	bf	InOut	real	Ideal forward beta
105	nf	InOut	real	Forward emission coefficient
106	vaf	InOut	real	Forward Early voltage
106	va	InOut	real	
107	ikf	InOut	real	Forward beta roll-off corner current
107	ik	InOut	real	
108	ise	InOut	real	B-E leakage saturation current
110	ne	InOut	real	B-E leakage emission coefficient
111	br	InOut	real	Ideal reverse beta
112	nr	InOut	real	Reverse emission coefficient
113	var	InOut	real	Reverse Early voltage
113	vb	InOut	real	
114	ikr	InOut	real	reverse beta roll-off corner current
115	isc	InOut	real	B-C leakage saturation current
117	nc	InOut	real	B-C leakage emission coefficient
118	rb	InOut	real	Zero bias base resistance
119	irb	InOut	real	Current for base resistance=(rb+rbm)/2
120	rbm	InOut	real	Minimum base resistance
121	re	InOut	real	Emitter resistance
122	rc	InOut	real	Collector resistance
123	cje	InOut	real	Zero bias B-E depletion capacitance
124	vje	InOut	real	B-E built in potential
124	pe	InOut	real	
125	mje	InOut	real	B-E junction grading coefficient
125	me	InOut	real	
126	tf	InOut	real	Ideal forward transit time
127	xtf	InOut	real	Coefficient for bias dependence of TF
128	vtf	InOut	real	Voltage giving VBC dependence of TF
129	itf	InOut	real	High current dependence of TF
130	ptf	InOut	real	Excess phase
131	cjc	InOut	real	Zero bias B-C depletion capacitance
132	vjc	InOut	real	B-C built in potential
132	pc	InOut	real	
133	mjc	InOut	real	B-C junction grading coefficient
133	mc	InOut	real	
134	xcjc	InOut	real	Fraction of B-C cap to internal base
135	tr	InOut	real	Ideal reverse transit time
136	cjs	InOut	real	Zero bias C-S capacitance
136	ccs	InOut	real	Zero bias C-S capacitance
137	vjs	InOut	real	Substrate junction built in potential
137	ps	InOut	real	
138	mjs	InOut	real	Substrate junction grading coefficient
138	ms	InOut	real	
139	xtb	InOut	real	Forward and reverse beta temp. exp.

140	eg	InOut	real	Energy gap for IS temp. dependency
141	xti	InOut	real	Temp. exponent for IS
142	fc	InOut	real	Forward bias junction fit parameter
301	invearlyvoltf	Out	real	Inverse early voltage:forward
302	invearlyvoltr	Out	real	Inverse early voltage:reverse
303	invrollofff	Out	real	Inverse roll off - forward
304	invrolloffr	Out	real	Inverse roll off - reverse
305	collectorconduct	Out	real	Collector conductance
306	emitterconduct	Out	real	Emitter conductance
307	transtimevbcfact	Out	real	Transit time VBC factor
308	excessphasefactor	Out	real	Excess phase fact.
143	tnom	InOut	real	Parameter measurement temperature
145	kf	InOut	real	Flicker Noise Coefficient
144	af	InOut	real	Flicker Noise Exponent

30.4.2   BJT - Bipolar Junction Transistor Level 2

BJT2 instance parameters

#	Name	Direction	Type	Description
2	off	InOut	flag	Device initially off
3	icvbe	InOut	real	Initial B-E voltage
4	icvce	InOut	real	Initial C-E voltage
1	area	InOut	real	(Emitter) Area factor
10	areab	InOut	real	Base area factor
11	areac	InOut	real	Collector area factor
9	m	InOut	real	Parallel Multiplier
5	ic	In	real vector	Initial condition vector
6	sens_area	In	flag	flag to request sensitivity WRT area
202	colnode	Out	integer	Number of collector node
203	basenode	Out	integer	Number of base node
204	emitnode	Out	integer	Number of emitter node
205	substnode	Out	integer	Number of substrate node
206	colprimenode	Out	integer	Internal collector node
207	baseprimenode	Out	integer	Internal base node
208	emitprimenode	Out	integer	Internal emitter node
211	ic	Out	real	Current at collector node
212	ib	Out	real	Current at base node
236	ie	Out	real	Emitter current
237	is	Out	real	Substrate current
209	vbe	Out	real	B-E voltage
210	vbc	Out	real	B-C voltage
215	gm	Out	real	Small signal transconductance
213	gpi	Out	real	Small signal input conductance - pi
214	gmu	Out	real	Small signal conductance - mu
225	gx	Out	real	Conductance from base to internal base
216	go	Out	real	Small signal output conductance
227	geqcb	Out	real	d(Ibe)/d(Vbc)
228	gcsub	Out	real	Internal Subs. cap. equiv. cond.
243	gdsub	Out	real	Internal Subs. Diode equiv. cond.
229	geqbx	Out	real	Internal C-B-base cap. equiv. cond.
239	cpi	Out	real	Internal base to emitter capactance
240	cmu	Out	real	Internal base to collector capactiance
241	cbx	Out	real	Base to collector capacitance
242	csub	Out	real	Substrate capacitance
218	cqbe	Out	real	Cap. due to charge storage in B-E jct.
220	cqbc	Out	real	Cap. due to charge storage in B-C jct.
222	cqsub	Out	real	Cap. due to charge storage in Subs. jct.
224	cqbx	Out	real	Cap. due to charge storage in B-X jct.
226	cexbc	Out	real	Total Capacitance in B-X junction
217	qbe	Out	real	Charge storage B-E junction
219	qbc	Out	real	Charge storage B-C junction
221	qsub	Out	real	Charge storage Subs. junction
223	qbx	Out	real	Charge storage B-X junction
238	p	Out	real	Power dissipation
235	sens_dc	Out	real	dc sensitivity
230	sens_real	Out	real	real part of ac sensitivity
231	sens_imag	Out	real	dc sens. & imag part of ac sens.
232	sens_mag	Out	real	sensitivity of ac magnitude
233	sens_ph	Out	real	sensitivity of ac phase
234	sens_cplx	Out	complex	ac sensitivity

| 7 | temp | InOut | real | instance temperature |
| 8 | dtemp | InOut | real | instance temperature delta from circuit |

BJT2 model parameters

#	Name	Direction	Type	Description
309	type	Out	string	NPN or PNP
101	npn	InOut	flag	NPN type device
102	pnp	InOut	flag	PNP type device
147	subs	InOut	integer	Vertical or Lateral device
103	is	InOut	real	Saturation Current
146	iss	InOut	real	Substrate Jct. Saturation Current
104	bf	InOut	real	Ideal forward beta
105	nf	InOut	real	Forward emission coefficient
106	vaf	InOut	real	Forward Early voltage
106	va	InOut	real	
107	ikf	InOut	real	Forward beta roll-off corner current
107	ik	InOut	real	
108	ise	InOut	real	B-E leakage saturation current
110	ne	InOut	real	B-E leakage emission coefficient
111	br	InOut	real	Ideal reverse beta
112	nr	InOut	real	Reverse emission coefficient
113	var	InOut	real	Reverse Early voltage
113	vb	InOut	real	
114	ikr	InOut	real	reverse beta roll-off corner current
115	isc	InOut	real	B-C leakage saturation current
117	nc	InOut	real	B-C leakage emission coefficient
118	rb	InOut	real	Zero bias base resistance
119	irb	InOut	real	Current for base resistance=(rb+rbm)/2
120	rbm	InOut	real	Minimum base resistance
121	re	InOut	real	Emitter resistance
122	rc	InOut	real	Collector resistance
123	cje	InOut	real	Zero bias B-E depletion capacitance
124	vje	InOut	real	B-E built in potential
124	pe	InOut	real	
125	mje	InOut	real	B-E junction grading coefficient
125	me	InOut	real	
126	tf	InOut	real	Ideal forward transit time
127	xtf	InOut	real	Coefficient for bias dependence of TF
128	vtf	InOut	real	Voltage giving VBC dependence of TF
129	itf	InOut	real	High current dependence of TF
130	ptf	InOut	real	Excess phase
131	cjc	InOut	real	Zero bias B-C depletion capacitance
132	vjc	InOut	real	B-C built in potential
132	pc	InOut	real	
133	mjc	InOut	real	B-C junction grading coefficient
133	mc	InOut	real	
134	xcjc	InOut	real	Fraction of B-C cap to internal base
135	tr	InOut	real	Ideal reverse transit time
136	cjs	InOut	real	Zero bias Substrate capacitance
136	csub	InOut	real	
137	vjs	InOut	real	Substrate junction built in potential
137	ps	InOut	real	
138	mjs	InOut	real	Substrate junction grading coefficient
138	ms	InOut	real	

139	xtb	InOut	real	Forward and reverse beta temp. exp.
140	eg	InOut	real	Energy gap for IS temp. dependency
141	xti	InOut	real	Temp. exponent for IS
148	tre1	InOut	real	Temp. coefficient 1 for RE
149	tre2	InOut	real	Temp. coefficient 2 for RE
150	trc1	InOut	real	Temp. coefficient 1 for RC
151	trc2	InOut	real	Temp. coefficient 2 for RC
152	trb1	InOut	real	Temp. coefficient 1 for RB
153	trb2	InOut	real	Temp. coefficient 2 for RB
154	trbm1	InOut	real	Temp. coefficient 1 for RBM
155	trbm2	InOut	real	Temp. coefficient 2 for RBM
142	fc	InOut	real	Forward bias junction fit parameter
301	invearlyvoltf	Out	real	Inverse early voltage:forward
302	invearlyvoltr	Out	real	Inverse early voltage:reverse
303	invrollofff	Out	real	Inverse roll off - forward
304	invrolloffr	Out	real	Inverse roll off - reverse
305	collectorconduct	Out	real	Collector conductance
306	emitterconduct	Out	real	Emitter conductance
307	transtimevbcfact	Out	real	Transit time VBC factor
308	excessphasefactor	Out	real	Excess phase fact.
143	tnom	InOut	real	Parameter measurement temperature
145	kf	InOut	real	Flicker Noise Coefficient
144	af	InOut	real	Flicker Noise Exponent

30.4.3   VBIC - Vertical Bipolar Inter-Company Model

VBIC instance parameters

#	Name	Direction	Type	Description
1	area	InOut	real	Area factor
2	off	InOut	flag	Device initially off
3	ic	In	real vector	Initial condition vector
4	icvbe	InOut	real	Initial B-E voltage
5	icvce	InOut	real	Initial C-E voltage
6	temp	InOut	real	Instance temperature
7	dtemp	InOut	real	Instance delta temperature
8	m	InOut	real	Multiplier
212	collnode	Out	integer	Number of collector node
213	basenode	Out	integer	Number of base node
214	emitnode	Out	integer	Number of emitter node
215	subsnode	Out	integer	Number of substrate node
216	collCXnode	Out	integer	Internal collector node
217	collCInode	Out	integer	Internal collector node
218	baseBXnode	Out	integer	Internal base node
219	baseBInode	Out	integer	Internal base node
220	baseBPnode	Out	integer	Internal base node
221	emitEInode	Out	integer	Internal emitter node
222	subsSInode	Out	integer	Internal substrate node
223	vbe	Out	real	B-E voltage
224	vbc	Out	real	B-C voltage
225	ic	Out	real	Collector current
226	ib	Out	real	Base current
227	ie	Out	real	Emitter current
228	is	Out	real	Substrate current
229	gm	Out	real	Small signal transconductance dIc/dVbe
230	go	Out	real	Small signal output conductance dIc/dVbc
231	gpi	Out	real	Small signal input conductance dIb/dVbe
232	gmu	Out	real	Small signal conductance dIb/dVbc
233	gx	Out	real	Conductance from base to internal base
247	cbe	Out	real	Internal base to emitter capacitance
248	cbex	Out	real	External base to emitter capacitance
249	cbc	Out	real	Internal base to collector capacitance
250	cbcx	Out	real	External Base to collector capacitance
251	cbep	Out	real	Parasitic Base to emitter capacitance
252	cbcp	Out	real	Parasitic Base to collector capacitance
259	p	Out	real	Power dissipation
243	geqcb	Out	real	Internal C-B-base cap. equiv. cond.
246	geqbx	Out	real	External C-B-base cap. equiv. cond.
234	qbe	Out	real	Charge storage B-E junction
235	cqbe	Out	real	Cap. due to charge storage in B-E jct.
236	qbc	Out	real	Charge storage B-C junction
237	cqbc	Out	real	Cap. due to charge storage in B-C jct.
238	qbx	Out	real	Charge storage B-X junction
239	cqbx	Out	real	Cap. due to charge storage in B-X jct.
258	sens_dc	Out	real	DC sensitivity
253	sens_real	Out	real	Real part of AC sensitivity
254	sens_imag	Out	real	DC sens. & imag part of AC sens.
255	sens_mag	Out	real	Sensitivity of AC magnitude
256	sens_ph	Out	real	Sensitivity of AC phase
257	sens_cplx	Out	complex	AC sensitivity

VBIC model parameters

#	Name	Direction	Type	Description
305	type	Out	string	NPN or PNP
101	npn	InOut	flag	NPN type device
102	pnp	InOut	flag	PNP type device
103	tnom	InOut	real	Parameter measurement temperature
104	rcx	InOut	real	Extrinsic coll resistance
105	rci	InOut	real	Intrinsic coll resistance
106	vo	InOut	real	Epi drift saturation voltage
107	gamm	InOut	real	Epi doping parameter
108	hrcf	InOut	real	High current RC factor
109	rbx	InOut	real	Extrinsic base resistance
110	rbi	InOut	real	Intrinsic base resistance
111	re	InOut	real	Intrinsic emitter resistance
112	rs	InOut	real	Intrinsic substrate resistance
113	rbp	InOut	real	Parasitic base resistance
114	is	InOut	real	Transport saturation current
115	nf	InOut	real	Forward emission coefficient
116	nr	InOut	real	Reverse emission coefficient
117	fc	InOut	real	Fwd bias depletion capacitance limit
118	cbeo	InOut	real	Extrinsic B-E overlap capacitance
119	cje	InOut	real	Zero bias B-E depletion capacitance
120	pe	InOut	real	B-E built in potential
121	me	InOut	real	B-E junction grading coefficient
122	aje	InOut	real	B-E capacitance smoothing factor
123	cbco	InOut	real	Extrinsic B-C overlap capacitance
124	cjc	InOut	real	Zero bias B-C depletion capacitance
125	qco	InOut	real	Epi charge parameter
126	cjep	InOut	real	B-C extrinsic zero bias capacitance
127	pc	InOut	real	B-C built in potential
128	mc	InOut	real	B-C junction grading coefficient
129	ajc	InOut	real	B-C capacitance smoothing factor
130	cjcp	InOut	real	Zero bias S-C capacitance
131	ps	InOut	real	S-C junction built in potential
132	ms	InOut	real	S-C junction grading coefficient
133	ajs	InOut	real	S-C capacitance smoothing factor
134	ibei	InOut	real	Ideal B-E saturation current
135	wbe	InOut	real	Portion of IBEI from Vbei, 1-WBE from Vbex
136	nei	InOut	real	Ideal B-E emission coefficient
137	iben	InOut	real	Non-ideal B-E saturation current
138	nen	InOut	real	Non-ideal B-E emission coefficient
139	ibci	InOut	real	Ideal B-C saturation current
140	nci	InOut	real	Ideal B-C emission coefficient
141	ibcn	InOut	real	Non-ideal B-C saturation current
142	ncn	InOut	real	Non-ideal B-C emission coefficient
143	avc1	InOut	real	B-C weak avalanche parameter 1
144	avc2	InOut	real	B-C weak avalanche parameter 2
145	isp	InOut	real	Parasitic transport saturation current
146	wsp	InOut	real	Portion of ICCP
147	nfp	InOut	real	Parasitic fwd emission coefficient
148	ibeip	InOut	real	Ideal parasitic B-E saturation current
149	ibenp	InOut	real	Non-ideal parasitic B-E saturation current
150	ibcip	InOut	real	Ideal parasitic B-C saturation current
151	ncip	InOut	real	Ideal parasitic B-C emission coefficient
152	ibcnp	InOut	real	Nonideal parasitic B-C saturation current

153	ncnp	InOut	real	Nonideal parasitic B-C emission coefficient
154	vef	InOut	real	Forward Early voltage
155	ver	InOut	real	Reverse Early voltage
156	ikf	InOut	real	Forward knee current
157	ikr	InOut	real	Reverse knee current
158	ikp	InOut	real	Parasitic knee current
159	tf	InOut	real	Ideal forward transit time
160	qtf	InOut	real	Variation of TF with base-width modulation
161	xtf	InOut	real	Coefficient for bias dependence of TF
162	vtf	InOut	real	Voltage giving VBC dependence of TF
163	itf	InOut	real	High current dependence of TF
164	tr	InOut	real	Ideal reverse transit time
165	td	InOut	real	Forward excess-phase delay time
166	kfn	InOut	real	B-E Flicker Noise Coefficient
167	afn	InOut	real	B-E Flicker Noise Exponent
168	bfn	InOut	real	B-E Flicker Noise 1/f dependence
169	xre	InOut	real	Temperature exponent of RE
170	xrb	InOut	real	Temperature exponent of RB
171	xrbi	InOut	real	Temperature exponent of RBI
172	xrc	InOut	real	Temperature exponent of RC
173	xrci	InOut	real	Temperature exponent of RCI
174	xrs	InOut	real	Temperature exponent of RS
175	xvo	InOut	real	Temperature exponent of VO
176	ea	InOut	real	Activation energy for IS
177	eaie	InOut	real	Activation energy for IBEI
179	eaic	InOut	real	Activation energy for IBCI/IBEIP
179	eais	InOut	real	Activation energy for IBCIP
180	eane	InOut	real	Activation energy for IBEN
181	eanc	InOut	real	Activation energy for IBCN/IBENP
182	eans	InOut	real	Activation energy for IBCNP
183	xis	InOut	real	Temperature exponent of IS
184	xii	InOut	real	Temperature exponent of IBEI,IBCI,IBEIP,IBCIP
185	xin	InOut	real	Temperature exponent of IBEN,IBCN,IBENP,IBCNP
186	tnf	InOut	real	Temperature exponent of NF
187	tavc	InOut	real	Temperature exponent of AVC2
188	rth	InOut	real	Thermal resistance
189	cth	InOut	real	Thermal capacitance
190	vrt	InOut	real	Punch-through voltage of internal B-C junction
191	art	InOut	real	Smoothing parameter for reach-through
192	ccso	InOut	real	Fixed C-S capacitance
193	qbm	InOut	real	Select SGP qb formulation
194	nkf	InOut	real	High current beta rolloff
195	xikf	InOut	real	Temperature exponent of IKF
196	xrcx	InOut	real	Temperature exponent of RCX
197	xrbx	InOut	real	Temperature exponent of RBX
198	xrbp	InOut	real	Temperature exponent of RBP
199	isrr	InOut	real	Separate IS for fwd and rev
200	xisr	InOut	real	Temperature exponent of ISR
201	dear	InOut	real	Delta activation energy for ISRR
202	eap	InOut	real	Exitivation energy for ISP
203	vbbe	InOut	real	B-E breakdown voltage
204	nbbe	InOut	real	B-E breakdown emission coefficient
205	ibbe	InOut	real	B-E breakdown current
206	tvbbe1	InOut	real	Linear temperature coefficient of VBBE
207	tvbbe2	InOut	real	Quadratic temperature coefficient of VBBE

208	tnbbe	InOut	real	Temperature coefficient of NBBE
209	ebbe	InOut	real	exp(-VBBE/(NBBE*Vtv))
210	dtemp	InOut	real	Locale Temperature difference
211	vers	InOut	real	Revision Version
212	vref	InOut	real	Reference Version

30.5 MOSFETs

30.5.1 MOS1 - Level 1 MOSFET model with Meyer capacitance model

MOS1 instance parameters

#	Name	Direction	Type	Description
21	m	InOut	real	Multiplier
2	l	InOut	real	Length
1	w	InOut	real	Width
4	ad	InOut	real	Drain area
3	as	InOut	real	Source area
6	pd	InOut	real	Drain perimeter
5	ps	InOut	real	Source perimeter
8	nrd	InOut	real	Drain squares
7	nrs	InOut	real	Source squares
9	off	In	flag	Device initially off
12	icvds	InOut	real	Initial D-S voltage
13	icvgs	InOut	real	Initial G-S voltage
11	icvbs	InOut	real	Initial B-S voltage
20	temp	InOut	real	Instance temperature
22	dtemp	InOut	real	Instance temperature difference
10	ic	In	real vector	Vector of D-S, G-S, B-S voltages
15	sens_l	In	flag	flag to request sensitivity WRT length
14	sens_w	In	flag	flag to request sensitivity WRT width
215	id	Out	real	Drain current
18	is	Out	real	Source current
17	ig	Out	real	Gate current
16	ib	Out	real	Bulk current
217	ibd	Out	real	B-D junction current
216	ibs	Out	real	B-S junction current
231	vgs	Out	real	Gate-Source voltage
232	vds	Out	real	Drain-Source voltage
230	vbs	Out	real	Bulk-Source voltage
229	vbd	Out	real	Bulk-Drain voltage
203	dnode	Out	integer	Number of the drain node
204	gnode	Out	integer	Number of the gate node
205	snode	Out	integer	Number of the source node
206	bnode	Out	integer	Number of the node
207	dnodeprime	Out	integer	Number of int. drain node
208	snodeprime	Out	integer	Number of int. source node
211	von	Out	real	
212	vdsat	Out	real	Saturation drain voltage
213	sourcevcrit	Out	real	Critical source voltage
214	drainvcrit	Out	real	Critical drain voltage
258	rs	Out	real	Source resistance
209	sourceconductance	Out	real	Conductance of source
259	rd	Out	real	Drain conductance
210	drainconductance	Out	real	Conductance of drain
219	gm	Out	real	Transconductance
220	gds	Out	real	Drain-Source conductance
218	gmb	Out	real	Bulk-Source transconductance
218	gmbs	Out	real	
221	gbd	Out	real	Bulk-Drain conductance
222	gbs	Out	real	Bulk-Source conductance
223	cbd	Out	real	Bulk-Drain capacitance
#	Name	Direction	Type	Description

#	Name	Direction	Type	Description
224	cbs	Out	real	Bulk-Source capacitance
233	cgs	Out	real	Gate-Source capacitance
236	cgd	Out	real	Gate-Drain capacitance
239	cgb	Out	real	Gate-Bulk capacitance
235	cqgs	Out	real	Capacitance due to gate-source charge storage
238	cqgd	Out	real	Capacitance due to gate-drain charge storage
241	cqgb	Out	real	Capacitance due to gate-bulk charge storage
243	cqbd	Out	real	Capacitance due to bulk-drain charge storage
245	cqbs	Out	real	Capacitance due to bulk-source charge storage
225	cbd0	Out	real	Zero-Bias B-D junction capacitance
226	cbdsw0	Out	real	
227	cbs0	Out	real	Zero-Bias B-S junction capacitance
228	cbssw0	Out	real	
234	qgs	Out	real	Gate-Source charge storage
237	qgd	Out	real	Gate-Drain charge storage
240	qgb	Out	real	Gate-Bulk charge storage
242	qbd	Out	real	Bulk-Drain charge storage
244	qbs	Out	real	Bulk-Source charge storage
19	p	Out	real	Instaneous power
256	sens_l_dc	Out	real	dc sensitivity wrt length
246	sens_l_real	Out	real	real part of ac sensitivity wrt length
247	sens_l_imag	Out	real	imag part of ac sensitivity wrt length
248	sens_l_mag	Out	real	sensitivity wrt l of ac magnitude
249	sens_l_ph	Out	real	sensitivity wrt l of ac phase
250	sens_l_cplx	Out	complex	ac sensitivity wrt length
257	sens_w_dc	Out	real	dc sensitivity wrt width
251	sens_w_real	Out	real	real part of ac sensitivity wrt width
252	sens_w_imag	Out	real	imag part of ac sensitivity wrt width
253	sens_w_mag	Out	real	sensitivity wrt w of ac magnitude
254	sens_w_ph	Out	real	sensitivity wrt w of ac phase
255	sens_w_cplx	Out	complex	ac sensitivity wrt width
#	Name	Direction	Type	Description

MOS1 model parameters

#	Name	Direction	Type	Description
133	type	Out	string	N-channel or P-channel MOS
101	vto	InOut	real	Threshold voltage
101	vt0	InOut	real	
102	kp	InOut	real	Transconductance parameter
103	gamma	InOut	real	Bulk threshold parameter
104	phi	InOut	real	Surface potential
105	lambda	InOut	real	Channel length modulation
106	rd	InOut	real	Drain ohmic resistance
107	rs	InOut	real	Source ohmic resistance
108	cbd	InOut	real	B-D junction capacitance
109	cbs	InOut	real	B-S junction capacitance
110	is	InOut	real	Bulk junction sat. current
111	pb	InOut	real	Bulk junction potential
112	cgso	InOut	real	Gate-source overlap cap.
113	cgdo	InOut	real	Gate-drain overlap cap.
114	cgbo	InOut	real	Gate-bulk overlap cap.
122	rsh	InOut	real	Sheet resistance
115	cj	InOut	real	Bottom junction cap per area
116	mj	InOut	real	Bottom grading coefficient

117	cjsw	InOut	real	Side junction cap per area
118	mjsw	InOut	real	Side grading coefficient
119	js	InOut	real	Bulk jct. sat. current density
120	tox	InOut	real	Oxide thickness
121	ld	InOut	real	Lateral diffusion
123	u0	InOut	real	Surface mobility
123	uo	InOut	real	
124	fc	InOut	real	Forward bias jct. fit parm.
128	nmos	In	flag	N type MOSfet model
129	pmos	In	flag	P type MOSfet model
125	nsub	InOut	real	Substrate doping
126	tpg	InOut	integer	Gate type
127	nss	InOut	real	Surface state density
130	tnom	InOut	real	Parameter measurement temperature
131	kf	InOut	real	Flicker noise coefficient
132	af	InOut	real	Flicker noise exponent

30.5.2   MOS2 - Level 2 MOSFET model with Meyer capacitance model

MOS 2 instance parameters

#	Name	Direction	Type	Description
80	m	InOut	real	Multiplier
2	l	InOut	real	Length
1	w	InOut	real	Width
4	ad	InOut	real	Drain area
3	as	InOut	real	Source area
6	pd	InOut	real	Drain perimeter
5	ps	InOut	real	Source perimeter
34	id	Out	real	Drain current
34	cd	Out	real	
36	ibd	Out	real	B-D junction current
35	ibs	Out	real	B-S junction current
18	is	Out	real	Source current
17	ig	Out	real	Gate current
16	ib	Out	real	Bulk current
50	vgs	Out	real	Gate-Source voltage
51	vds	Out	real	Drain-Source voltage
49	vbs	Out	real	Bulk-Source voltage
48	vbd	Out	real	Bulk-Drain voltage
8	nrd	InOut	real	Drain squares
7	nrs	InOut	real	Source squares
9	off	In	flag	Device initially off
12	icvds	InOut	real	Initial D-S voltage
13	icvgs	InOut	real	Initial G-S voltage
11	icvbs	InOut	real	Initial B-S voltage
77	temp	InOut	real	Instance operating temperature
81	dtemp	InOut	real	Instance temperature difference
10	ic	In	real vector	Vector of D-S, G-S, B-S voltages
15	sens_l	In	flag	flag to request sensitivity WRT length
14	sens_w	In	flag	flag to request sensitivity WRT width
22	dnode	Out	integer	Number of drain node
23	gnode	Out	integer	Number of gate node
24	snode	Out	integer	Number of source node
25	bnode	Out	integer	Number of bulk node
26	dnodeprime	Out	integer	Number of internal drain node
27	snodeprime	Out	integer	Number of internal source node
30	von	Out	real	
31	vdsat	Out	real	Saturation drain voltage
32	sourcevcrit	Out	real	Critical source voltage
33	drainvcrit	Out	real	Critical drain voltage
78	rs	Out	real	Source resistance
28	sourceconductance	Out	real	Source conductance
79	rd	Out	real	Drain resistance
29	drainconductance	Out	real	Drain conductance
38	gm	Out	real	Transconductance
39	gds	Out	real	Drain-Source conductance
37	gmb	Out	real	Bulk-Source transconductance
37	gmbs	Out	real	
40	gbd	Out	real	Bulk-Drain conductance
41	gbs	Out	real	Bulk-Source conductance
42	cbd	Out	real	Bulk-Drain capacitance
43	cbs	Out	real	Bulk-Source capacitance

52	cgs	Out	real	Gate-Source capacitance
55	cgd	Out	real	Gate-Drain capacitance
58	cgb	Out	real	Gate-Bulk capacitance
44	cbd0	Out	real	Zero-Bias B-D junction capacitance
45	cbdsw0	Out	real	
46	cbs0	Out	real	Zero-Bias B-S junction capacitance
47	cbssw0	Out	real	
54	cqgs	Out	real	Capacitance due to gate-source charge storage
57	cqgd	Out	real	Capacitance due to gate-drain charge storage
60	cqgb	Out	real	Capacitance due to gate-bulk charge storage
62	cqbd	Out	real	Capacitance due to bulk-drain charge storage
64	cqbs	Out	real	Capacitance due to bulk-source charge storage
53	qgs	Out	real	Gate-Source charge storage
56	qgd	Out	real	Gate-Drain charge storage
59	qgb	Out	real	Gate-Bulk charge storage
61	qbd	Out	real	Bulk-Drain charge storage
63	qbs	Out	real	Bulk-Source charge storage
19	p	Out	real	Instantaneous power
75	sens_l_dc	Out	real	dc sensitivity wrt length
70	sens_l_real	Out	real	real part of ac sensitivity wrt length
71	sens_l_imag	Out	real	imag part of ac sensitivity wrt length
74	sens_l_cplx	Out	complex	ac sensitivity wrt length
72	sens_l_mag	Out	real	sensitivity wrt l of ac magnitude
73	sens_l_ph	Out	real	sensitivity wrt l of ac phase
76	sens_w_dc	Out	real	dc sensitivity wrt width
65	sens_w_real	Out	real	dc sensitivity and real part of ac sensitivity wrt width
66	sens_w_imag	Out	real	imag part of ac sensitivity wrt width
67	sens_w_mag	Out	real	sensitivity wrt w of ac magnitude
68	sens_w_ph	Out	real	sensitivity wrt w of ac phase
69	sens_w_cplx	Out	complex	ac sensitivity wrt width

MOS2 model parameters

#	Name	Direction	Type	Description
141	type	Out	string	N-channel or P-channel MOS
101	vto	InOut	real	Threshold voltage
101	vt0	InOut	real	
102	kp	InOut	real	Transconductance parameter
103	gamma	InOut	real	Bulk threshold parameter
104	phi	InOut	real	Surface potential
105	lambda	InOut	real	Channel length modulation
106	rd	InOut	real	Drain ohmic resistance
107	rs	InOut	real	Source ohmic resistance
108	cbd	InOut	real	B-D junction capacitance
109	cbs	InOut	real	B-S junction capacitance
110	is	InOut	real	Bulk junction sat. current
111	pb	InOut	real	Bulk junction potential
112	cgso	InOut	real	Gate-source overlap cap.
113	cgdo	InOut	real	Gate-drain overlap cap.
114	cgbo	InOut	real	Gate-bulk overlap cap.
122	rsh	InOut	real	Sheet resistance
115	cj	InOut	real	Bottom junction cap per area
116	mj	InOut	real	Bottom grading coefficient
117	cjsw	InOut	real	Side junction cap per area
118	mjsw	InOut	real	Side grading coefficient

119	js	InOut	real	Bulk jct. sat. current density
120	tox	InOut	real	Oxide thickness
121	ld	InOut	real	Lateral diffusion
123	u0	InOut	real	Surface mobility
123	uo	InOut	real	
124	fc	InOut	real	Forward bias jct. fit parm.
135	nmos	In	flag	N type MOSfet model
136	pmos	In	flag	P type MOSfet model
125	nsub	InOut	real	Substrate doping
126	tpg	InOut	integer	Gate type
127	nss	InOut	real	Surface state density
129	delta	InOut	real	Width effect on threshold
130	uexp	InOut	real	Crit. field exp for mob. deg.
134	ucrit	InOut	real	Crit. field for mob. degradation
131	vmax	InOut	real	Maximum carrier drift velocity
132	xj	InOut	real	Junction depth
133	neff	InOut	real	Total channel charge coeff.
128	nfs	InOut	real	Fast surface state density
137	tnom	InOut	real	Parameter measurement temperature
139	kf	InOut	real	Flicker noise coefficient
140	af	InOut	real	Flicker noise exponent

30.5.3   MOS3 - Level 3 MOSFET model with Meyer capacitance model

MOS3 instance parameters

#	Name	Direction	Type	Description
80	m	InOut	real	Multiplier
2	l	InOut	real	Length
1	w	InOut	real	Width
4	ad	InOut	real	Drain area
3	as	InOut	real	Source area
6	pd	InOut	real	Drain perimeter
5	ps	InOut	real	Source perimeter
34	id	Out	real	Drain current
34	cd	Out	real	Drain current
36	ibd	Out	real	B-D junction current
35	ibs	Out	real	B-S junction current
18	is	Out	real	Source current
17	ig	Out	real	Gate current
16	ib	Out	real	Bulk current
50	vgs	Out	real	Gate-Source voltage
51	vds	Out	real	Drain-Source voltage
49	vbs	Out	real	Bulk-Source voltage
48	vbd	Out	real	Bulk-Drain voltage
8	nrd	InOut	real	Drain squares
7	nrs	InOut	real	Source squares
9	off	In	flag	Device initially off
12	icvds	InOut	real	Initial D-S voltage
13	icvgs	InOut	real	Initial G-S voltage
11	icvbs	InOut	real	Initial B-S voltage
10	ic	InOut	real vector	Vector of D-S, G-S, B-S voltages
77	temp	InOut	real	Instance operating temperature
81	dtemp	InOut	real	Instance temperature difference
15	sens_l	In	flag	flag to request sensitivity WRT length
14	sens_w	In	flag	flag to request sensitivity WRT width
22	dnode	Out	integer	Number of drain node
23	gnode	Out	integer	Number of gate node
24	snode	Out	integer	Number of source node
25	bnode	Out	integer	Number of bulk node
26	dnodeprime	Out	integer	Number of internal drain node
27	snodeprime	Out	integer	Number of internal source node
30	von	Out	real	Turn-on voltage
31	vdsat	Out	real	Saturation drain voltage
32	sourcevcrit	Out	real	Critical source voltage
33	drainvcrit	Out	real	Critical drain voltage
78	rs	Out	real	Source resistance
28	sourceconductance	Out	real	Source conductance
79	rd	Out	real	Drain resistance
29	drainconductance	Out	real	Drain conductance
38	gm	Out	real	Transconductance
39	gds	Out	real	Drain-Source conductance
37	gmb	Out	real	Bulk-Source transconductance
37	gmbs	Out	real	Bulk-Source transconductance
40	gbd	Out	real	Bulk-Drain conductance
41	gbs	Out	real	Bulk-Source conductance
42	cbd	Out	real	Bulk-Drain capacitance
43	cbs	Out	real	Bulk-Source capacitance

52	cgs	Out	real	Gate-Source capacitance
55	cgd	Out	real	Gate-Drain capacitance
58	cgb	Out	real	Gate-Bulk capacitance
54	cqgs	Out	real	Capacitance due to gate-source charge storage
57	cqgd	Out	real	Capacitance due to gate-drain charge storage
60	cqgb	Out	real	Capacitance due to gate-bulk charge storage
62	cqbd	Out	real	Capacitance due to bulk-drain charge storage
64	cqbs	Out	real	Capacitance due to bulk-source charge storage
44	cbd0	Out	real	Zero-Bias B-D junction capacitance
45	cbdsw0	Out	real	Zero-Bias B-D sidewall capacitance
46	cbs0	Out	real	Zero-Bias B-S junction capacitance
47	cbssw0	Out	real	Zero-Bias B-S sidewall capacitance
63	qbs	Out	real	Bulk-Source charge storage
53	qgs	Out	real	Gate-Source charge storage
56	qgd	Out	real	Gate-Drain charge storage
59	qgb	Out	real	Gate-Bulk charge storage
61	qbd	Out	real	Bulk-Drain charge storage
19	p	Out	real	Instantaneous power
76	sens_l_dc	Out	real	dc sensitivity wrt length
70	sens_l_real	Out	real	real part of ac sensitivity wrt length
71	sens_l_imag	Out	real	imag part of ac sensitivity wrt length
74	sens_l_cplx	Out	complex	ac sensitivity wrt length
72	sens_l_mag	Out	real	sensitivity wrt l of ac magnitude
73	sens_l_ph	Out	real	sensitivity wrt l of ac phase
75	sens_w_dc	Out	real	dc sensitivity wrt width
65	sens_w_real	Out	real	real part of ac sensitivity wrt width
66	sens_w_imag	Out	real	imag part of ac sensitivity wrt width
67	sens_w_mag	Out	real	sensitivity wrt w of ac magnitude
68	sens_w_ph	Out	real	sensitivity wrt w of ac phase
69	sens_w_cplx	Out	complex	ac sensitivity wrt width

MOS3 model parameters

#	Name	Direction	Type	Description
144	type	Out	string	N-channel or P-channel MOS
133	nmos	In	flag	N type MOSfet model
134	pmos	In	flag	P type MOSfet model
101	vto	InOut	real	Threshold voltage
101	vt0	InOut	real	
102	kp	InOut	real	Transconductance parameter
103	gamma	InOut	real	Bulk threshold parameter
104	phi	InOut	real	Surface potential
105	rd	InOut	real	Drain ohmic resistance
106	rs	InOut	real	Source ohmic resistance
107	cbd	InOut	real	B-D junction capacitance
108	cbs	InOut	real	B-S junction capacitance
109	is	InOut	real	Bulk junction sat. current
110	pb	InOut	real	Bulk junction potential
111	cgso	InOut	real	Gate-source overlap cap.
112	cgdo	InOut	real	Gate-drain overlap cap.
113	cgbo	InOut	real	Gate-bulk overlap cap.
114	rsh	InOut	real	Sheet resistance
115	cj	InOut	real	Bottom junction cap per area
116	mj	InOut	real	Bottom grading coefficient
117	cjsw	InOut	real	Side junction cap per area
118	mjsw	InOut	real	Side grading coefficient
119	js	InOut	real	Bulk jct. sat. current density
120	tox	InOut	real	Oxide thickness
121	ld	InOut	real	Lateral diffusion
145	xl	InOut	real	Length mask adjustment
146	wd	InOut	real	Width Narrowing (Diffusion)
147	xw	InOut	real	Width mask adjustment
148	delvto	InOut	real	Threshold voltage Adjust
148	delvt0	InOut	real	
122	u0	InOut	real	Surface mobility
122	uo	InOut	real	
123	fc	InOut	real	Forward bias jct. fit parm.
124	nsub	InOut	real	Substrate doping
125	tpg	InOut	integer	Gate type
126	nss	InOut	real	Surface state density
131	vmax	InOut	real	Maximum carrier drift velocity
135	xj	InOut	real	Junction depth
129	nfs	InOut	real	Fast surface state density
138	xd	InOut	real	Depletion layer width
139	alpha	InOut	real	Alpha
127	eta	InOut	real	Vds dependence of threshold voltage
128	delta	InOut	real	Width effect on threshold
140	input_delta	InOut	real	
130	theta	InOut	real	Vgs dependence on mobility
132	kappa	InOut	real	Kappa
141	tnom	InOut	real	Parameter measurement temperature
142	kf	InOut	real	Flicker noise coefficient
143	af	InOut	real	Flicker noise exponent

30.5.4   MOS6 - Level 6 MOSFET model with Meyer capacitance model

MOS6 instance parameters

#	Name	Direction	Type	Description
2	l	InOut	real	Length
1	w	InOut	real	Width
22	m	InOut	real	Parallel Multiplier
4	ad	InOut	real	Drain area
3	as	InOut	real	Source area
6	pd	InOut	real	Drain perimeter
5	ps	InOut	real	Source perimeter
215	id	Out	real	Drain current
215	cd	Out	real	Drain current
18	is	Out	real	Source current
17	ig	Out	real	Gate current
16	ib	Out	real	Bulk current
216	ibs	Out	real	B-S junction capacitance
217	ibd	Out	real	B-D junction capacitance
231	vgs	Out	real	Gate-Source voltage
232	vds	Out	real	Drain-Source voltage
230	vbs	Out	real	Bulk-Source voltage
229	vbd	Out	real	Bulk-Drain voltage
8	nrd	InOut	real	Drain squares
7	nrs	InOut	real	Source squares
9	off	In	flag	Device initially off
12	icvds	InOut	real	Initial D-S voltage
13	icvgs	InOut	real	Initial G-S voltage
11	icvbs	InOut	real	Initial B-S voltage
20	temp	InOut	real	Instance temperature
21	dtemp	InOut	real	Instance temperature difference
10	ic	In	real vector	Vector of D-S, G-S, B-S voltages
15	sens_l	In	flag	flag to request sensitivity WRT length
14	sens_w	In	flag	flag to request sensitivity WRT width
203	dnode	Out	integer	Number of the drain node
204	gnode	Out	integer	Number of the gate node
205	snode	Out	integer	Number of the source node
206	bnode	Out	integer	Number of the node
207	dnodeprime	Out	integer	Number of int. drain node
208	snodeprime	Out	integer	Number of int. source node
258	rs	Out	real	Source resistance
209	sourceconductance	Out	real	Source conductance
259	rd	Out	real	Drain resistance
210	drainconductance	Out	real	Drain conductance
211	von	Out	real	Turn-on voltage
212	vdsat	Out	real	Saturation drain voltage
213	sourcevcrit	Out	real	Critical source voltage
214	drainvcrit	Out	real	Critical drain voltage
218	gmbs	Out	real	Bulk-Source transconductance
219	gm	Out	real	Transconductance
220	gds	Out	real	Drain-Source conductance
221	gbd	Out	real	Bulk-Drain conductance
222	gbs	Out	real	Bulk-Source conductance
233	cgs	Out	real	Gate-Source capacitance
236	cgd	Out	real	Gate-Drain capacitance
239	cgb	Out	real	Gate-Bulk capacitance

223	cbd	Out	real	Bulk-Drain capacitance
224	cbs	Out	real	Bulk-Source capacitance
225	cbd0	Out	real	Zero-Bias B-D junction capacitance
226	cbdsw0	Out	real	
227	cbs0	Out	real	Zero-Bias B-S junction capacitance
228	cbssw0	Out	real	
235	cqgs	Out	real	Capacitance due to gate-source charge storage
238	cqgd	Out	real	Capacitance due to gate-drain charge storage
241	cqgb	Out	real	Capacitance due to gate-bulk charge storage
243	cqbd	Out	real	Capacitance due to bulk-drain charge storage
245	cqbs	Out	real	Capacitance due to bulk-source charge storage
234	qgs	Out	real	Gate-Source charge storage
237	qgd	Out	real	Gate-Drain charge storage
240	qgb	Out	real	Gate-Bulk charge storage
242	qbd	Out	real	Bulk-Drain charge storage
244	qbs	Out	real	Bulk-Source charge storage
19	p	Out	real	Instaneous power
256	sens_l_dc	Out	real	dc sensitivity wrt length
246	sens_l_real	Out	real	real part of ac sensitivity wrt length
247	sens_l_imag	Out	real	imag part of ac sensitivity wrt length
248	sens_l_mag	Out	real	sensitivity wrt l of ac magnitude
249	sens_l_ph	Out	real	sensitivity wrt l of ac phase
250	sens_l_cplx	Out	complex	ac sensitivity wrt length
257	sens_w_dc	Out	real	dc sensitivity wrt width
251	sens_w_real	Out	real	real part of ac sensitivity wrt width
252	sens_w_imag	Out	real	imag part of ac sensitivity wrt width
253	sens_w_mag	Out	real	sensitivity wrt w of ac magnitude
254	sens_w_ph	Out	real	sensitivity wrt w of ac phase
255	sens_w_cplx	Out	complex	ac sensitivity wrt width

MOS6 model parameters

#	Name	Direction	Type	Description
140	type	Out	string	N-channel or P-channel MOS
101	vto	InOut	real	Threshold voltage
101	vt0	InOut	real	
102	kv	InOut	real	Saturation voltage factor
103	nv	InOut	real	Saturation voltage coeff.
104	kc	InOut	real	Saturation current factor
105	nc	InOut	real	Saturation current coeff.
106	nvth	InOut	real	Threshold voltage coeff.
107	ps	InOut	real	Sat. current modification par.
108	gamma	InOut	real	Bulk threshold parameter
109	gamma1	InOut	real	Bulk threshold parameter 1
110	sigma	InOut	real	Static feedback effect par.
111	phi	InOut	real	Surface potential
112	lambda	InOut	real	Channel length modulation param.
113	lambda0	InOut	real	Channel length modulation param. 0
114	lambda1	InOut	real	Channel length modulation param. 1
115	rd	InOut	real	Drain ohmic resistance
116	rs	InOut	real	Source ohmic resistance
117	cbd	InOut	real	B-D junction capacitance
118	cbs	InOut	real	B-S junction capacitance
119	is	InOut	real	Bulk junction sat. current
120	pb	InOut	real	Bulk junction potential
121	cgso	InOut	real	Gate-source overlap cap.
122	cgdo	InOut	real	Gate-drain overlap cap.
123	cgbo	InOut	real	Gate-bulk overlap cap.
131	rsh	InOut	real	Sheet resistance
124	cj	InOut	real	Bottom junction cap per area
125	mj	InOut	real	Bottom grading coefficient
126	cjsw	InOut	real	Side junction cap per area
127	mjsw	InOut	real	Side grading coefficient
128	js	InOut	real	Bulk jct. sat. current density
130	ld	InOut	real	Lateral diffusion
129	tox	InOut	real	Oxide thickness
132	u0	InOut	real	Surface mobility
132	uo	InOut	real	
133	fc	InOut	real	Forward bias jct. fit parm.
137	nmos	In	flag	N type MOSfet model
138	pmos	In	flag	P type MOSfet model
135	tpg	InOut	integer	Gate type
134	nsub	InOut	real	Substrate doping
136	nss	InOut	real	Surface state density
139	tnom	InOut	real	Parameter measurement temperature

30.5.5   MOS9 - Modified Level 3 MOSFET model

MOS9 instance parameters

#	Name	Direction	Type	Description
80	m	InOut	real	Multiplier
2	l	InOut	real	Length
1	w	InOut	real	Width
4	ad	InOut	real	Drain area
3	as	InOut	real	Source area
6	pd	InOut	real	Drain perimeter
5	ps	InOut	real	Source perimeter
34	id	Out	real	Drain current
34	cd	Out	real	Drain current
36	ibd	Out	real	B-D junction current
35	ibs	Out	real	B-S junction current
18	is	Out	real	Source current
17	ig	Out	real	Gate current
16	ib	Out	real	Bulk current
50	vgs	Out	real	Gate-Source voltage
51	vds	Out	real	Drain-Source voltage
49	vbs	Out	real	Bulk-Source voltage
48	vbd	Out	real	Bulk-Drain voltage
8	nrd	InOut	real	Drain squares
7	nrs	InOut	real	Source squares
9	off	In	flag	Device initially off
12	icvds	InOut	real	Initial D-S voltage
13	icvgs	InOut	real	Initial G-S voltage
11	icvbs	InOut	real	Initial B-S voltage
10	ic	InOut	real vector	Vector of D-S, G-S, B-S voltages
77	temp	InOut	real	Instance operating temperature
81	dtemp	InOut	real	Instance operating temperature difference
15	sens_l	In	flag	flag to request sensitivity WRT length
14	sens_w	In	flag	flag to request sensitivity WRT width
22	dnode	Out	integer	Number of drain node
23	gnode	Out	integer	Number of gate node
24	snode	Out	integer	Number of source node
25	bnode	Out	integer	Number of bulk node
26	dnodeprime	Out	integer	Number of internal drain node
27	snodeprime	Out	integer	Number of internal source node
30	von	Out	real	Turn-on voltage
31	vdsat	Out	real	Saturation drain voltage
32	sourcevcrit	Out	real	Critical source voltage
33	drainvcrit	Out	real	Critical drain voltage
78	rs	Out	real	Source resistance
28	sourceconductance	Out	real	Source conductance
79	rd	Out	real	Drain resistance
29	drainconductance	Out	real	Drain conductance
38	gm	Out	real	Transconductance
39	gds	Out	real	Drain-Source conductance
37	gmb	Out	real	Bulk-Source transconductance
37	gmbs	Out	real	Bulk-Source transconductance
40	gbd	Out	real	Bulk-Drain conductance
41	gbs	Out	real	Bulk-Source conductance
42	cbd	Out	real	Bulk-Drain capacitance
43	cbs	Out	real	Bulk-Source capacitance

52	cgs	Out	real	Gate-Source capacitance
55	cgd	Out	real	Gate-Drain capacitance
58	cgb	Out	real	Gate-Bulk capacitance
54	cqgs	Out	real	Capacitance due to gate-source charge storage
57	cqgd	Out	real	Capacitance due to gate-drain charge storage
60	cqgb	Out	real	Capacitance due to gate-bulk charge storage
62	cqbd	Out	real	Capacitance due to bulk-drain charge storage
64	cqbs	Out	real	Capacitance due to bulk-source charge storage
44	cbd0	Out	real	Zero-Bias B-D junction capacitance
45	cbdsw0	Out	real	Zero-Bias B-D sidewall capacitance
46	cbs0	Out	real	Zero-Bias B-S junction capacitance
47	cbssw0	Out	real	Zero-Bias B-S sidewall capacitance
63	qbs	Out	real	Bulk-Source charge storage
53	qgs	Out	real	Gate-Source charge storage
56	qgd	Out	real	Gate-Drain charge storage
59	qgb	Out	real	Gate-Bulk charge storage
61	qbd	Out	real	Bulk-Drain charge storage
19	p	Out	real	Instantaneous power
76	sens_l_dc	Out	real	dc sensitivity wrt length
70	sens_l_real	Out	real	real part of ac sensitivity wrt length
71	sens_l_imag	Out	real	imag part of ac sensitivity wrt length
74	sens_l_cplx	Out	complex	ac sensitivity wrt length
72	sens_l_mag	Out	real	sensitivity wrt l of ac magnitude
73	sens_l_ph	Out	real	sensitivity wrt l of ac phase
75	sens_w_dc	Out	real	dc sensitivity wrt width
65	sens_w_real	Out	real	real part of ac sensitivity wrt width
66	sens_w_imag	Out	real	imag part of ac sensitivity wrt width
67	sens_w_mag	Out	real	sensitivity wrt w of ac magnitude
68	sens_w_ph	Out	real	sensitivity wrt w of ac phase
69	sens_w_cplx	Out	complex	ac sensitivity wrt width

MOS9 model parameters

#	Name	Direction	Type	Description
144	type	Out	string	N-channel or P-channel MOS
133	nmos	In	flag	N type MOSfet model
134	pmos	In	flag	P type MOSfet model
101	vto	InOut	real	Threshold voltage
101	vt0	InOut	real	
102	kp	InOut	real	Transconductance parameter
103	gamma	InOut	real	Bulk threshold parameter
104	phi	InOut	real	Surface potential
105	rd	InOut	real	Drain ohmic resistance
106	rs	InOut	real	Source ohmic resistance
107	cbd	InOut	real	B-D junction capacitance
108	cbs	InOut	real	B-S junction capacitance
109	is	InOut	real	Bulk junction sat. current
110	pb	InOut	real	Bulk junction potential
111	cgso	InOut	real	Gate-source overlap cap.
112	cgdo	InOut	real	Gate-drain overlap cap.
113	cgbo	InOut	real	Gate-bulk overlap cap.
114	rsh	InOut	real	Sheet resistance
115	cj	InOut	real	Bottom junction cap per area
116	mj	InOut	real	Bottom grading coefficient
117	cjsw	InOut	real	Side junction cap per area
118	mjsw	InOut	real	Side grading coefficient
119	js	InOut	real	Bulk jct. sat. current density
120	tox	InOut	real	Oxide thickness
121	ld	InOut	real	Lateral diffusion
145	xl	InOut	real	Length mask adjustment
146	wd	InOut	real	Width Narrowing (Diffusion)
147	xw	InOut	real	Width mask adjustment
148	delvto	InOut	real	Threshold voltage Adjust
148	delvt0	InOut	real	
122	u0	InOut	real	Surface mobility
122	uo	InOut	real	
123	fc	InOut	real	Forward bias jct. fit parm.
124	nsub	InOut	real	Substrate doping
125	tpg	InOut	integer	Gate type
126	nss	InOut	real	Surface state density
131	vmax	InOut	real	Maximum carrier drift velocity
135	xj	InOut	real	Junction depth
129	nfs	InOut	real	Fast surface state density
138	xd	InOut	real	Depletion layer width
139	alpha	InOut	real	Alpha
127	eta	InOut	real	Vds dependence of threshold voltage
128	delta	InOut	real	Width effect on threshold
140	input_delta	InOut	real	
130	theta	InOut	real	Vgs dependence on mobility
132	kappa	InOut	real	Kappa
141	tnom	InOut	real	Parameter measurement temperature
142	kf	InOut	real	Flicker noise coefficient
143	af	InOut	real	Flicker noise exponent

30.5.6   BSIM1 - Berkeley Short Channel IGFET Model

BSIM1 instance parameters

#	Name	Direction	Type	Description
2	l	InOut	real	Length
1	w	InOut	real	Width
14	m	InOut	real	Parallel Multiplier
4	ad	InOut	real	Drain area
3	as	InOut	real	Source area
6	pd	InOut	real	Drain perimeter
5	ps	InOut	real	Source perimeter
8	nrd	InOut	real	Number of squares in drain
7	nrs	InOut	real	Number of squares in source
9	off	InOut	flag	Device is initially off
11	vds	InOut	real	Initial D-S voltage
12	vgs	InOut	real	Initial G-S voltage
10	vbs	InOut	real	Initial B-S voltage
13	ic	In	unknown vector	Vector of DS,GS,BS initial voltages

BSIM1 Model Parameters

#	Name	Direction	Type	Description
101	vfb	InOut	real	Flat band voltage
102	lvfb	InOut	real	Length dependence of vfb
103	wvfb	InOut	real	Width dependence of vfb
104	phi	InOut	real	Strong inversion surface potential
105	lphi	InOut	real	Length dependence of phi
106	wphi	InOut	real	Width dependence of phi
107	k1	InOut	real	Bulk effect coefficient 1
108	lk1	InOut	real	Length dependence of k1
109	wk1	InOut	real	Width dependence of k1
110	k2	InOut	real	Bulk effect coefficient 2
111	lk2	InOut	real	Length dependence of k2
112	wk2	InOut	real	Width dependence of k2
113	eta	InOut	real	VDS dependence of threshold voltage
114	leta	InOut	real	Length dependence of eta
115	weta	InOut	real	Width dependence of eta
116	x2e	InOut	real	VBS dependence of eta
117	lx2e	InOut	real	Length dependence of x2e
118	wx2e	InOut	real	Width dependence of x2e
119	x3e	InOut	real	VDS dependence of eta
120	lx3e	InOut	real	Length dependence of x3e
121	wx3e	InOut	real	Width dependence of x3e
122	dl	InOut	real	Channel length reduction in um
123	dw	InOut	real	Channel width reduction in um
124	muz	InOut	real	Zero field mobility at VDS=0 VGS=VTH
125	x2mz	InOut	real	VBS dependence of muz
126	lx2mz	InOut	real	Length dependence of x2mz
127	wx2mz	InOut	real	Width dependence of x2mz
128	mus	InOut	real	Mobility at VDS=VDD VGS=VTH, channel length modulation
129	lmus	InOut	real	Length dependence of mus
130	wmus	InOut	real	Width dependence of mus
131	x2ms	InOut	real	VBS dependence of mus
132	lx2ms	InOut	real	Length dependence of x2ms
133	wx2ms	InOut	real	Width dependence of x2ms

134	x3ms	InOut	real	VDS dependence of mus
135	lx3ms	InOut	real	Length dependence of x3ms
136	wx3ms	InOut	real	Width dependence of x3ms
137	u0	InOut	real	VGS dependence of mobility
138	lu0	InOut	real	Length dependence of u0
139	wu0	InOut	real	Width dependence of u0
140	x2u0	InOut	real	VBS dependence of u0
141	lx2u0	InOut	real	Length dependence of x2u0
142	wx2u0	InOut	real	Width dependence of x2u0
143	u1	InOut	real	VDS depence of mobility, velocity saturation
144	lu1	InOut	real	Length dependence of u1
145	wu1	InOut	real	Width dependence of u1
146	x2u1	InOut	real	VBS depence of u1
147	lx2u1	InOut	real	Length depence of x2u1
148	wx2u1	InOut	real	Width depence of x2u1
149	x3u1	InOut	real	VDS depence of u1
150	lx3u1	InOut	real	Length dependence of x3u1
151	wx3u1	InOut	real	Width depence of x3u1
152	n0	InOut	real	Subthreshold slope
153	ln0	InOut	real	Length dependence of n0
154	wn0	InOut	real	Width dependence of n0
155	nb	InOut	real	VBS dependence of subthreshold slope
156	lnb	InOut	real	Length dependence of nb
157	wnb	InOut	real	Width dependence of nb
158	nd	InOut	real	VDS dependence of subthreshold slope
159	lnd	InOut	real	Length dependence of nd
160	wnd	InOut	real	Width dependence of nd
161	tox	InOut	real	Gate oxide thickness in um
162	temp	InOut	real	Temperature in degree Celcius
163	vdd	InOut	real	Supply voltage to specify mus
164	cgso	InOut	real	Gate source overlap capacitance per unit channel width(m)
165	cgdo	InOut	real	Gate drain overlap capacitance per unit channel width(m)
166	cgbo	InOut	real	Gate bulk overlap capacitance per unit channel length(m)
167	xpart	InOut	real	Flag for channel charge partitioning
168	rsh	InOut	real	Source drain diffusion sheet resistance in ohm per square
169	js	InOut	real	Source drain junction saturation current per unit area
170	pb	InOut	real	Source drain junction built in potential
171	mj	InOut	real	Source drain bottom junction capacitance grading coefficient
172	pbsw	InOut	real	Source drain side junction capacitance built in potential
173	mjsw	InOut	real	Source drain side junction capacitance grading coefficient
174	cj	InOut	real	Source drain bottom junction capacitance per unit area
175	cjsw	InOut	real	Source drain side junction capacitance per unit area
176	wdf	InOut	real	Default width of source drain diffusion in um
177	dell	InOut	real	Length reduction of source drain diffusion
180	kf	InOut	real	Flicker noise coefficient
181	af	InOut	real	Flicker noise exponent
178	nmos	In	flag	Flag to indicate NMOS
179	pmos	In	flag	Flag to indicate PMOS

30.5.7   BSIM2 - Berkeley Short Channel IGFET Model

BSIM2 instance parameters

#	Name	Direction	Type	Description
2	l	InOut	real	Length
1	w	InOut	real	Width
14	m	InOut	real	Parallel Multiplier
4	ad	InOut	real	Drain area
3	as	InOut	real	Source area
6	pd	InOut	real	Drain perimeter
5	ps	InOut	real	Source perimeter
8	nrd	InOut	real	Number of squares in drain
7	nrs	InOut	real	Number of squares in source
9	off	InOut	flag	Device is initially off
11	vds	InOut	real	Initial D-S voltage
12	vgs	InOut	real	Initial G-S voltage
10	vbs	InOut	real	Initial B-S voltage
13	ic	In	unknown vector	Vector of DS,GS,BS initial voltages

BSIM2 model parameters

#	Name	Direction	Type	Description
101	vfb	InOut	real	Flat band voltage
102	lvfb	InOut	real	Length dependence of vfb
103	wvfb	InOut	real	Width dependence of vfb
104	phi	InOut	real	Strong inversion surface potential
105	lphi	InOut	real	Length dependence of phi
106	wphi	InOut	real	Width dependence of phi
107	k1	InOut	real	Bulk effect coefficient 1
108	lk1	InOut	real	Length dependence of k1
109	wk1	InOut	real	Width dependence of k1
110	k2	InOut	real	Bulk effect coefficient 2
111	lk2	InOut	real	Length dependence of k2
112	wk2	InOut	real	Width dependence of k2
113	eta0	InOut	real	VDS dependence of threshold voltage at VDD=0
114	leta0	InOut	real	Length dependence of eta0
115	weta0	InOut	real	Width dependence of eta0
116	etab	InOut	real	VBS dependence of eta
117	letab	InOut	real	Length dependence of etab
118	wetab	InOut	real	Width dependence of etab
119	dl	InOut	real	Channel length reduction in um
120	dw	InOut	real	Channel width reduction in um
121	mu0	InOut	real	Low-field mobility, at VDS=0 VGS=VTH
122	mu0b	InOut	real	VBS dependence of low-field mobility
123	lmu0b	InOut	real	Length dependence of mu0b
124	wmu0b	InOut	real	Width dependence of mu0b
125	mus0	InOut	real	Mobility at VDS=VDD VGS=VTH
126	lmus0	InOut	real	Length dependence of mus0
127	wmus0	InOut	real	Width dependence of mus
128	musb	InOut	real	VBS dependence of mus
129	lmusb	InOut	real	Length dependence of musb
130	wmusb	InOut	real	Width dependence of musb
131	mu20	InOut	real	VDS dependence of mu in tanh term
132	lmu20	InOut	real	Length dependence of mu20
133	wmu20	InOut	real	Width dependence of mu20
134	mu2b	InOut	real	VBS dependence of mu2

135	lmu2b	InOut	real	Length dependence of mu2b
136	wmu2b	InOut	real	Width dependence of mu2b
137	mu2g	InOut	real	VGS dependence of mu2
138	lmu2g	InOut	real	Length dependence of mu2g
139	wmu2g	InOut	real	Width dependence of mu2g
140	mu30	InOut	real	VDS dependence of mu in linear term
141	lmu30	InOut	real	Length dependence of mu30
142	wmu30	InOut	real	Width dependence of mu30
143	mu3b	InOut	real	VBS dependence of mu3
144	lmu3b	InOut	real	Length dependence of mu3b
145	wmu3b	InOut	real	Width dependence of mu3b
146	mu3g	InOut	real	VGS dependence of mu3
147	lmu3g	InOut	real	Length dependence of mu3g
148	wmu3g	InOut	real	Width dependence of mu3g
149	mu40	InOut	real	VDS dependence of mu in linear term
150	lmu40	InOut	real	Length dependence of mu40
151	wmu40	InOut	real	Width dependence of mu40
152	mu4b	InOut	real	VBS dependence of mu4
153	lmu4b	InOut	real	Length dependence of mu4b
154	wmu4b	InOut	real	Width dependence of mu4b
155	mu4g	InOut	real	VGS dependence of mu4
156	lmu4g	InOut	real	Length dependence of mu4g
157	wmu4g	InOut	real	Width dependence of mu4g
158	ua0	InOut	real	Linear VGS dependence of mobility
159	lua0	InOut	real	Length dependence of ua0
160	wua0	InOut	real	Width dependence of ua0
161	uab	InOut	real	VBS dependence of ua
162	luab	InOut	real	Length dependence of uab
163	wuab	InOut	real	Width dependence of uab
164	ub0	InOut	real	Quadratic VGS dependence of mobility
165	lub0	InOut	real	Length dependence of ub0
166	wub0	InOut	real	Width dependence of ub0
167	ubb	InOut	real	VBS dependence of ub
168	lubb	InOut	real	Length dependence of ubb
169	wubb	InOut	real	Width dependence of ubb
170	u10	InOut	real	VDS depence of mobility
171	lu10	InOut	real	Length dependence of u10
172	wu10	InOut	real	Width dependence of u10
173	u1b	InOut	real	VBS depence of u1
174	lu1b	InOut	real	Length depence of u1b
175	wu1b	InOut	real	Width depence of u1b
176	u1d	InOut	real	VDS depence of u1
177	lu1d	InOut	real	Length depence of u1d
178	wu1d	InOut	real	Width depence of u1d
179	n0	InOut	real	Subthreshold slope at VDS=0 VBS=0
180	ln0	InOut	real	Length dependence of n0
181	wn0	InOut	real	Width dependence of n0
182	nb	InOut	real	VBS dependence of n
183	lnb	InOut	real	Length dependence of nb
184	wnb	InOut	real	Width dependence of nb
185	nd	InOut	real	VDS dependence of n
186	lnd	InOut	real	Length dependence of nd
187	wnd	InOut	real	Width depence of nd
188	vof0	InOut	real	Threshold voltage offset AT VDS=0 VBS=0
189	lvof0	InOut	real	Length dependence of vof0

190	wvof0	InOut	real	Width dependence of vof0
191	vofb	InOut	real	VBS dependence of vof
192	lvofb	InOut	real	Length dependence of vofb
193	wvofb	InOut	real	Width dependence of vofb
194	vofd	InOut	real	VDS dependence of vof
195	lvofd	InOut	real	Length dependence of vofd
196	wvofd	InOut	real	Width dependence of vofd
197	ai0	InOut	real	Pre-factor of hot-electron effect.
198	lai0	InOut	real	Length dependence of ai0
199	wai0	InOut	real	Width dependence of ai0
200	aib	InOut	real	VBS dependence of ai
201	laib	InOut	real	Length dependence of aib
202	waib	InOut	real	Width dependence of aib
203	bi0	InOut	real	Exponential factor of hot-electron effect.
204	lbi0	InOut	real	Length dependence of bi0
205	wbi0	InOut	real	Width dependence of bi0
206	bib	InOut	real	VBS dependence of bi
207	lbib	InOut	real	Length dependence of bib
208	wbib	InOut	real	Width dependence of bib
209	vghigh	InOut	real	Upper bound of the cubic spline function.
210	lvghigh	InOut	real	Length dependence of vghigh
211	wvghigh	InOut	real	Width dependence of vghigh
212	vglow	InOut	real	Lower bound of the cubic spline function.
213	lvglow	InOut	real	Length dependence of vglow
214	wvglow	InOut	real	Width dependence of vglow
215	tox	InOut	real	Gate oxide thickness in um
216	temp	InOut	real	Temperature in degree Celcius
217	vdd	InOut	real	Maximum Vds
218	vgg	InOut	real	Maximum Vgs
219	vbb	InOut	real	Maximum Vbs
220	cgso	InOut	real	Gate source overlap capacitance per unit channel width(m)
221	cgdo	InOut	real	Gate drain overlap capacitance per unit channel width(m)
222	cgbo	InOut	real	Gate bulk overlap capacitance per unit channel length(m)
223	xpart	InOut	real	Flag for channel charge partitioning
224	rsh	InOut	real	Source drain diffusion sheet resistance in ohm per square
225	js	InOut	real	Source drain junction saturation current per unit area
226	pb	InOut	real	Source drain junction built in potential
227	mj	InOut	real	Source drain bottom junction capacitance grading coefficient
228	pbsw	InOut	real	Source drain side junction capacitance built in potential
229	mjsw	InOut	real	Source drain side junction capacitance grading coefficient
230	cj	InOut	real	Source drain bottom junction capacitance per unit area
231	cjsw	InOut	real	Source drain side junction capacitance per unit area
232	wdf	InOut	real	Default width of source drain diffusion in um
233	dell	InOut	real	Length reduction of source drain diffusion
236	kf	InOut	real	Flicker noise coefficient
237	af	InOut	real	Flicker noise exponent
234	nmos	In	flag	Flag to indicate NMOS
235	pmos	In	flag	Flag to indicate PMOS

30.5.8   BSIM3

Detailed ewscriptions are still missing here. Please refer to the excellent manual issued by University of Berkeley.

30.5.9   BSIM4

Detailed ewscriptions are still missing here. Please refer to the excellent manual issued by University of Berkeley.

Chapter 31

Compilation notes

This file describes the procedures to install ngspice from sources.

31.1 Ngspice Installation under LINUX (and other 'UNIXes')

31.1.1 Prerequisites

Ngspice is written in C and thus a complete C compilation environment is needed. Almost any UNIX comes with a complete C development environment. Ngspice is developed on GNU/Linux with gcc and GNU make.

The following software must be installed in your system to compile ngspice: **bison**, **flex** and **X11**.

If you want to compile the CVS source you need additional software: **autoconf, automake, libtool, texinfo**.

The following software may be needed when enabling additional features: **editline, tcl/tk**

31.1.2 Install from CVS

This section describes how to install from source code taken direct from CVS. It is intended more for developers than for users as the code in CVS may be unstable. For user install instructions using source from released distributions, please see the sections titled 'Basic Install' and 'Advanced Install'.

Download source from CVS as described on the sourceforge project page (see http://sourceforge.net/projects/ngspice/ and click on the CVS link)

Now change directories in to the top-level source directory (where this INSTALL file can be found).

The project uses the GNU build process. You should be able to do the following:

```
$ ./autogen.sh
$ ./configure --enable-xspice --enable-cider --disable-debug --with-editline=yes
$ make
$ sudo make install
```

At present it is normal for there to be some warning generated during this process.

See the section titled 'Advanced Install' for instructions about arguments that can be passed to `./configure` to customise the build and installation. The following arguments are already used here and may be called sort of "standard":

`--enable-xspice` Include the XSPICE extensions (see chapters 13 and 27)

`--enable-cider` Include CIDER numerical device simuzlator (see chapter 29)

`--disable-debug` No debugging information included (optimized annd compact code)

`--with-editline=yes` Include an editor for the input command line (command history, backspace, insert etc.). If editline is not available, readline may be used.

If a problem is found with the build proccess, please submit a report to the Ngspice development team. Please provide information about your system and any `./configure` arguments you are using, together with any error messages. Ideally you would have tried to fix the problem yourself first. If you have fixed the problem then the development team will love to hear from you.

31.1.3   Basic Install

This covers installation from a tarball (for example ngspice-rework-22.tgz, to be found at http://sourceforge.net/projects/n
After downloading the tar ball to a local directory unpack it using:

    $ tar -zxvf ngspice-rework-22.tgz

Now change directories in to the top-level source directory (where this text from the INSTALL file
can be found).

You should be able to do:

    $ ./autogen.sh
    $ ./configure
    $ make
    $ sudo make install

The default install dir is /usr/local/bin

See the section titled 'Advanced Install' for instructions about arguments that can be passed to
`./configure` to customise the build and installation.

31.1.4   Advanced Install

Some extra options can be provided to './configure'. To get all available options do:

    $ ./configure --help

Some of these options are generic to the GNU build process that is used by Ngspice, other are specific
to Ngspice.

The following sections provide some guidance and descriptions for many, but not all, of these options.

1.4.1 Options Specific to Using Ngspice

`--enable-adms` ADMS is an experimental model compiler that translates Verilog-A compact models into
C code that can be compiled into ngspice. This is still experimental, but working with some limitations
to the models (e.g. no noise models). If you want to use it, please refer to the ADMS section on ngspice
web site .

`--enable-cider` Cider is a mixed-level simulator that couples Spice3 and DSIM to simulate devices
from their technological parameters. This part of the simulator is not compiled by default.

`--enable-ndev` Enable NDEV interface, (experimental) A TCP/IP interface to external device simu-
lator such as GSS. For more information, please visit the homepage of GSS at http://gss-tcad.sourceforge.net

`--enable-newpred` Enable the NEWPRED symbol in the code.

`--enable-xspice` Enable XSpice enhancements, (experimental) A mixed signal simulator built upon
spice3 with codemodel dynamic loading support. See chapter 13 and section II for details.

`--with-editline=yes` Enables the use of the BSD editline library (libedit). See http://www.thrysoee.dk/editline/.
This is preferred over GNU readline due to licensing issues.

`--with-readline=yes` Enable GNU readline support for the command line interface.

`--with-tcl=tcldir` When configured with this option the tcl module "tclspice" is compiled and
installed instead of plain ngspice.

`--enable-openmp` Compile ngspice for multi-core processors. Parallelization is done by OpenMP (see
chapt. 17.10).

The following options are seldom used today, not tested, some may even no longer be impleneted.

`--enable-capbypass` Bypass calculation of cbd/cbs in the mosfets if the vbs/vbd voltages are un-
changed.

`--enable-capzerobypass` Bypass all the cbd/cbs calculations if Czero is zero. This is enabled by
default since rework-18.

`--enable-cluster` Clustering code for distributed simulation. This is a contribution never tested.
This code comes from TCLspice implementation and is implemented for transient analysis only.

`--enable-expdevices` Enable experimental devices. This option is used by developers to mask devices
under development. Almost useless for users.

`--enable-experimental` This enables some experimental code. Specifically it enables: * support for
altering options in interactive mode by adding the interactive keyword 'options'. * The ability to save
and load snapshots: adds interactive keywords 'savesnap' and 'loadsnap'.

`--enable-help` Force building nghelp. This is deprecated.

`--enable-newtrunc` Enable the newtrunc option

`--enable-nodelimiting` Experimental damping scheme

`--enable-nobypass` Don't bypass recalculations of slowly changing variables

`--enable-nosqrt` Use always log/exp for non-linear capacitances –enable-predictor Enable a predictor method for convergence

`--enable-sense2` Use spice2 sensitivity analysis

`--enable-xgraph` Compile the Xgraph plotting program. Xgraph is a plotting package for X11 and was once very popular.

1.4.2 Options Useful for Debugging Ngspice

`--disable-debug` This option will remove the '-g' option passed to the compiler. This speeds up execution time (and compilation) a *lot*, and is recommended for normal use.

The following options are seldom used today, not tested, some may even no longer be impleneted.

`--enable-ansi` Configure will try to find an option for your compiler so that it expects ansi-C.

`--enable-asdebug` Debug sensitivity code *ASDEBUG*.

`--enable-blktmsdebug` Debug distortion code *BLOCKTIMES*

`--enable-checkergcc` Option for compilation with checkergcc.

`--enable-cpdebug` Enable ngspice shell code debug.

`--enable-ftedebug` Enable ngspice frontend debug.

`--enable-gc` Enable the Boehm-Weiser Conservative Garbage Collector.

`--enable-pzdebug` Debug pole/zero code.

`--enable-sensdebug` Debug sensitivity code *SENSDEBUG*.

`--enable-smltmsdebug` Debug distortion code *SMALLTIMES*

`--enable-smoketest` Enable smoketest compile.

`--enable-stepdebug` Turns on debugging of convergence steps in transient analysis

31.1.5 Compilation using an user defined directory tree for object files

The procedures described above will store the *.o files (output of the compilation step) into the directories where the sources (*.c) are located. This may not be the best option if you want for example to maintain a debug version and in parallel a release version of ngspice (`./configure --disable-debug`). So if you intend to create a separate object file tree like ng-spice-rework/ngbuild/release, you may do the following, starting from the default directory ng-spice-rework:

```
mkdir -p ngbuild/release
cd ngbuild/release
../../configure <some options>
make install
```

This will create an object file directory tree, similar to the source file directory tree, the object files are now separated from the source files. For the debug version, you may do the same as described above, replacing 'release' by 'debug', and obtain another separated object file directory tree. If you already have run `./configure` in ng-spice-rework, you have to do a maintainer-clean, before the above procedure will work. Unfortunately this procedure will not work with Xspice enabled, so your options should not include `--enable-xspice`. This bug will be removed in a future version of ngspice.

31.1.6 Compilers and Options

Some systems require unusual options for compilation or linking that the 'configure' script does not know about. You can give 'configure' initial values for variables by setting them in the environment. Using a Bourne-compatible shell, you can do that on the command line like this:

```
CC=c89
CFLAGS=-O2
LIBS=-lposix
./configure
```

Or on systems that have the 'env' program, you can do it like this:

```
env CPPFLAGS=-I/usr/local/include
LDFLAGS=-s
./configure
```

31.1.7   Compiling For Multiple Architectures

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you must use a version of 'make' that supports the 'VPATH' variable, such as GNU 'make'. 'cd' to the directory where you want the object files and executables to go and run the 'configure' script. 'configure' automatically checks for the source code in the directory that 'configure' is in and in '..'.

If you have to use a 'make' that does not supports the 'VPATH' variable, you have to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use 'make distclean' before reconfiguring for another architecture.

31.1.8   Installation Names

By default, 'make install' will install the package's files in '/usr/local/bin', '/usr/local/man', etc. You can specify an installation prefix other than '/usr/local' by giving 'configure' the option '–prefix=PATH'.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you give 'configure' the option '–exec-prefix=PATH', the package will use PATH as the prefix for installing programs and libraries. Documentation and other data files will still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like '–bindir=PATH' to specify different values for particular kinds of files. Run 'configure –help' for a list of the directories you can set and what kinds of files go in them.

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving 'configure' the option '–program-prefix=PREFIX' or '–program-suffix=SUFFIX'.

When installed on MinGW with MSYS alternative paths are not fully supported. See 'How to make ngspice with MINGW and MSYS' below for details.

31.1.9   Optional Features

Some packages pay attention to '–enable-FEATURE' options to 'configure', where FEATURE indicates an optional part of the package. They may also pay attention to '–with-PACKAGE' options, where PACKAGE is something like 'gnu-as' or 'x' (for the X Window System). The 'README' should mention any '–enable-' and '–with-' options that the package recognizes.

For packages that use the X Window System, 'configure' can usually find the X include and library files automatically, but if it doesn't, you can use the 'configure' options '–x-includes=DIR' and '–x-libraries=DIR' to specify their locations.

31.1.10   Specifying the System Type

There may be some features 'configure' can not figure out automatically, but needs to determine by the type of host the package will run on. Usually 'configure' can figure that out, but if it prints a message saying it can not guess the host type, give it the '–host=TYPE' option. TYPE can either be a short name for the system type, such as 'sun4', or a canonical name with three fields: CPU-COMPANY-SYSTEM

See the file 'config.sub' for the possible values of each field. If 'config.sub' isn't included in this package, then this package doesn't need to know the host type.

If you are building compiler tools for cross-compiling, you can also use the '–target=TYPE' option to select the type of system they will produce code for and the '–build=TYPE' option to select the type of system on which you are compiling the package.

31.1.11   Sharing Defaults

If you want to set default values for 'configure' scripts to share, you can create a site shell script called 'config.site' that gives default values for variables like 'CC', 'cache_file', and 'prefix'. 'configure' looks for 'PREFIX/share/config.site' if it exists, then 'PREFIX/etc/config.site' if it exists. Or, you can set the 'CONFIG_SITE' environment variable to the location of the site script. A warning: not all 'configure' scripts look for a site script.

31.1.12   Operation Controls

'configure' recognizes the following options to control how it operates.

‘`--cache-file=FILE`’ Use and save the results of the tests in FILE instead of ‘./config.cache’. Set FILE to ‘/dev/null’ to disable caching, for debugging ‘configure’.

‘`--help`’ Print a summary of the options to ‘configure’, and exit.

‘`--quiet`’ ‘`--silent`’ ‘`-q`’ Do not print messages saying which checks are being made. To suppress all normal output, redirect it to ‘/dev/null’ (any error messages will still be shown).

‘`--srcdir=DIR`’ Look for the package’s source code in directory DIR. Usually ‘configure’ can determine that directory automatically.

‘`--version`’ Print the version of Autoconf used to generate the ‘configure’ script, and exit.

‘`configure`’ also accepts some other, not widely useful, options.

31.2 NGSPICE COMPILATION UNDER WINDOWS OS

31.2.1 How to make ngspice with MINGW and MSYS

Creating ngspice with MINGW is now a straight forward procedure, if you have MSYS/MINGW installed properly. Unfortunately this is rather tedious because you will need several enhancements to the standard install, especially if you want to include XSpice. Some links are given below which describe the procedures. The default installation location of ngspice is the Windows path C:\spice. The install path can be altered by passing `--prefix=NEWPATH` as an argument to `./configure` during the build process.

Put the install path you desire inside "", e.g. `"D:/NewSpice"`. Be careful to use forward slashes "/", not backward slashes "\" (something still to be fixed). Then add `--prefix="D:/NewSpice"` as an argument to `./configure` in the normal way.

The procedure of compiling a distribution (for example, a tarball from the ngspice website), is as follows:

```
$ cd ng-spice-rework
$ ./configure --with-windows ...and other options
$ make
$ make install
```

The useful options are:

`--enable-xspice` (this requires FLEX and BISON available in MSYS, see below).

`--enable-cider`

`--disable-debug` (-O2 optimization, no debug information)

However, to compile code extracted from the CVS repository the procedure is a little different, thus:

```
$ cd ng-spice-rework
$ ./autogen.sh
$ ./configure --enable-maintainer-mode --with-windows ...and other options
$ make
$ make install
```

MINGW and MSYS can be downloaded from http://www.mingw.org/. The making of the code models *.cm for XSpice requires installation of BISON and FLEX to MSYS. A typical installation was tested with: bison-2.0-MSYS.tar.gz flex-2.5.4a-1-bin.zip libiconv-1.9.2-1-bin.zip libintl-0.14.4-bin.zip

Bison 2.0 is now superseeded by newer releases (Bison 2.3, see http://sourceforge.net/project/showfiles.php?group_id=

The last three are from http://sourceforge.net/project/showfiles.php?group_id=23617.

You may also look at

http://www.mingw.org/wiki/HOWTO_Install_the_MinGW_GCC_Compiler_Suite

http://www.mingw.org/wiki/MSYS

http://www.mingw.org/wiki/HOWTO_Create_an_MSYS_Build_Environment.

31.2.2 64 Bit executables with MINGW-w64

Procedure:

Install MSYS, plus bison, flex, auto tools, perl, libiconv, libintl

Install MINGW-w64, activate OpenMP support

See either http://mingw-w64.sourceforge.net/ or http://tdm-gcc.tdragon.net/

(allows to generate both 32 or 64 bit executables by setting flag -m32 or -m64)

Set path to compiler in msys/xx/etc/fstab (e.g. c:/MinGW64 /mingw)

Start compiling with

'./compile_min.sh' or './compile_min.sh 64'

Options used in the script:

–adms and –enable-adms ADMS is an experimental model compiler that translates Verilog-A compact models into C code that can be compiled into ngspice. This is still experimental, but working with some limitations to the models (e.g. no noise models). If you want to use it, please refer to the ADMS section on ngspice web site .

CIDER, XSPICE, and OpenMP may be selected at will.

–disable-debug will give O2 optimization (versus O0 for debug) and removes all debugging info.

The install script will copy all files to C:\Spice or C:\Spice64, the code models for XSPICE will be stored in C:\Spice\lib\spice or C:\Spice64\lib\spice respectively.

A word of caution: Be aware that there might be some bugs in your 64 bit code. We still have some compiler warnings about integer incompatibility (e.g. interger versus size_t etc.)! We will take care of that for the next release.

31.2.3   make ngspice with MS Visual Studio 2008

ngspice may be compiled with MS Visual Studio 2008. Support for MS Visual Studio 2010 is not yet available.

CIDER and XSPICE are included, but the code models for XSPICE (*.cm) are not (yet) made. You may however use the code models (which in fact are dlls) created with MINGW, as e.g. found in the ngspice binary distribution. There is currently no installation procedure provided, you may however install the executable manually as described in the installation tree below. The directory (visualc) with its files vngspice.sln (project starter) and vngspice.vcproj (project contents) allows to compile and link ngspice with MS Visual Studio 2008. The project is probably not compatible with Visual Studio 2005 and not yet with 2010.

/visualc/include contains a dedicated config.h file. It contains the preprocessor definitions required to properly compile the code. strings.h has been necessary during setting up the project.

Install Microsoft Visual Studio 2008 C++ . The MS VS 2008 C++ Express Edition (which is available at no cost from http://www.microsoft.com/express/product/default.aspx) is adequate, if you do not wish to have OpenMP or 64 bit support. So the express edition will allow a 32 bit Release and a Debug version of ngspice, using the Win32 flag. In addition you may select a console version without graphics interface. The professional edition will ofer Release and Debug and Console also for 64 bit (flag x64), as well as an OpenMP variant for 32 or 64 bit.

Procedure:

Goto /ng-spice-rework/visualc.

Start MS Visual Studio 2008 by double click onto vngspice.sln. After MS Visual Studio has opened up, select debug or release version by checking 'Erstellen' , 'Konfigurations-Manager' 'Debug' or 'Release'. Start making ngspice (called vngspice.exe) by selecting 'Erstellen' and 'vngspice neu erstellen'. Object files will be created and stored in visualc/debug or visualc/release. The executable will be stored to visualc/debug/bin or visualc/release/bin.

An installation tree (as provided with MINGW make install) and also used by vngspice in its current distribution is shown in the following table (maybe created manually):

If you intend to install vngspice into another directory, e.g. D:\MySpice, you have to edit /visualc/include/config.h and alter the following entries from:

```
#define NGSPICEBINDIR "C:/Spice/bin"
#define NGSPICEDATADIR "C:/Spice/share/ng-spice-rework"
to
#define NGSPICEBINDIR "D:/MySpice/bin"
#define NGSPICEDATADIR "D:/MySpice/share/ng-spice-rework"
```

nghelp.exe is deprecated and no longer offered, but still available in the binary distribution. If the code model files *.cm are not available, you will get warning messages, but you may use ngspice in the normal way (of course without XSPICE extensions). **ToDo**: Some commands in how-to-ngspice-vstudio.txt and mentioned above have to be translated to English.

31.2.4   make ngspice with pure CYGWIN

If you don't have libdl.a you may need to link libcygwin.a to libdl.a symbolically.

for example:

```
$ cd /lib $ ln -s libcygwin.a libdl.a.
```

The procedure of compiling is the same as Linux.

```
C:\Spice\
    bin\
        ngspice.exe
        nghelp.exe
        ngmakeidx.exe
        ngnutmeg.exe
        cmpp.exe
    lib\
        spice\
            analog.cm
            digital.cm
            spice2poly.cm
            extradev.cm
            extravt.cm
    share\
        info\
            dir
            ngspice.info
            ngspice.info-1
            ..
            ngspice.info-10
        man\
        man1\
            ngmultidec.1
            ngnutmeg.1
            ngsconvert.1
            ngspice.1
    ng-spice-rework\
        helpdir\
            ngspice.idx
            ngspice.txt
        scripts\
            ciderinit
            devaxis
            devload
            setplot
            spectrum
            spinit
```

Table 31.1: ngspice standard installation tree under MS Windows

31.2.5   make ngspice with CYGWIN and external MINGW32

The next two compilation options are deprecated and not tested.

according to http://www.geocrawler.com/lists/3/SourceForge/6013/0/7321042/
```
$ cd ng-spice-rework
$ export PATH="/cygdrive/g/gcc_mingw/bin:$PATH"
$ autoconf
$ rm config.cache
$ ./configure --with-windows --prefix="/cygdrive/g/gcc_mingw/bin"
$ make clean
$ make 2> make.err
$ cp config.h config_ming.h
```
ngspice.exe is o.k.,but make tests does not work (cannot direct console output into file). Needs to add .save "what" "where.test" to every input (*.cir) file. Also all given output files have to be adapted to WINDOWS (CR/LF instead of only LF at each line ending) for allowing proper comparison.

31.2.6   make ngspice with CYGWIN and internal MINGW32 (use config.h made above)

```
$ cd ng-spice-rework
$ rm config.cache
$ export CFLAGS="-mno-cygwin -g -O2"
$ export LDFLAGS="-L/lib/mingw"
$ export CPPFLAGS="-I/usr/include/mingw"
$ ./configure --with-windows
$ cp config_ming.h config.h
$ make clean
$ make 2> make.err
```
`./configure` does not work correctly: It finds headers and libs which are not really available in the -mno-cygwin port of MINGW32. Therefore config.h is not o.k.

ToDo: find appropriate presets for variables ? rewrite tests for headers and libs (search exclusively in mingw directories)

31.3   Reporting errors

Setting up ngspice is a complex task. The source code contains over 1500 files. ngspice should run on various oprating systems. Therefore errors may be found, some still eveloving from the original spice3f5 code, others introduced during the ongoing code enhancements.

If you happen to experience an error during compilation of ngspice, please send a report to the development team. Ngspice is hosted on sourceforge, the preferred place to post a bug report is the ngspice bug tracker. We would prefer to have your bug tested against the actual source code available at CVS, but of course a report using the most recent ngspice release is welcome! Please provide the following information with your report:

Ngspice version

Operating system

Small input file to reproduce the bug (if to report a runtime error)

Actual output versus the expected output

Chapter 32

Copyrights and lincenses

32.1 Documentation licence

32.1.1 Spice documentation copyright

Copyright 1996 The Regents of the University of California.

Permission to use, copy, modify, and distribute this software and its documentation for educational, research and non-profit purposes, without fee, and without a written agreement is hereby granted, provided that the above copyright notice, this paragraph and the following three paragraphs appear in all copies. This software program and documentation are copyrighted by The Regents of the University of California. The software program and documentation are supplied "as is", without any accompanying services from The Regents. The Regents does not warrant that the operation of the program will be uninterrupted or error-free. The end-user understands that the program was developed for research purposes and is advised not to rely exclusively on the program for any reason.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTA-TION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSI-BILITY OF SUCH DAMAGE. THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PRO-VIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

32.1.2 XSPICE SOFTWARE USER'S MANUAL copyright

Copyright © 1992 Georgia Tech Research Corporation All Rights Reserved.
This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7013 (Oct. 1988)

32.1.3 CIDER RESEARCH SOFTWARE AGREEMENT

This chapter specifies the terms under which the CIDER software and documentation coming with the original distribution are provided.

Software is distributed as is, completely without warranty or service support. The University of California and its employ- ees are not liable for the condition or performance of the software.

The University does not warrant that it owns the copyright or other proprietary rights to all software and documentation provided under this agreement, notwithstanding any copyright notice, and shall not be liable for any infringement of copyright or proprietary rights brought by third parties against the recipient of the software and documentation provided under this agreement.

THE UNIVERSITY OF CALIFORNIA HEREBY DISCLAIMS ALL IMPLIED WARRANTIES, IN-CLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PAR-TICULAR PURPOSE. THE UNIVERSITY IS NOT LIABLE FOR ANY DAMAGES INCURRED BY

THE RECIPIENT IN USE OF THE SOFTWARE AND DOCUMENTATION, INCLUDING DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

The University of California grants the recipient the right to modify, copy, and redistribute the software and documentation, both within the recipient's organization and externally, subject to the following restrictions:

(a) The recipient agrees not to charge for the University of California code itself. The recipient may, however, charge for additions, extensions, or support.

(b) In any product based on the software, the recipient agrees to acknowledge the research group that developed the software. This acknowledgement shall appear in the product documentation.

(c) The recipient agrees to obey all U.S. Government restrictions governing redistribution or export of the software and documentation.

All BSD licenses have been changed to the "modfied" BSD license by UCB in 1999 (see chapt. 32.2.1).

32.2   ngspice license

The SPICE license is the **"Modified" BSD licence**, (see http://embedded.eecs.berkeley.edu/pubs/downloads/spice/index **ngspice adopts this "Modified" BSD licence as well.**
****

Copyright (c) 1985-1991 The Regents of the University of California.

All rights reserved.

Permission is hereby granted, without written agreement and without license or royalty fees, to use, copy, modify, and distribute this software and its documentation for any purpose, provided that the above copyright notice and the following two paragraphs appear in all copies of this software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVER-SITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO PROVIDE MAIN-TENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
****

32.2.1   "Modified" BSD license

All "old" BSD licenses (of SPICE or CIDER) have been changed to the "modified" BSD license according to the following publication (see ftp://ftp.cs.berkeley.edu/pub/4bsd/README.Impt.License.Change):

July 22, 1999

To All Licensees, Distributors of Any Version of BSD:

As you know, certain of the Berkeley Software Distribution ("BSD") source code files require that further distributions of products containing all or portions of the software, acknowledge within their advertising materials that such products contain software developed by UC Berkeley and its contributors.

Specifically, the provision reads:

" 3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors."

Effective immediately, licensees and distributors are no longer required to include the acknowledgement within advertising materials. Accordingly, the foregoing paragraph of those BSD Unix files containing it is hereby deleted in its entirety.

William Hoskins

Director, Office of Technology Licensing

University of California, Berkeley

32.2.2   Linking to GPLd libraries (e.g. readline):

The readline manual at http://tiswww.case.edu/php/chet/readline/rltop.html states: Readline is free software, distributed under the terms of the GNU General Public License, version 3. This means that if

you want to use Readline in a program that you release or distribute to anyone, the program must be free software and have a GPL-compatible license.

According to http://www.gnu.org/licenses/license-list.html, the **modified BSD license**, thus also the ngspice license, belong to the family of **GPL-Compatible Free Software Licenses**. Therefore the linking restrictions to readline, which have existed with the old BSD licence, are no longer in effect.